Deliverable
D3.4 AI Algorithms and Simulation Tools. Final

Project Acronym:	URBANAGE	
Project title:	Enhanced URBAN planning for AGE-friendly cities through disruptive technologies	
Grant Agreement No.	101004590	
Website:	www.urbanage.eu	
Version:	1.0	
Date:	31/01/2023	
Responsible Partner:	TEC	
Contributing Partners:	IMEC, UH	
Reviewers:	Julien Verplanken (IMEC) Sakis Dalianis (ATC)	
Dissemination Level:	Public	x
	Confidential – only consortium members and European Commission	

This project has received funding from the European Union’s Horizon 2020 research and innovation programme under grant agreement No 101004590

Revision History

Revision	Date	Author	Organization	Description
0.1	10/10/2023	Eneko Osaba	TEC	Initial version
0.2	17/01/2023	Andoni Aranguren	TEC	Ready for internal review
0.2	25/01/2023	Sakis Dalianis, Julien Verplanken	ATC, IMEC	Internal review
0.3	30/01/2023	Ibai Laña, Eneko Osaba	TEC	Updates from internal review
0.4	30/01/2023	Sergio Campos	TEC	Ready for final review
0.5	31/01/2023	Claudia Vicari, Giuseppe Ciulla	ENG	Internal review
1.0	31/01/2023	Sergio Campos	TEC	Final Version

Table of Contents

Table of Figures

List of Tables

List of abbreviations

Abbreviation	Explanation
AI	Artificial Intelligence
API	Application Programming Interface
APT	Amenity Projection Tool
ASF	Amenity Spread Function
CNN	Convolutional Neural Network
DEM	Digital Elevation Model
DL	Deep Learning
GPS	Global Positioning System
GTFS	General Transit Feed Specification
LIDAR	Light detection and ranging
ML	Machine Learning
OSM	Open Street Map
OSRM	Open Source Routing Machine
OTP	Open Trip Planner
POM	Project Object Model
SOET	Standardized OSM Enrichment Tool

1 Executive Summary

This document contains the description of the work done regarding the Artificial Intelligence (AI) Algorithms and Simulation Tools in the context of the second year of URBANAGE project. All the work detailed in this manuscript is mainly framed in task T3.2 – *AI Algorithms and simulations*, part of the WP3 – Data & Intelligence. In the same way, this deliverable is the evolved version of the previously D3.3 – *AI Algorithms and Simulation Tools. Initial*.

Thus, this document starts by briefly summarizing the work done, and framing the artificial intelligence and simulation tools within the whole URBANAGE architecture. In any case, **the central content of this manuscript is the detailed description of all the tools developed in the project, contextualizing them on the use case for which they have been primarily developed. Additionally, main novelties of each component are introduced for each component, as well as principal research innovations**. Finally, demonstrations of the functionalities developed will be shown for each tool implemented.

According to the Grant Agreement of the project, T3.2 finishes in M30. On this regard, even if this is the last deliverable purely dedicated to this task, further details on the evolution of the work done regarding *AI Algorithms and simulations* will be provided in the upcoming D3.7 - *Data & Intelligence framework* (delivered in M30).

2 Introduction

Nowadays, municipalities and city servants are fully aware of the need to include citizens for the design of valuable urban projects. Instead, although the importance of citizen engagement is demonstrated, it is complicated for urban planning to completely ensure the consideration and representation of all the citizen collectives in new developments. Additionally, it is still an open challenge to really engage citizens towards inclusive, sustainable, and scalable cities.

On another vein, many European cities are suffering a slow but progressive ageing of their populations, arising a considerable spectrum of new problems and concerns that should be taken into account. For this reason, and as a result of these concerns, policy makers and urban planners are in a constant search of novel initiatives and interventions for enhancing the participation of senior citizens.

In this context, Artificial Intelligence (AI) has found to be a promising knowledge area for dealing with ageing people related concerns. For this reason, a significant number of municipalities and cities have adopted AI solutions in their daily activity, implementing different systems for constructing innovative age-friendly functionalities. Some examples of tools that contribute to building a smart and inclusive city are route planners, smart information panels and interactive public transportation stops. In this context, simulation tools are also an excellent alternative for similar purposes. These tools have the ability of analyzing tentative scenarios in the city as a result of simulating actions such as the placement of new urban infrastructures.

In URBANAGE, a group of functionalities has been developed and embedded in the URBANAGE Ecosystem, contemplating systems such as routing, simulation or optimization of the different actions. All the developed functionalities have been tested over three different pilot cases placed in Santander (Spain), Helsinki (Finland) and the region of Flanders (Belgium). More specifically, the adequacy of each tool has been demonstrated on a single pilot case, even though we also demonstrate the replicability of each system in this same manuscript. It should be highlighted here that each age-friendly system developed for URBANAGE has been designed after the conduction of several participative workshops, assuring in this way that the implemented components could improve the life quality of ageing citizens. Moreover, developed tools are used by citizens, policy makers and/or urban planners.

This manuscript is the second and last version of the *AI Algorithms and Simulation Tools* deliverable, contemplating the work carried out in the second year on the WP3 - *Data & Intelligence*. **Thus, the principal goal of this document is to describe in depth all the components implemented, spotlighting their main characteristics, applicability, replicability, and innovative level. In fact, this is the main contribution regarding the previous version of this deliverable.**

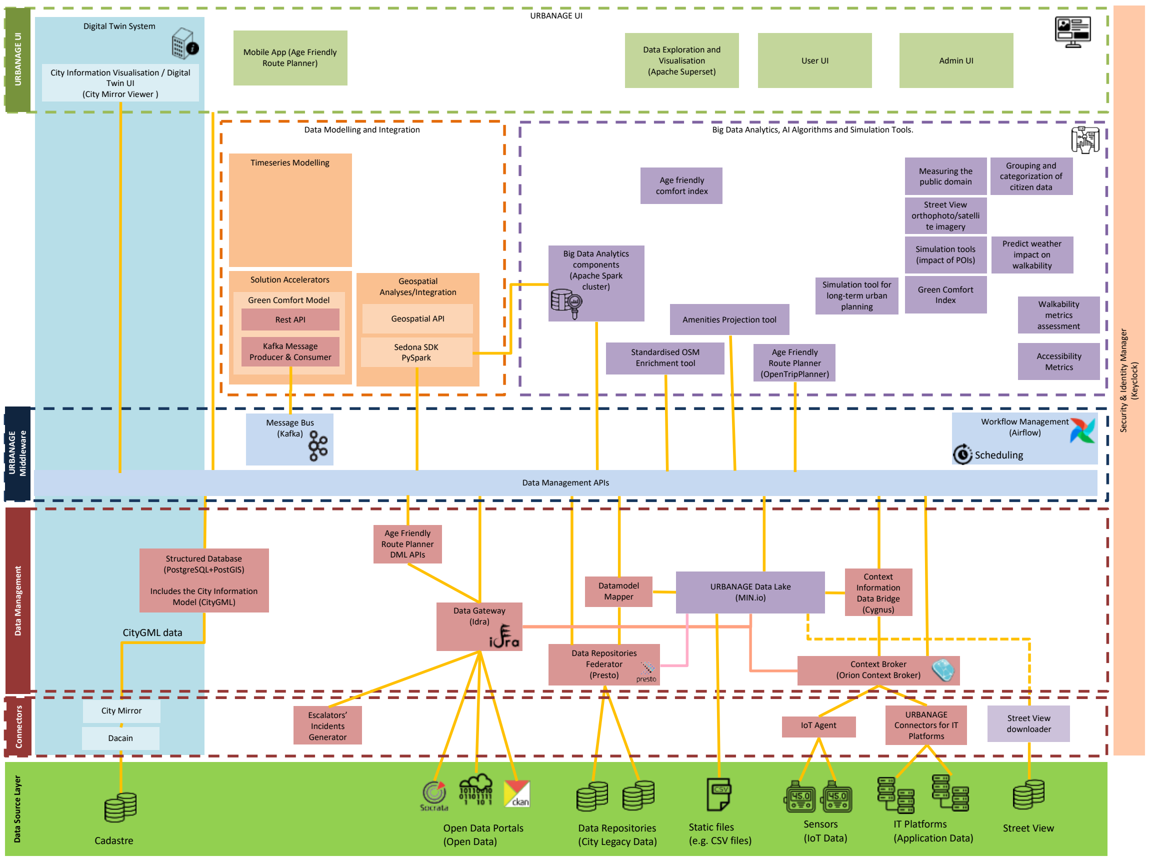
Finally, this deliverable is organized into four additional sections. Section 3 consists of a general description of the work conducted on the topics dealt in this deliverable. Section 4 is devoted to describing in detail all the tools implemented. Section 5 presents the conclusions and comments for the future steps until the end of the project, while Section 6 represents the references cited in this deliverable.

3 AI algorithms and simulation in URBANAGE

The principal scope of the task T3.2 is to design, implement and deploy a group of AI and data-science related methods with the main objectives of *i)* providing means to support decision making duties for policy makers and urban planners; and ii) offering valuable and fully usable mechanisms and tools for aiding ageing citizens in the conduction of concrete daily routines, such as going to any health care center or attending an event in a civic center.

All the work conducted in the T3.2 and summarized both in the present manuscript and in its previous version D3.3 can be placed in the module named as *Data Analytics*, and part of the complete URBANAGE architecture. For properly place the Data Analytics module, we depict in Figure 1 the conceptual design of the URBANAGE architecture. We refer interested readers to Deliverable D3.3 for further detail on each of the *Data Analytics* submodules.

Figure 1: URBANAGE platform overview


URBANAGE UI
Digital Twin System
City Information Visualisation / Digital Twin UI (City Mirror Viewer)
Mobile App (Age Friendly Route Planner)
URBANAGE UI
Data Exploration and Visualisation (Apache Superset)
User UI
Admin UI
Data Modelling and Integration
Timeseries Modelling
Solution Accelerators
Green Comfort Model
Rest API
Kafka Message Producer & Consumer
Geospatial Analyses/Integration
Geospatial API
Sedona SDK PySpark
Big Data Analytics, AI Algorithms and Simulation Tools.
Age friendly comfort index
Measuring the public domain
Grouping and categorization of citizen data
Street View orthophoto/satellite imagery
Big Data Analytics components (Apache Spark cluster)
Simulation tools (impact of POIs)
Predict weather impact on walkability
Simulation tool for long-term urban planning
Green Comfort Index
Walkability metrics assessment
Amenities Projection tool
Standardised OSM Enrichment tool
Age Friendly Route Planner (OpenTripPlanner)
Accessibility Metrics
Security & Identity Manager (Keycloak)
URBANAGE Middleware
Message Bus (Kafka)
Workflow Management (Airflow)
Scheduling
Data Management APIs
Data Management
Structured Database (PostgreSQL+PostGIS)
Includes the City Information Model (CityGML)
Age Friendly Route Planner DML APIs
CityGML data
Data Gateway (Idra)
Datamodel Mapper
URBANAGE Data Lake (MIN.io)
Context Information Data Bridge (Cygnus)
Data Repositories Federator (Presto)
Context Broker (Orion Context Broker)
Connectors
City Mirror
Dacain
Escalators' Incidents Generator
IoT Agent
URBANAGE Connectors for IT Platforms
Street View downloader
Data Source Layer
Cadastre
Open Data Portals (Open Data)
Data Repositories (City Legacy Data)
Static files (e.g. CSV files)
Sensors (IoT Data)
IT Platforms (Application Data)
Street View

On this line, **task T3.2 is completely devoted to the design and implementation of AI systems and tools that use the information and data gathered and provided by the *Data Management* module** (developed in T3.1

of URBANAGE). These models and tools **can offer advanced functionalities for an evidence-based decision-making process**.

Furthermore, **the work conducted in task T3.2 has also contemplated the development of simulation engines that allow policymakers and urban-planners to perform physic-based simulations**. These simulation tools are valuable for providing service provisioning and urban planning activities, offering evaluations of potential changes in the city, and measuring the situations of public services and infrastructures.

Lastly, it is interesting to mention here that the outputs provided by all the tools of the *Data Analytics* module are used as input in the *City Information Visualizer* module, which is also part of the complete URBANAGE system. In that module, different visualizers will be developed in order to properly provide the end users with all the information calculated by AI tools in a friendly way.

4 AI functionalities per URBANAGE use case

In this section the principal AI functionalities developed in the context of URBANAGE project are described. For each tool, different aspects such as the functional description, data used, main novelties and innovations, installation instructions, system architecture or user manual are detailed, among many other aspects. Furthermore, for each tool, a demonstration is included for showing their principal features, contextualized in, at least, one pilot case considered in the project. The code repository collects the prototypes of the main components (e.g., baseline tools, libraries, datasets, etc.) constituting the Artificial Intelligence and their future developments. The deliverable "D5.2 Initial Platform Prototype" provides details about the CI/CD process and the code repository.

4.1 Age-friendly route planner

4.1.1 Functional Description of the component

Today, route planning is a hot topic in urban planning and in the scientific community. The reason for this popularity can be broken down into two distinct factors. On the one hand, due to their complexity, it is a challenge to try to solve this type of problems. Hence the **inherent scientific appeal of these problems is irrevocable**. On the other hand, the business benefits of efficient logistics, and the social advantages that this would bring, make the treatment and resolution of these problems of **great social interest**.

Evidence of this interest is the growing number of scientific publications that are added to the literature year after year, as can be seen in papers such as [1] and [2]. **Even more interesting is the number of open-source frameworks for route planning that can be found in the community**, which can be used for solving routing problems of different kind.

In line with this, as can be read in the project General Agreement, in the description related to T3.2 - *AI algorithms and simulation*: *"In order to implement the necessary capabilities, Task 3.2 will reuse existing AI, ML lifecycle and ETL frameworks and assets that will be customized and trained to fit project objectives"*.

For this reason, and following this main philosophy, in URBANAGE the legal capacities of modifying and adapting an existing routing framework has been considered, using not only an existing and well-established platform, but also widely known standards (such as Open Street Maps, OSM). Therefore, multiple **free and open-source** route planners have been analyzed with objective of finding the specific framework that satisfies the capabilities that apply to URBANAGE project needs. The alternative route planners, their description and characteristics are described in the following list and Table 1:

- **Google Maps:** This multi-modal route planner is the most known planner. Unfortunately, the software is not open source. Anyhow, it is the baseline to which other routing softwares will always be compared to, because of how well-known Google Maps is. Like such, most of its features are considered mandatory for the average user. These are multi-modal route planning, alternative routing and address geocoding. Even though Google Maps is a great tool used daily by millions of users, it still lacks accessibility features such as wheelchair accessible routing, low incline routes, comfortability or routing through funiculars and conveying stairs. All these lacking features combined with the fact that no open-source version is available makes it a rather unfriendly route planner, despite of its real-time GTFS and address geocoding features.

- **Open Trip Planner (OTP):** As can be read in its web page, *OTP is an open-source platform for multi-modal and multi-agency journey planning. It follows a client-server model, providing several map-based web interfaces as well as a REST API for use by third-party applications. OTP relies on open data standards including GTFS for transit and OpenStreetMap for street networks. OTP deployments now exist around the world and OTP is also the routing engine behind several popular smartphone applications.* The base code used for OTP has been written in JAVA programming language, and it can be obtained from the GitHub of Open Trip Planner Project[1]. This project not only contains the complete base code of OTP, which can be used and modified, but also a client with testing purposes. As summary, as can be read in the GitHub, the downloaded project *"includes a REST API for journey planning as well as a map-based Javascript client. OpenTripPlanner can also create travel time contour visualizations and compute accessibility indicators for planning and research applications.* Since its creation, OTP has demonstrated to be an extremely adaptable tool, which has the main advantage of easily working with open standards. **For URBANAGE, OTP has been used as base framework, because of the reasons explained later in this section**.

- **GraphHopper:** As can be read in Wikipedia, '*GraphHopper is an open-source routing library and server written in Java and provides a web interface called GraphHopper Maps as well as a routing API over HTTP. It runs on the server, desktop, Android, iOS or Raspberry Pi. By default, OpenStreetMap data for the road network and elevation data from the Shuttle Radar Topography Mission is used*.' And also, the following mindset can be found in their webpage '*Our flagships are the GraphHopper routing engine and jsprit, the toolkit for solving rich vehicle routing problems*.' Having said that, GraphHopper's solutions are mainly focused on continental or inner-city vehicle routing optimization, which does not apply to age-friendly route planning that relies on pedestrian routes. It can be considered as a great disadvantage for being considered in URBANAGE.

- **Open-Source Routing Machine (OSRM):** OSRM is a C++ implementation of a high-performance routing engine for shortest paths in road networks and can be used as a library (libosrm) via C++ instead of using it through the HTTP interface and osrm-routed. It uses open free network data from OSM, and it focuses on speed for shortest path computation on a continental sized network. It also

[1] https://github.com/opentripplanner/ OpenTripPlanner

accepts walking and biking as traversing modes. In any case, OSRM lacks GTFS data for public transport and its main goal being continental travel computation, this software drifts off the use case's objective.

- **Route XL:** As can be read in their webpage, *RouteXL is a road route planner for multiple destinations that finds the best multi-stop route for deliveries, pickups and services with smart algorithm that sorts addresses to minimize the overall duration of the route*. Also, at the bottom of their webpage they state that *the routes are calculated with Graphhopper, OSRM and Gosmore* (an unmaintained router and viewer from 2011). Taking GraphHopper's and OSRM's continental approach in mind, it does not seem to fit this projects interest, also because its inner-city application is mainly deliveries and pickups.

- **Mapotempo:** This router and fleet management software is open source, but not offered as free. As can be read in their webpage, '*Our solutions enable you to manage routing problems, regardless of the type of service you offer (delivery, collection, maintenance, sales, medical check-up, etc.), your company's activity and the type of fleet (light and heavy vehicle, bicycle, pedestrian, electric scooter, etc.).*' This description clearly states that their software's main features surround fleet management, which doesn't match with this project goal.

- **OpenRoute Service:** This software tool has a crowdsourced API that gets its data from OSM, which provides three types of planning tools: classic maps, disaster maps and new VueJS clients. From all these three tools only the first one is appliable in URBANAGE. The classic map router has multiple modes of transportation, and it provides global coverage with road type restrictions. Additionally, the Geocoding API converts address and streets into geocoded coordinates.

- **Traccar:** as can be read in Wikipedia, *Traccar is a free and open-source GPS tracking server*. According to a 2019 report by Windows Report, *Traccar was noted amongst the seven best free software solutions for fleet management and GPS tracking*. It has lots of different features like driver behavior monitoring, detailed and summary reports, geofencing functionality, alarms and notifications, account and device management and email and SMS support. Unluckily, as many of the other routing softwares, its features are mainly design for fleet management which deviates from URBANAGE's goals.

- **OptaPlanner:** as can be read in their webpage, *OptaPlanner is an open-source solver that optimizes plans and schedules with hard constraints and soft constraints; it reduces costs substantially, improves service quality, fulfills employee wishes and lowers carbon emissions*. Its main features are vehicle routing, employee rostering, maintenance scheduling, conference scheduling, school timetabling, task assignment, cloud optimization and job shop scheduling. This software has an interesting goal of leveraging employer satisfaction, but it limits itself in the enterprise and its route planner is designed for vehicles, which is not the route planner URBANAGE needs.

- **Open Door Logistics:** as can be read in Wikipedia, *Open Door Logistics Studio is a standalone open-source application for performing (a) geographic analysis of customers, (b) sales territory design, mapping and management and (c) 'last mile' transport planning using the jsprit vehicle routing toolkit*. Its features are designed for territory design, management and mapping which help with vehicle fleet planning. Therefore, this software does not provide a good baseline of features to develop a route planner for aging citizens.

Table 1: Free and open-source route planners (based on goodfirms.co).

Software	Open Source	Free	Core Features
Google Maps	No	No	GTFS integration, multi-modal trip planning, real-time alerts and reports, geocoding
Open Trip Planner (OTP)	Yes	Yes	Multi-modal trip planning, Real-time Alert and Reporting
GraphHopper	Yes	Yes	Vehicle Routing, Navigation, GPS Tracking, Cloud-based Online Maps
Open Source Routing Machine	Yes	Yes	Global Coverage, Multi-modal trip planning, Multi-stop Routing
Route XL	Yes	Yes	Multi-stop Routing, Smart Algorithms, Import Address
Mapotempo	Yes	No	Route Simulation, Vehicle Maintenance, GPS Tracking
OpenRoute Service	Yes	No	Vehicle Routing, Global Coverage, Disaster Management
Traccar	Yes	Yes	GPS Tracking, Route Management, Real-time Alert and Reporting
OptaPlanner	Yes	No	Route Optimization, Fleet Scheduling, Vehicle Maintenance
Open Door Logistics	Yes	Yes	Vehicle Routing, Territory Design, Transport Management

On another vein, usually, **routing algorithms are developed for a general purpose, which means that certain groups, such as ageing people, are often marginalized because of the broad approach of their designs**. Routing algorithms aim optimize efficiency factors: traverse speed, distance, and public transport transfers. But these factors might still conclude on a route that challenges underrepresented groups with specific physical necessities such as periodic resting, hydration to prevent heat strokes and incontinence. These groups, traditionally ageing people and physically divergent, require route planers with a different approach that integrates accessibility factors. The main objective is to build comfortable routes, instead of simply efficient routes.

With this motivation in mind, **the principal objective of the Age-Friendly Route Planner is to provide senior citizen with the friendliest routes. The goal of these routes is to improve the experience in the city for this ageing users**. In order to measure this friendliness, several variables are considered: such as the number of amenities along the route, the number of elements that improve the comfortability of the user along the path, the usage of urban infrastructures or the consideration of flat streets instead of sloppy sections.

As will be explained later, in Section 4.1.2, **for calculating friendly routes, the tool implemented in URBANAGE is based on the well-known Open Trip Planner (OTP[2]) framework, which has allowed us to modify and adapt it in order to meet all the requirements needed in the project**. This way, the solution developed permits the user to calculate routes conducted by walking and using public transportation. Also, it allows the building of routes considering wheelchair users. Furthermore, **for each petition, the planner returns different alternative routes, based on several criteria that are described along the following subsections**. In a nutshell, routes enhanced by AI algorithms, equilibrated routes and paths considering the spread of amenities through the use of flexible mathematical formulas have been considered for this Age-Friendly Route Planner.

Moreover, the Age-Friendly route planning has been adapted to the particular needs of each *customized group*. These groups are considered in order to take into account the diversity of needs and requirements in this segment of population: mobility and cognitive aspects, impaired abilities, among others. For this purpose, the API of the base OTP has been extensively modified in order to allow ageing citizens to adapt the routes to their particular situations.

Finally, information about the most popular routes together with the number of users on those routes is of great value for policymakers and urban planners in order to provide a better service to the citizen, e.g. organize the schedule of civil works in order to minimize the number of citizens affected, selection of activity places that less affect the senior citizens routes. The information about the most popular streets and routes will be calculated in URBANAGE Project by analysing the historic usage of the Age-Friendly route planner.

In the following sections, how the Age-Friendly route planner works, and its main functionalities are deeply described, along with the specific details about its implementation and installation. Also, a sort of user manual is included detailing the steps to follow for properly using the tool.

4.1.2 Technical Description of the component

4.1.2.1 Justification of the optimization framework used.

After analysing a significant amount of the most well-known route planners, what can be seen is that most routing softwares are designed for vehicles. This fact accentuates the need of a solution that has citizens as the primary target. **With this in mind, OTP has been selected as the framework for being used in the work conducted in URBANAGE T3.2. This does not imply that the advantages of the other alternatives should be underestimated, but the main features, flexibility, and advantages that OTP provides to developers has led URBANAGE working team to choosing it as an excellent platform for reaching the main objectives established in the project**. Going deeper, several key reasons have encouraged us for considering OTP for the project:

[2] https://www.opentripplanner.org/

- It is fully open source, meaning that it can be fully adapted to the URBANAGE Requirements.
- Both the calling API and the outcome JSON are fully adaptable for the URBANAGE needs.
- It efficiently works with widely known standards such as OSM (for building the map), GTFS (for defining the public transport services) or Geotiff (for defining the altitudes of the city), among others.
- It is written in JAVA, a language in which the working group of T3.2 has significant expertise.
- Having originally been published in 2009, OTP is a platform with a long trajectory. Thanks to this, it is very well documented, and it has a wide and active community working on it. This facilitates the understanding of the framework.

4.1.2.2 Architecture of the component (describing also the data used)

In a nutshell, the Age-Friendly route planning system has a central module, coined as route calculator, which is the one in charge of calculating the routes using as input both the data available and the input information of the user. As explained in previous section, this route calculator is based on the widely known OTP. In Figure 2, we represent the overall architecture of the Age-Friendly route planner. This picture also includes the input data and the ad-hoc tools implemented for gathering this data.
To adopt for all the requirements that the routing system should adhere to, the data sources mentioned in the following subsections have been embedded into the OTP platform.

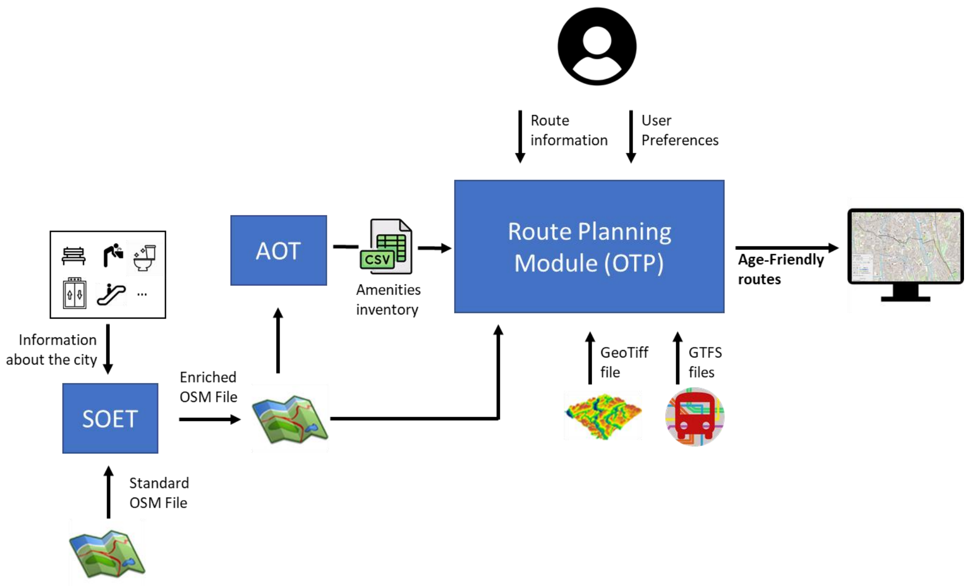


Figure 2: overall architecture of the Age-Friendly route planner

Furthermore, we have summarized the different datasets used in the following table:

Table 2: Input datasets for Age-friendly route planner

Datasource	Extension	Description
Open Street File	.osm	This file is the map itself, containing information about the street roads and amenities.
Amenities inventory	.csv	This file contains information about amenities and its geographical place in the city.
GTFS	.zip	These files contain all the information regarding the public transportation of the city.
Elevations	.tif	This file contains the elevation of the city, and it is needed for considering slopes in the planned routes.

4.1.2.2.1 Open Street Map files

For adequately generating the optimal trajectory, the route planner needs to be aware of the complete street network. **To this end, we have used the corresponding OSM map files from the cities at hand (Santander, Helsinki and Ghent)**, obtained for the *Planet OSM* [3] open platform using *BBBike*[4] functionality. Using this tool, all maps have been obtained in .osm format, containing all nodes, streets and relations for deploying the map. This format is completely compatible with OTP, which consumes the files automatically, building a representation of the corresponding road network.

Furthermore, this OSM file also includes important elements for the planner such as elevators, benches, fountains, toilets, and automatic ramps. In line with this, it should be highlighted that, usually, the **OSM files that can be openly obtained from open platforms are not as specific as URBANAGE needs for this last purpose. OSM open files have demonstrated a great efficiency for routing, but in terms of amenity related content, they are not exhaustive, because their completeness depends on the community effort**.

Because the amenities that are spread over the city are crucial for the route planner for building comfortable and adequate routes, **in the context of URBANAGE we have developed an ad-hoc tool for the enrichment of the Standardized OSM files**. We have coined this tool as **Standardized OSM Enrichment Tool, or SOET**.

4.1.2.2.1.1 Standardized OSM Enhanced Tool – SOET

OSM is a great tool that provides open data of both routes and amenities. The data collection is based on user input, so specific information such as amenities is usually outdated or missing. Furthermore, routing solutions usually require not only the data from OSM, but also external data which could be private for licensing or privacy reasons.

[3] http://planet.osm.org

[4] http://download.bbbike.org/osm

In the use cases of URBANAGE, it is necessary to contemplate the amenities that are spread over the city, this issue being a crucial one for the success of the Age-Friendly route planner. For this reason, in the specific case of Santander, for example, the city council has provided a series of files that contained a list of amenities with their geospatial information. Among these amenities, we find benches, drinkable water fountains, handrails, and toilets. Because the data was not provided in OSM format, it needed to be preprocessed in order to be consumed. **With this motivation, a standardized and easy to use OSM data enrichment script has been developed in the context of URBANAGE, coined as Standardized OSM Enrichment Tool, or SOET.**

A data enrichment solution has been chosen for this purpose over other options. Furthermore, being OSM the base for the map building in OTP, the SOET provides a cascade effect for all applications that are based on the OSM file. Some examples are data visualization in OTP and, also, the **amenity projection tool, or APT** (described in more detail in Section Amenities file). In Figure 3: Main architecture of SOET, we represent the main architecture of SOET.

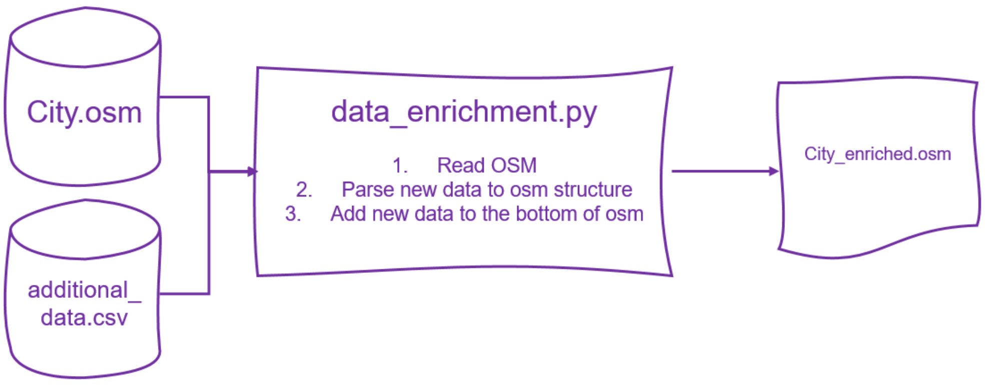


Figure 3: Main architecture of SOET

The required files for the Python based SOET is the OSM file, in which the data is going to be stored, and a csv file containing the amenities that are going to be loaded. This csv should follow the format depicted in Table 3. This is the information which should be deposited in this csv.

- *Mandatory data*: longitude and latitude of the amenity, and its type (amenity column).
- *Optional*: Any column-row information can be added as a node tag, source or date. Despite this data is not used for the enrichment, it could be important to contextualize the information.

Table 3: Datasets for the SOET component

Source.Name	Lon	Lat	amenity	source	source:date
str the name of the file that has been used (it's not added to osm it's	*Float*	*Float*	*str tag that should follow osm value conventions*	*str of who has provided de data*	*str of the date when the data was provided*

recorded as QoL info)					
AparcamientosBici.csv	-3,78	43,47	bicycle_parking	Ayuntamiento de Santander	18/06/2021

In the following figures we can see a clear example of an OSM file before the enrichment (Figure 4), and the result of the application of SOET (Figure 5). In Figure 5 we can see the newly added elements: benches (pink points), drinking fountains (blue points) and garbage cans (red points).

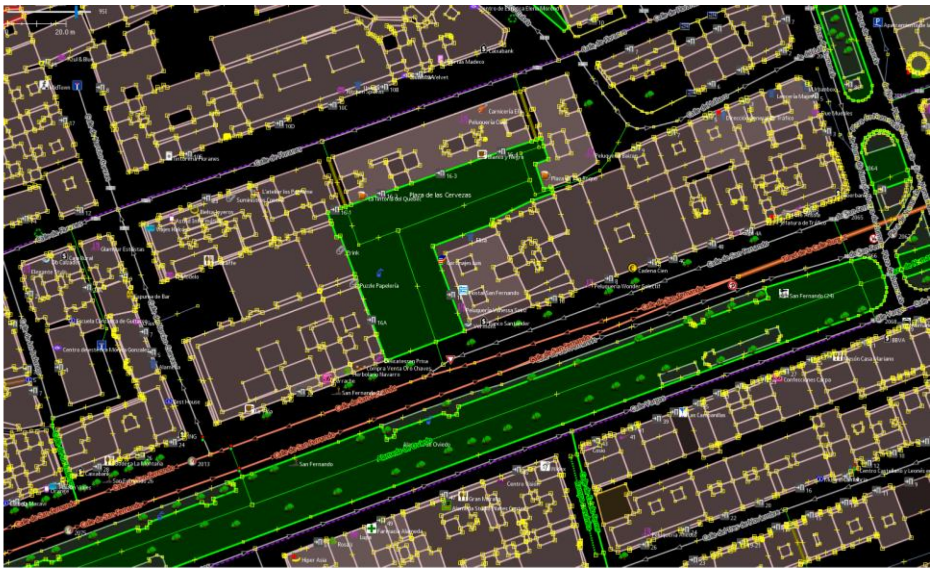


Figure 4: A map excerpt of Santander before the SOET application

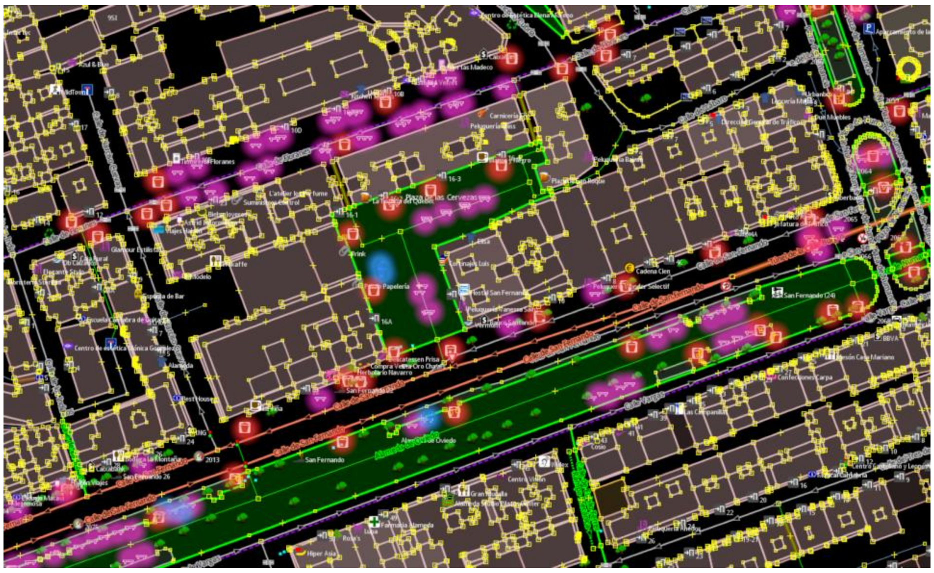


Figure 5: A map excerpt of Santander after the SOET application

4.1.2.2.2 Amenities file

One of the principal innovations of the Age-Friendly route planner is the consideration of the amenities spread over the city for calculating comfortable routes. At this moment, four different main amenities have been considered in the route calculation: public toilets, benches, handrails and drinking fountains. All these amenities have been extracted from the OSM maps, and for doing that, a new tool has been developed in the context of the URBANAGE project. We have coined this tool as **Amenity Projection Tool (APT)**.

4.1.2.2.2.1 APT - Amenity Projection Tool

The Age-Friendly route planner has the possibility of building routes based on the amenities that the user can find along the path. For this reason, a routing platform based on the amenity availability must model the correlation between the route segments and the close amenities. Luckily, both segment and amenity data are openly available in the OSM file, which describes urban areas as graphs and nodes. Therefore, for properly considering these interesting elements, the routing planner should correlate these two information sources: amenities and the city map. To achieve this feature a python script has been developed, coined as the **amenity projection tool (APT)**. In Figure 6, we represent the architecture of this tool.

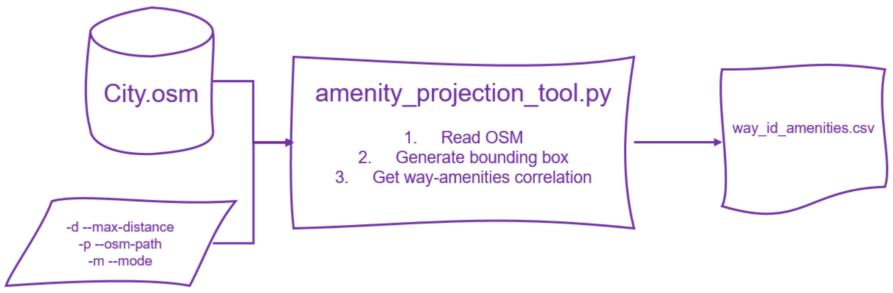


Figure 6: Architecture of the APT

The APT is developed to correlate route segments (ways) and amenities (nodes). But it has also been developed for taking a replicable, efficient and customizable paradigm. The customization is particularly interesting so different circumstances can be considered such as handrails that should be at a maximum of 5 meters or benches and drinking water that could be at 20 meters away from the way. Thus, the algorithm developed in Python requires an OSM file and a distance parameter, in meters, so it can output a *.csv* file with the correlation between *wayid* and amenity amounts that are as close as defined by the distance provided. In

The APT starts by reading the OSM file and then creates a bounding box that encloses each route. This bounding box should be big enough to include close nodes (in Figure 7 it is depicted as *loose bbox*), so it is bigger than the minimum *bounding box* by maximum distance (represented as minimum box in Figure 7). This bounding box is created so it reduces the amount of correlation calculation from $O(N_t * M_t)$, where N_t and M_t are the total amount of amenities and way node respectively, to the bare minimum by adding a $O(M_t)$ preprocess resulting in a $O\left(N_p * M_p + N_t\right) = O(N_p * M_p)$, where N_p and M_p are a partial amount of nodes and way nodes which are also smaller or equal to N_t and M_t. This approach has been chosen such that the algorithm would be efficient. The Figure 7, we represent this situation graphically, the green amenities have been considered close enough to the street in order to correlate these amenities with the road.

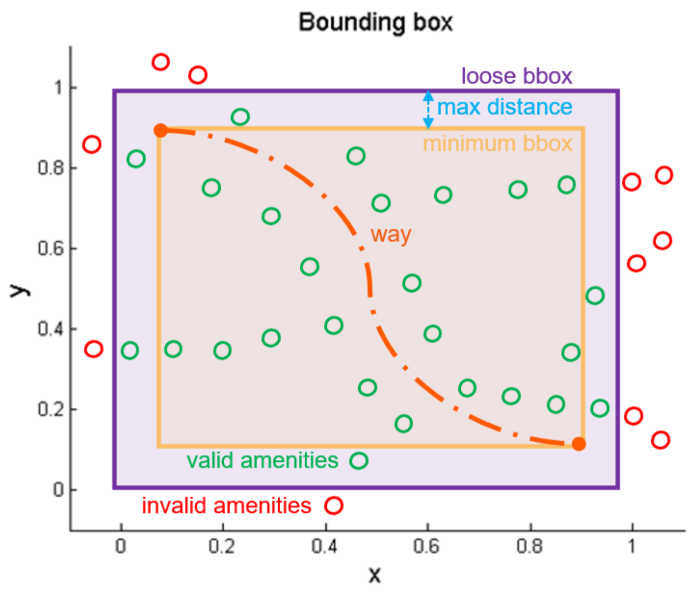


Figure 7: Amenity Projection Tool criteria for considering amenities.

After this first step, each amenity node (point) inside the bounding box is projected to each road segment (line) with an orthographic projection. From this projection multiple measurements can be extracted but only one is currently considered, the distance between the point and its projection. This distance defines how far the amenity is from the segment and if it is equal or smaller than the provided maximum distance parameter, the amenity is added to its amenity-type count. Additionally, the projection factor (ratio) could be extracted and could be used to accurately define where in the segment the node is but for our purposes only the amounts are required and collected. Figure 8, represents this situation.

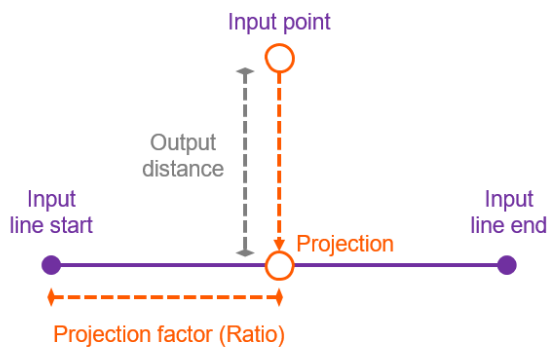
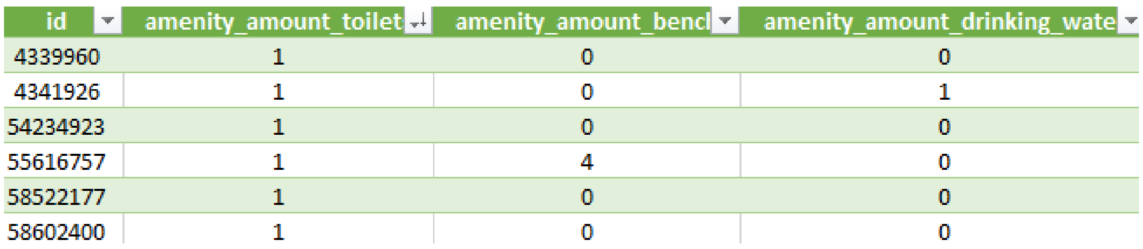


Figure 8: How the APT relates amenities and ways

As previously stated, all these correlations are stored in a *.csv* that has the *wayid* of each way as index and multiple columns with the number of amenities of that type that are located within the parametrized maximum distance from the road. We depict in Figure 9 an example of AOT output.



id	amenity_amount_toilet	amenity_amount_bencl	amenity_amount_drinking_wate
4339960	1	0	0
4341926	1	0	1
54234923	1	0	0
55616757	1	4	0
58522177	1	0	0
58602400	1	0	0

Figure 9: An example of AOT output relating wayids with benches, toilets and drinking fountains.

The replicability of this tool is achieved simply by using publicly available OSM data and providing a problem agnostic script that only requires the OSM file. OSM contains information for cities all around the world and it is properly standardized, so it doesn't need any additional tunning. Lastly, the usage of this tool can be enhanced if the OSM introduced as input is the result of the **Standardized OSM Enrichment Tool**, or SOET, specifically designed for this project, and described in section 4.1.2.2.1.1. As such, the number of amenities collected can be higher than the ones available in the open OSM files. This is how we have proceeded in URBANAGE.

4.1.2.2.3 GTFS files for Public Transportation

In a nutshell and resorting to the official definition that can be found in the Google Transit APIs webpage[5], the General Transit Feed Specification, or GTFS, *defines a common format for public transportation schedules and associated geographic information. GTFS "feeds" let public transit agencies publish their transit data and developers write applications that consume that data in an interoperable way*. In other words, **thanks to GTFS files, a route planner can obtain all the information about the public transport network of a city, including routes, schedules, or a detailed itinerary of each trip, among other valuable information**. As explained in the previous section, **OTP can automatically read GTFS files for subsequently be able to calculate routes that include public transportation**. Going deeper into its format, a correct GTFS should be composed by several files:

- *Agency*: a file mentioning the main agencies that provide the public transportation.
- *Stops*: specific locations of public transportation stops.
- *Routes*: a file containing the available routes. A route is composed of a group of *Trips*.
- *Trip*: this file includes all the trips, being a trip a sequence of stops that occur at a concrete time.
- *Stop Times*: specific times in which a vehicle arrives and depart from a concrete stop. Each time is associated to a specific *Trip*.
- *Calendar*: this file contains the specific dates in which a route is available. It is depicted using a weekly schedule, and also allows the user to introduce specific date ranges.
- *Calendar dates*: it is used for describing exceptions in which a route is not available.
- *Fare attributes*: information about the fares of using each public transportation service.
- *Fare rules*: this complementary file permits to introduce rules about the tariffication of the services.
- *Shapes*: this file has a purely aesthetic purpose, and it is used for correctly drawing a route into a map.

[5] https://developers.google.com/transit/gtfs/

Thus, thanks to the GTFS, the Age-Friendly route planner is able to calculate routes that take into account public transportation services. In the specific context of URBANAGE, three different GTFS data have been employed: the one related to Santander and openly available in *Datos Abiertos Santander*[6], and the ones related to Helsinki[7] and Flanders[8], both of them have also been obtained from the Transit Feed platform.

4.1.2.2.4 Elevation Data

In order to calculate friendly routes for ageing citizens, **the Age-Friendly route planner considers the inclination of the streets as one crucial aspect, deeming that steep streets are preferably avoided**. For this purpose, an elevation map of the city is compulsory. More specifically, **OTP allows the introduction of this information using the widely known GeoTIFF metadata standard**. GeoTIFF permits the use of different georeferenced information embedded into a *.tif* file. **With such a file, OTP can assign to a street its corresponding elevation**. In the case of URBANAGE, all .tif files have been obtained from the *SRTM 90m Digital Elevation Database*[9] open platform.

4.1.2.3 Technical specifications

About the technological specifications of the Age-Friendly route planner, **it should be highlighted that it has been completely developed in JAVA, the language in which OTP is implemented.** In few words, JAVA is a class-based, object-oriented, general-purpose, and high-level programming language. The main philosophy of JAVA is to need as few dependencies as possible, making its use more efficient. JAVA is based on the principle known as WORA, which mean *Write Once and Run Anywhere*. Thanks to WORA philosophy, a compiled JAVA project can be executed in any platform that supports JAVA not requiring any recompilation.

Furthermore, **the project can be also deployed using the MAVEN mechanism, which is a software project management and comprehension tool**. MAVEN is based on Project Object Model (POM) concept. Thanks to this concept, MAVEN can manage the building, reporting and documentation of a project needing a unique file.

Regarding **other technical aspects for the Age-Friendly route planner**, we can provide the following summary:

- The format for building the street map is **OSM**.
- The public transportation information is provided using **GTFS** open standard.
- For the elevations, **GeoTIFF** metadata standard is employed.
- Both SOET and APT have been developed using **Python** language.
- Information about the amenities is provided to OTP in **CSV** format.

[6] http://datos.santander.es/dataset/?id=programacion-tus
[7] https://transitfeeds.com/p/helsinki-regional-transport/735
[8] https://transitfeeds.com/p/vlaamse-vervoersmaatschappij-de-lijn/530
[9] https://cgiarcsi.community/data/srtm-90m-digital-elevation-database-v4-1/

- The Age-Friendly route planer is called using the **REST-API** philosophy.
- OTP returns the results using **JSON** format.

4.1.3 Technical Contribution of the Component and Demonstration of the main functionalities

In this section, the main technical contribution of the Age-Friendly route planner is described. **Along the section, we will detail each functionality accompanied with several demonstrations for each of them and highlighting the technical aspects surrounding each feature. This helps us to clearly highlight the main innovations of our developed Age-Friendly route planner**.

In few words, **the Age-Friendly route planner allows the user to build routes of two different nature: public transportation and walking routes. These alternatives can also be planned for people in wheelchair, and for citizens without this specifical need**. Additionally, users of the route planner will be able to set some preferences regarding the importance of slopes, duration, amenities and comfortability on the calculation of the route. Further preferences are also available, which will be depicted along this section. In the following table we summarize the type of routes that can be calculated by the user.

Table 4: Possible routes to be calculated. These routes can also be conducted by users in wheelchair.

Route	Description
Preferences based route	A walking route that considers the preferences of the user in terms of quickness, slope, comfortability and amenities found.
Balanced route	A walking route that leverages all the preferences.
AI-enhanced route	T A walking route that uses a differential evolution algorithm for calculating an improved route.
Public transportation routes	A route that uses the public transportation available in the city.
ASF-route	A walking route that ensures the regular distribution of amenities along the path.

It should be pointed here that, **in order to highlight the replicability of the Age-Friendly route planner, the demonstrations shown along these sections will contemplate three different cities involved in URBANAGE project**: Santander, Helsinki and Ghent.

4.1.3.1 Walking routes

The Age-Friendly route planner will allow the users to calculate route for pedestrians. More specifically, the Age-Friendly route planner provides up four different alternative routes each time a citizen asks for this kind of routes.

4.1.3.1.1 Preference based route

The first interesting functionality implemented for the Age-Friendly route planner is the *Square Optimization*, which is a fundamental feature for the first of the walking routes provided by the planner. In the basic version of the OTP, a *Triangle Optimization* is available, which is applied for paths partially or completely made by bikes. This kind of optimization allows the user the definition of three different preferences for the calculation of this bike related routes:

- **Slope**: this factor regards the inclination of the route. The higher this factor, the flattener the routes calculated by the planner. The streets inclination is calculated using the elevation data described in section 4.1.2.2.4.
- **Duration**: this factor regards the duration of the route. The higher this factor, the shorter the routes calculated in terms of time.
- **Safety**: This factor is related to the safety of the bikers. This measure is calculated based on factors such as the inclination or the kind of roads.

For informative purposes, we depict in Figure 10, a visual example of how this *Triangle Optimization* is introduced in the testing purposes webpage provided by the basic OTP.

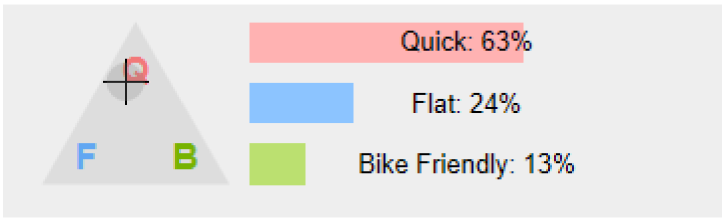


Figure 10: A graphic example of the Triangle Optimization in the Basic OTP for bike routes..

On this regard, **two different improvements have been implemented for the Age-Friendly route planner developed in the URBANAGE project context**. On the one hand, the first one is the **extension of the *Triangle Optimization* to a *Square Optimization***, not only incrementing the number of factors by one, but contemplating two factors which fit the main requirements of URBANAGE. We depict in Figure 11a visual example of our newly developed *Square Optimization*. **It is important to highlight here that the sum of all preferences should be equal to 100**.

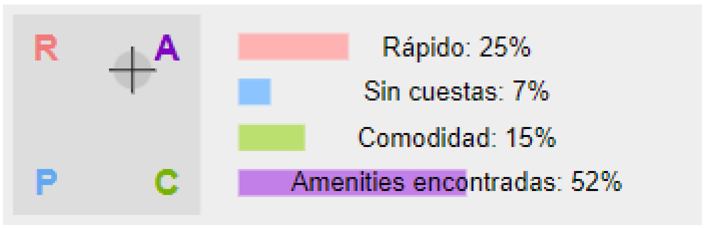


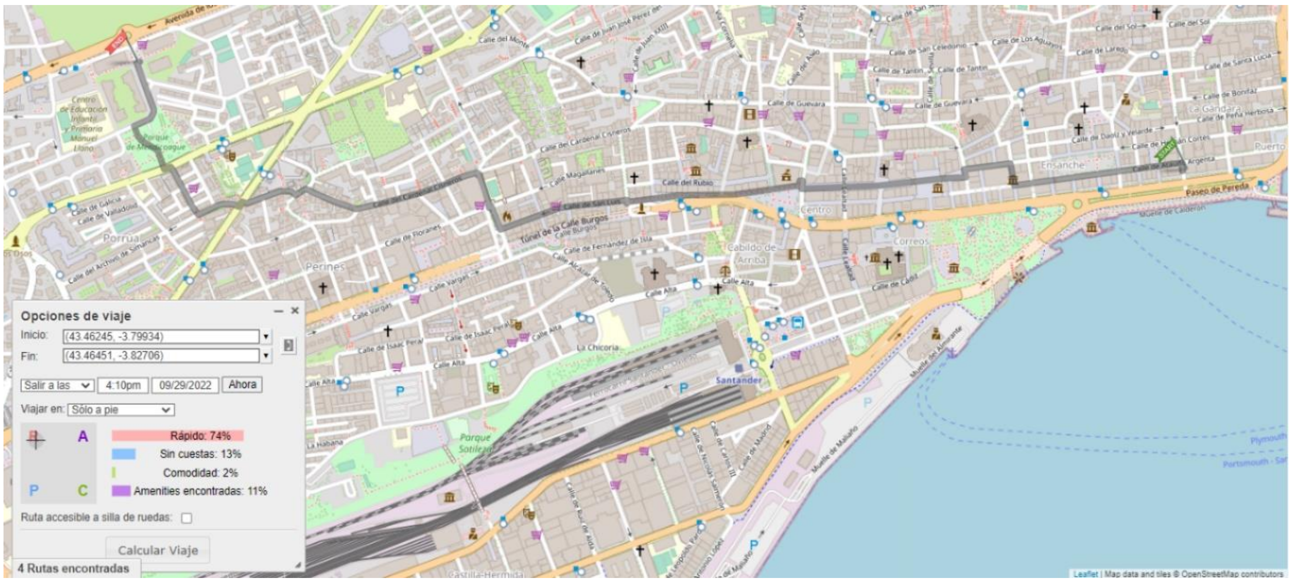
Figure 11: Graphic example of the Square Optimization in the Age-Friendly route planner for walking routes.

Furthermore, in addition to the slope and duration factors, which are the same as the Basic OTP, the AGE-Friendly route planner considers two factors:

- **Amenities found along the route**: this factor considers the amenities described in the previous section 4.1.2.2.2. In this case: benches, toilets and drinking water fountains. This way, as higher this factor, more amenities will be found along the route. In other words, a high value of this factor implies that the route planner will prioritize going through streets that count with these considered amenities.
- **Comfortability factor**: the comfortability factor considers those elements that make the route more comfortable for the user. At the time of writing this deliverable, and because of the lack of additional data, only handrails have been deemed on this comfortability factor. In future stages of the Age-Friendly route planner, additional aspects such as shadows will be contemplated for this factor. This way, and as same as for the amenities factor, as higher the comfortably factor is, the more comfortable the routes will be.

On the other hand, **the second of the innovations of the Age-Friendly route planned is the inclusion of this *Square Optimization* for walking routes. This improvement supposes an improvement not only regarding the Basic OTP but also the vast majority of general-purpose route planners available in the literature.**

Now, we present four different examples of walking routes, each one clearly prioritizing one of the deemed factors. This demonstration is taking place in the city of Santander, and in order to properly show the impact of the four factors, the same origins and destinations are considered for each route. Thus, Figure 12, represents a route prioritizing a quick route, Figure 13, depicts a path devoted to use low-inclination routes, while Figure 14 shows a route prioritizing the finding of amenities along the route, and Figure 15, represents a path which gives a higher importance to the comfortability.


Opciones de viaje

Inicio: (43.46245, -3.79934)

Fin: (43.46451, -3.82706)

Salir a las | 4:10pm | 09/29/2022 | Ahora

Viajar en: Sólo a pie

Rápido: 74%

Sin cuestas: 13%

Comodidad: 2%

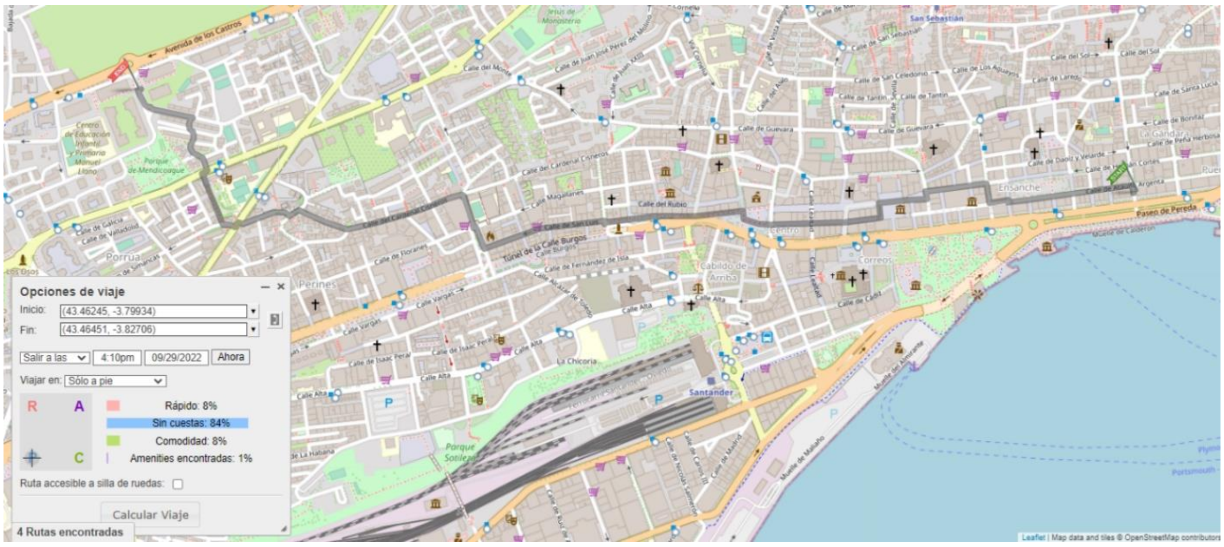
Amenities encontradas: 11%

Ruta accesible a silla de ruedas: ☐

Calcular Viaje

4 Rutas encontradas

Figure 12: A preference-based walking route prioritizing quickness.


Opciones de viaje

Inicio: (43.46245, -3.79934)

Fin: (43.46451, -3.82706)

Salir a las | 4:10pm | 09/29/2022 | Ahora

Viajar en: Sólo a pie

Rápido: 8%

Sin cuestas: 84%

Comodidad: 8%

Amenities encontradas: 1%

Ruta accesible a silla de ruedas: ☐

Calcular Viaje

4 Rutas encontradas

Figure 13: A preference-based walking route prioritizing low-slope streets.

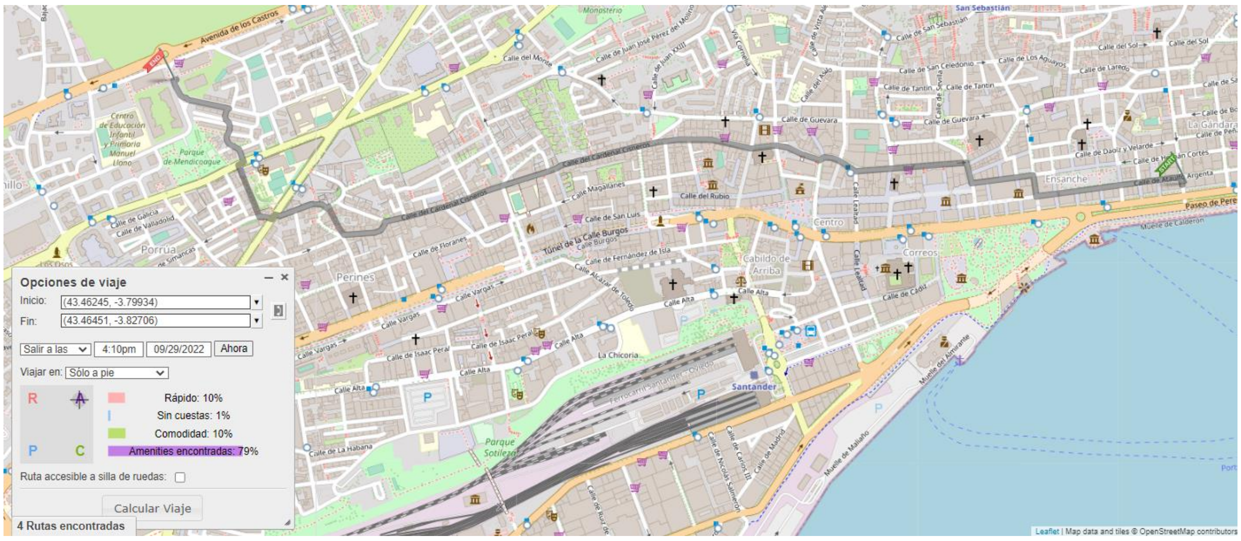

Opciones de viaje
Inicio: (43.46245, -3.79934)
Fin: (43.46451, -3.82706)
Salir a las | 4:10pm | 09/29/2022 | Ahora
Viajar en: Sólo a pie
Rápido: 10%
Sin cuestas: 1%
Comodidad: 10%
Amenities encontradas: 79%
Ruta accesible a silla de ruedas:
Calcular Viaje
4 Rutas encontradas

Figure 14: A preference-based walking route prioritizing the finding of amenities along the route.

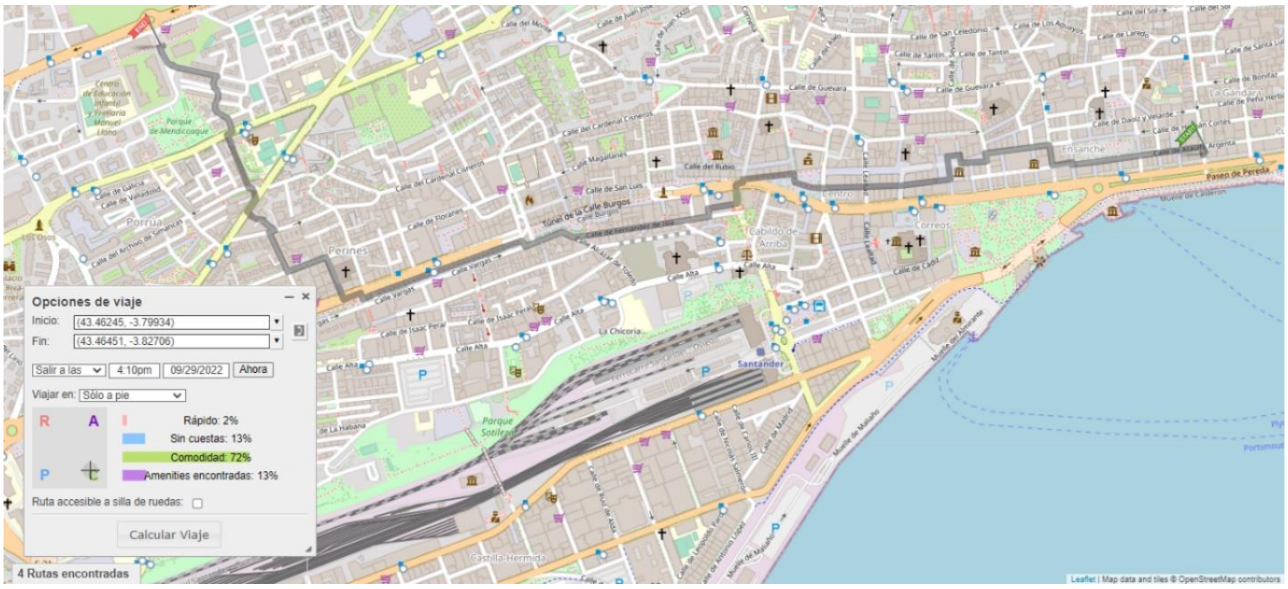
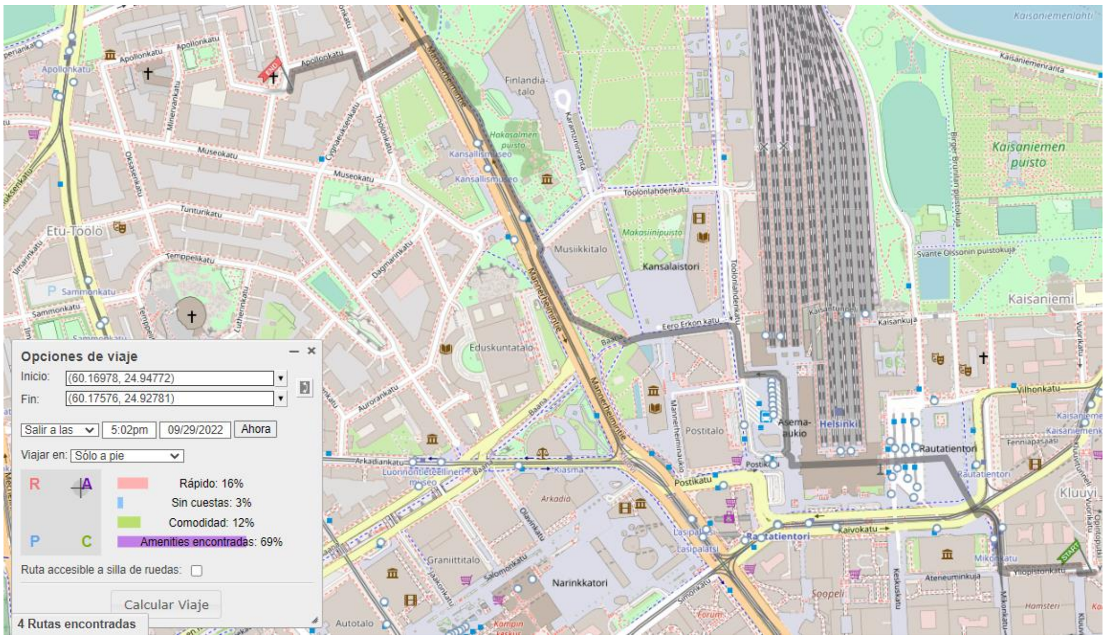

Opciones de viaje
Inicio: (43.46245, -3.79934)
Fin: (43.46451, -3.82706)
Salir a las | 4:10pm | 09/29/2022 | Ahora
Viajar en: Sólo a pie
Rápido: 2%
Sin cuestas: 13%
Comodidad: 72%
Amenities encontradas: 13%
Ruta accesible a silla de ruedas:
Calcular Viaje
4 Rutas encontradas

Figure 15: A preference-based walking route prioritizing the comfortability of the route

4.1.3.1.2 Balanced routes

In addition to the preference-based route described in the previous section**, the Age-Friendly route planner also provides a route which leverages all the preferences to 25%. The main motivation of this route is to provide the user with an equilibrated path for its consideration, trying to offer different alternatives which can improve the route in any of the four factors detailed in 4.1.3.1.1**. In Figure 16, we depict a route placed in the city of Helsinki and prioritizing the founding of amenities along the route, while in Figure 17, we show the balanced route, which slightly differs from the preference-based one.

Going deeper into these routes, the one depicted in Figure 16, with a clear preference on the finding of amenities, has a 1’7 km distance, and it needs 29 minutes to be made. On the other hand, the route shown in Figure 17 is a little shorter, 1’6 km, and it needs only 27 minutes. This difference in the time is mainly based in the fact the equilibrated route finds less amenities (15) in the route in comparison to the preference based one (29), and it provides a shorter route.


Opciones de viaje

Inicio: (60.16978, 24.94772)

Fin: (60.17576, 24.92781)

Salir a las 5:02pm 09/29/2022 Ahora

Viajar en: Sólo a pie

🟥 Rápido: 16%

🟦 Sin cuestas: 3%

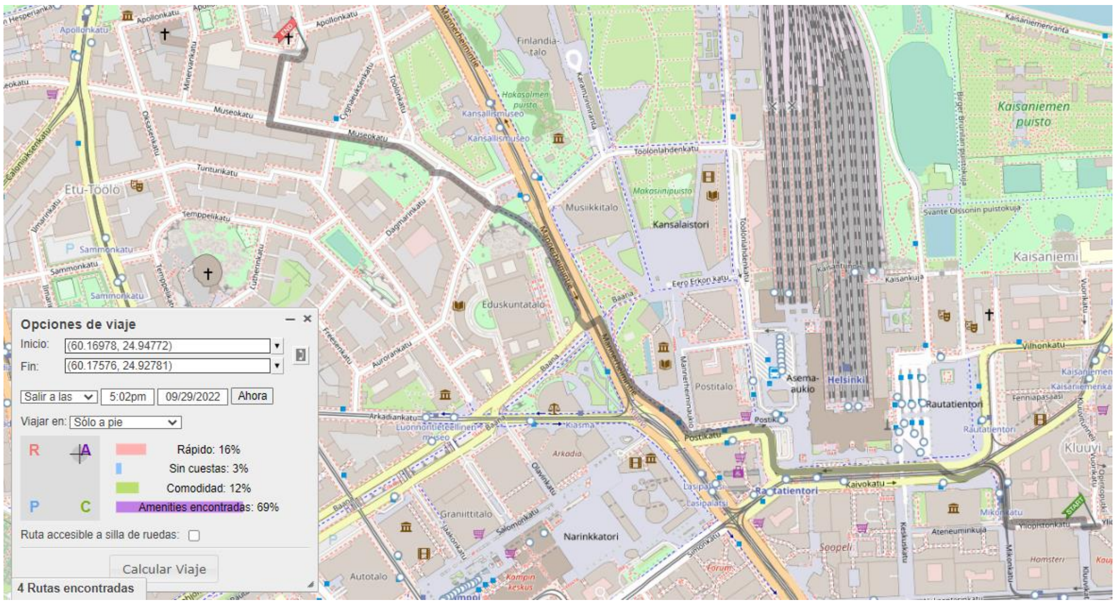
🟩 Comodidad: 12%

🟪 Amenities encontradas: 69%

Ruta accesible a silla de ruedas: ☐

4 Rutas encontradas

Figure 16: A preference-based walking route in Helsinki prioritizing the finding of amenities


Opciones de viaje

Inicio: (60.16978, 24.94772)

Fin: (60.17576, 24.92781)

Salir a las 5:02pm 09/29/2022 Ahora

Viajar en: Sólo a pie

🟥 Rápido: 16%

🟦 Sin cuestas: 3%

🟩 Comodidad: 12%

🟪 Amenities encontradas: 69%

Ruta accesible a silla de ruedas: ☐

Calcular Viaje

4 Rutas encontradas

Figure 17: A balanced walking route in Helsinki

4.1.3.1.3 Artificial Intelligence enhanced route

One of the main objectives of the Age-Friendly route planner is to provide the most optimized routes to the user, based on his/her preferences and needs. **In addition to this, and with the intention of offering the user a pool of alternatives to choose from, the third path calculated by the Age-Friendly route planner is the so-called AI-enhanced route. In a nutshell, the AI-enhanced route is based on an Evolutionary Algorithm, and more specifically a Differential Evolution (DE, [3])**, with the main goal of improving the preference-based route previously explained in section 4.1.3.1.1.

Thus, the main goal of the AI-enhanced route is to offer the citizen an alternative user exploiting the inherent tolerance that the users usually have when they ask for a specific path calculation. For instance, even though a citizen can stablish a fixed preference regarding the importance of slopes or amenities found along the route, it is usually common to be flexible to very slight modifications on the preferences in case the calculated route offers any advantage. **The calculation of this kind of routes is a contribution implemented in the Age-Friendly route planner, and not being available in the basic version of the OTP, nor in the majority of the general-use route planners.**

In this way, each time a user asks for a route made by walking, the Age-Friendly route planner calculates if any alternative route can be delivered by performing a slight modification in the preferences introduced as input. In overall, four are the parameters that the DE modify for the calculation of the AI-enhanced route. We summarize these parameters in Table 5.

Name	Description
squareAmenitiesFactor	For walk square routing, how much amenities matter (range 0-1).
squareComfortabilityFactor	For walk square routing, how much comfortability matters (range 0-1).
squareSlopeFactor	For walk square routing, how much slope matters (range 0-1).
squareTimeFactor	For walk square routing, how much time matters (range 0-1).

Table 5: parameters modified by the DE algorithm for the AI-enhanced route.

As mentioned, the AI-enhanced route is based on a DE algorithm. The DE is one of the most well-known algorithms for solving optimization problems since its inception during the nineties. More specifically, DE was introduced by Rainer Storn and Kenneth Price [3], when Price tried to deal with the problem coined as Chebyshev Polynomial fitting problem. Despite the DE is a classical algorithm, many works today are focused on both presenting improvements to the basic DE [4] or on the using of the algorithm to innovative application fields [5]. Thanks to this intense research, the DE algorithm has already been applied to a wide variety of research fields, such as transportation [6], medicine [7], industry [8] or energy [9].

Furthermore, the DE has been proposed as a solver for facing real-world problems both as sole solver [10], potentially hybridized with other metaheuristics, such as Simulated Annealing [11], Ant Colony Optimization [12], or Particle Swarm Optimization [13]. For further information about the DE, we suggest surveys such as [14] and [15].

From a development and research point of view, it is clear that other metaheuristics could be also adopted for the same purpose, such as the bat algorithm [16], cuckoo search [17], artificial bee colony [18] or the above-mentioned genetic algorithms or ant colony optimization, among many other. **These are some reasons that encouraged us to adopt this classical method: *i)* it is a population-based method, *ii)* it is a flexible method, and it allows the adoption of multiple solving strategies, *iii)* it is easy to implement and powerful in terms of results and robustness, and *iv)* it is fast to run**.

In fact, this last reason has a significant importance for the URBANAGE project. The Age-Friendly route planning system is oriented to solve a real-world problem, in which a user asks for a set of routes. In this context, the running time is crucial, being prohibitive to implement techniques which require a long execution time. The DE developed for the Age-Friendly route planner needs from 7 to 9 seconds to complete the optimization, being a quite acceptable running time considering that each configuration of the population must be tested using the OTP engine. In any case, **the Age-Friendly route planner provides the user the possibility of deactivating the AI-based routes, using the API parameter coined as *enhancedRoute***.

Deepening on the philosophy of the DE, as the vast majority of optimization meta-heuristics that can be found in the literature, it performs an interactive procedure for improving a solution based on a given objective function. Additionally, and because of its population-based nature, the DE optimizes the problem at hand by using a group of complete solutions, which are improved through the constant interaction of the components of the population and through slight modifications on each member. Finally, after each iteration, the DE selects those population members with the better fitness value and, when the termination criterion is met, it provides as solution the individual with the better objective function value.

In the DE developed for the Age-Friendly route planner, each candidate solution is represented by an array of a dimension equal to four, representing the values taken by the parameters described in Table 6. It should be considered also that the sum of all the values should be equal to 1.0, which requires a normalization of the values each time a new candidate is generated. Using a schematic description, $f(x)$ depicts the objective function of the problem at hand, which in this case is the distance needed to complete the route, and which should be minimized. Additionally, x represents a solution, $F \in [0,2]$ is the differential weight, and $CR \in [0,1]$ depicts the crossover probability. Thus, the DE working flow can be described as:

1. *Randomly initialize the population of candidate solutions* $x = [x_1 \ldots x_n]$
2. *For each solution* x *in the population*
   - *Pick at random two solutions* b *and* c *from the population (different from* x*).*
   - *Randomly chose an index* $R\{1, \ldots, n\}$*, being* n *the size of the problem.*
   - *Calculate the position of the newly generated candidate* $x' = [x'_1 \ldots x'_n]$ *as follows:*
     - *Chose a randon number* $r_i \equiv U(0,1)$ *for every* $i \in \{1, \ldots, n\}$*.*
     - *If* $r_i < CR \rightarrow x'_i = best_i + F * (b_i - c_i). In\ any\ other\ case, x'_i = x_i$
   - *If the newly generated candidate* x' *is better than the original* x*, that is, it* $f(x') < f(x)$*, so* $x = x'$
3. *Return to step 2 is the termination criterion has not been reached.*

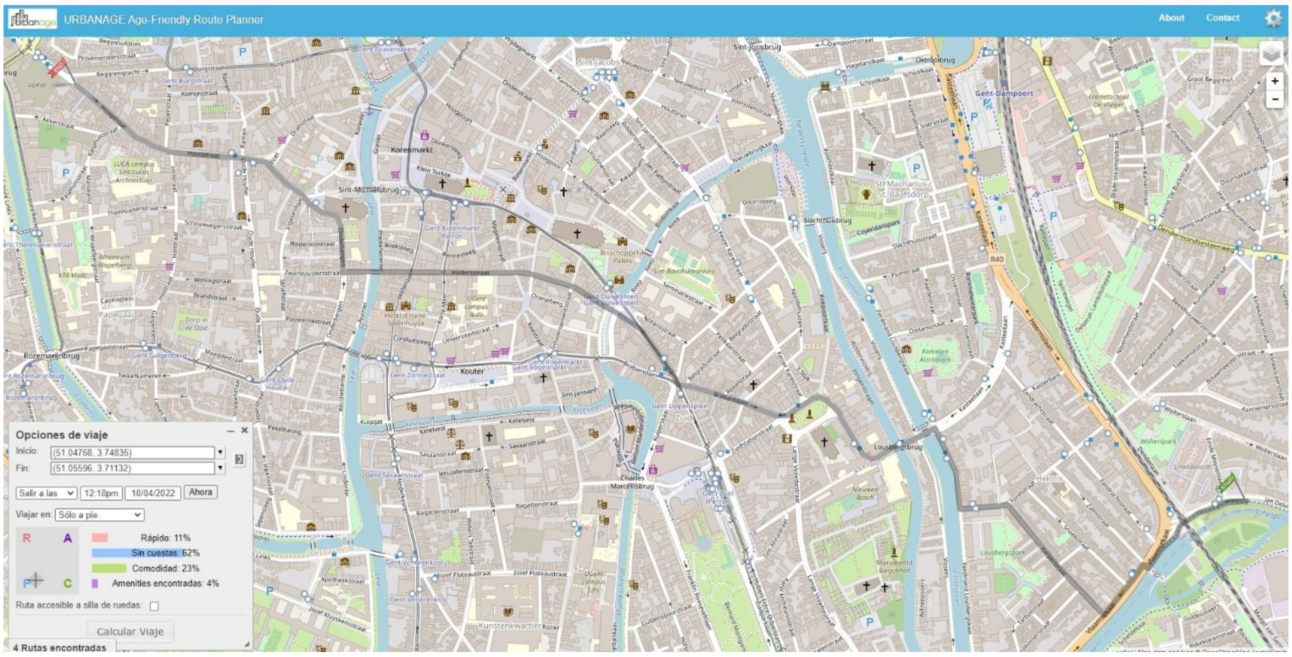
Regarding the parameterization of the DE algorithm developed, it has been set after a thorough empirical testing process, using several benchmarking routes as testing baseline. The proper election of parameters such as the size of the population (NP), CR and F have a crucial influence in the algorithm's performance. Thus, we summarize in Table X the parameterization used. It is also worth mentioning that the variant of DE implemented is the Best-1-Bin. The main motivation for selecting this variant is its efficiency in terms of computational time.

Name of the parameter	Value
DE variant	Best-1-Bin
Crossover constant (CR)	0.8
Select weighting factor (F)	0.4
Population Size (NP)	12
Maximum number of iterations	18
Dimension	4
Maximum number of random choices	10

Table 6: Parameterization of the DE

Additionally, in order to properly include this new functionality into the OTP main structure, two packages have been created: `org.opentripplaner.routing.genetic` and `org.opentripplanner.routing.genetic.de`. The composition of these packages is deeply detailed in Section X.

As a demonstration of the AI-enhanced routes functionality, in Figure 18 we depict a route placed in the city of Ghent and prioritizing the choosing of low-slope streets, while in Figure 19 we show the route found by the DE algorithm, which clearly differs from the preference-based one. Going deeper into these routes, the one depicted in Figure 18, has a distance of 3'4 km of length and 54 minutes of duration, with the following preferences: [duration: 11%, slope 62%, comfortability 29%, amenities found 4%]. On the other hand, the route shown in Figure 19 is shorter, 3'2 km, and it needs 51 minutes to be made. This difference in the time and distance is based on the new preferences found by the DE, which slightly differs from the original ones: [duration: 26%, slope 54%, comfortability 15%, amenities found 5%].



Opciones de viaje

Inicio: (51.04768, 3.74835)

Fin: (51.05596, 3.71132)

Salir a las · 12:18pm · 10/04/2022 · Ahora

Viajar en: Sólo a pie

🟥 Rápido: 11%

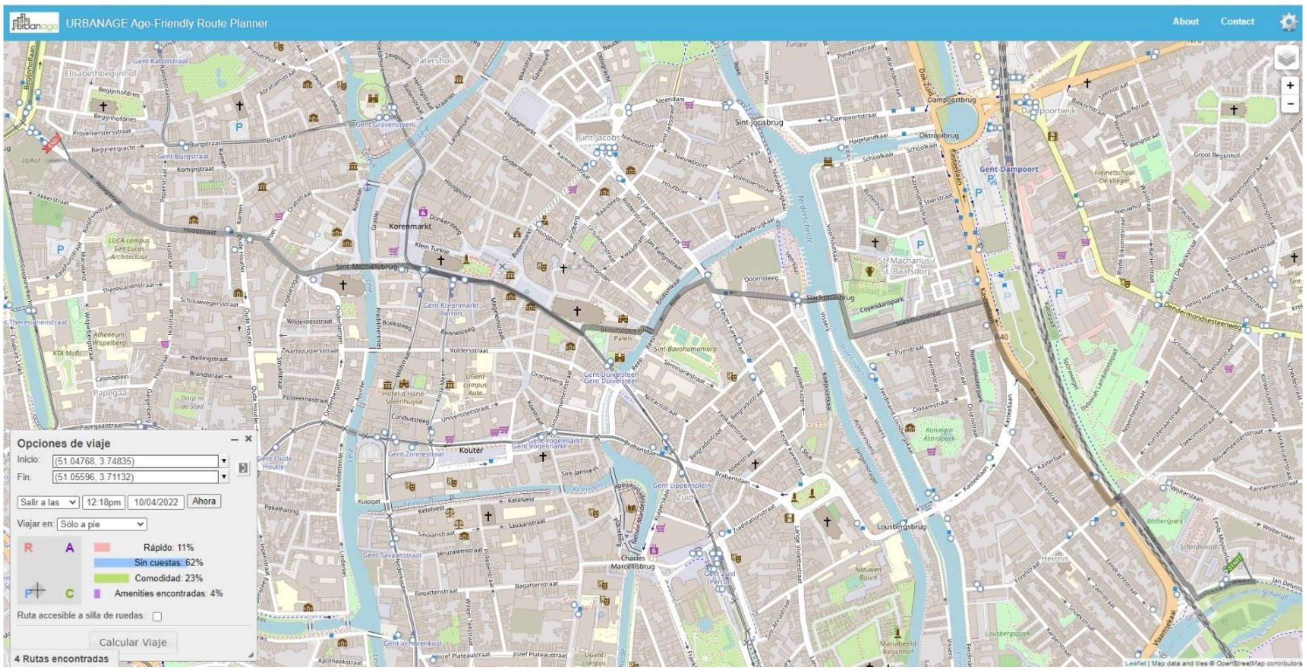
🟦 Sin cuestas: 62%

🟩 Comodidad: 23%

🟪 Amenities encontradas: 4%

Ruta accesible a silla de ruedas: ☐

Figure 18: A Preference-based route in Ghent, prioritizing low-slope street



Opciones de viaje

Inicio: (51.04768, 3.74835)

Fin: (51.05596, 3.71132)

Salir a las 12:18pm 10/04/2022 Ahora

Viajar en: Sólo a pie

Rápido: 11%

Sin cuestas: 62%

Comodidad: 23%

Amenities encontradas: 4%

Ruta accesible a silla de ruedas: ☐

4 Rutas encontradas

Figure 19: An AI-Enhanced route in Ghent, providing an alternative shorter route.

4.1.3.1.4 Route to ensure the distribution of the amenities along the route

As it can be easily deduced, each amenity fulfills a different role in the route. Every single one helps alleviating the weariness of a route in a specific way. For example, benches offer a place to rest, drinkable water sources alleviate the heat and public toilettes are only consciously considered when you really need them.

In any case, each amenity is expected at a different rate. Some ageing citizens might want a bench every 200 meters, but they won't need a drinkable water source because they are carrying a water bottle. Ergo, to properly evaluate the friendliness of a route the route planner should consider not only the number of amenities it has, but also, how spread out these amenities are. And these parameters should also consider user preferences, although having default values could offer useful expert knowledge built in the planner.

Considering this as our main motivation, the Age-Friendly route planner does not only calculate friendly routes in terms of the number of amenities found in the route (as in the preference-based routes, Section 4.1.3.1.3), but it also provides an alternative route that prioritizes the regular distributions of the amenities along the whole route. Also, as will be explained in the following section, the user is even able to select which amenities should be considered for the calculation of this route, and which amenities should be kept outside the calculation (this preference is also possible to be set in preference-based routes). **This alternative route supposes an important innovation for the Age-Friendly route planner, not being present in the basic version of the OTP, nor in the majority of general-use route planners.**

For doing that, **a mathematical formula coined as Amenity Spread Function (ASF) has been modeled, which is applied to every amenity considered by the user**. Thus, the route produced by the Age-Friendly route planner based on this formula has been coined as **ASF-based formula**. The ASF function modelled should follow the fulfillment of these features:

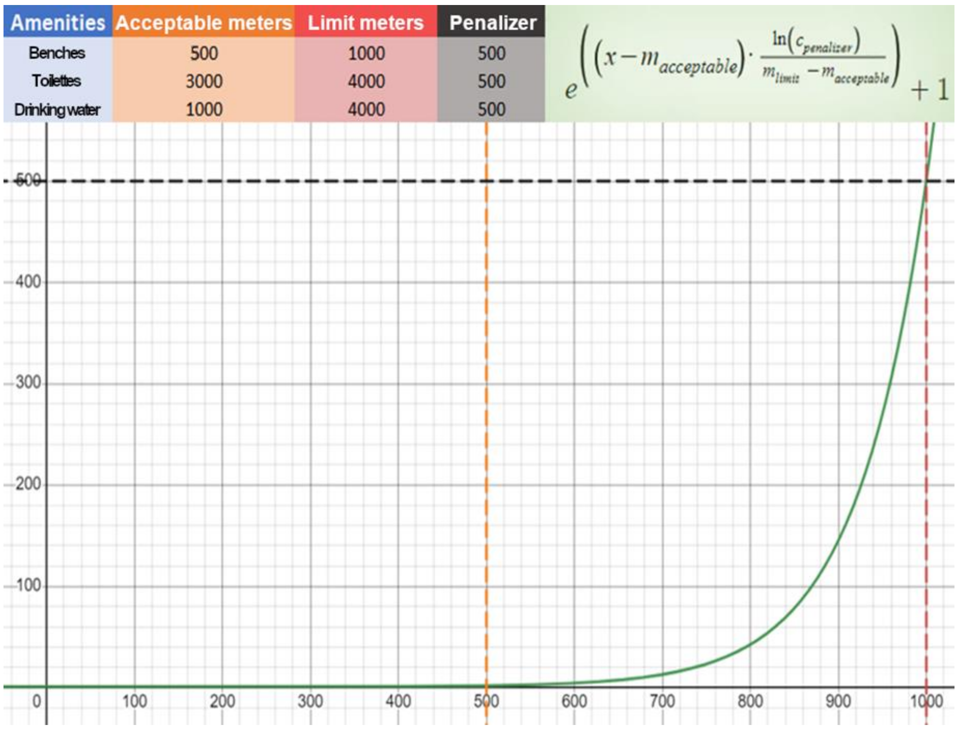
- The ASF should consider how spread out the amenities are, and not simply how many amenities a route has.
- The ASF should be applicable to each amenity type considered, and it should be done separately.
- The function…
  - … outputs value of 1 when it's fulfilling an adequate distribution of amenities.
  - … should have an exponential growth when the route is traversing an inadequate length with no amenities found.
- It should parametrizable per amenity so it can better portray each reality: need for a rest, to drink water or go to the bathroom.
- It can accept both default values and user preferences so it can adapt itself to different use cases: going for a longer walk with a water bottle, needing a toilette more often because of a health issue…

With all this in mind, the ASF has been modeled as follows:

$$e^{\left(\left(x - m_{acceptable}\right) \cdot \frac{\ln\left(c_{penalizer}\right)}{m_{limit} - m_{acceptable}}\right)} + 1$$

Figure 20: ASF function for calculating ASF-based routes.

Where x is the input for the equation and it is the amount of meters traversed since the last amenity found of that kind. Additionally, $m_{acceptable}$ and and m_{limit} are both distances measured in meters; where the first one is the acceptable meter to traverse before each meter costs and extra effort to the user, and the latter is the point where each meter is not traversable anymore. Lastly, $c_{penalizer}$ is the cost of a meter at m_{limit}. The combination of these two values fixes the exponential curve to a very high number and makes the route no longer traversable. We depict in Figure 21 a representation of the ASF, in which $m_{acceptable}$ = 500, m_{limit}= 1000 and $c_{penalizer}$= 500.



Amenities	Acceptable meters	Limit meters	Penalizer
Benches	500	1000	500
Toilettes	3000	4000	500
Drinking water	1000	4000	500

$$e^{\left(\left(x-m_{acceptable}\right)\cdot\frac{\ln\left(c_{penalizer}\right)}{m_{limit}-m_{acceptable}}\right)}+1$$

Figure 21: A visual example of the ASF function.

As a demonstration of the ASF-based routes functionality, in Figure 22 we depict a route placed in the city of Helsinki and prioritizing the quickness, while in Figure 23 we show the route found calculated with the ASF mathematical formulas, using the same user preferences. Going deeper into these routes, the one depicted in Figure 23, is significantly larger than the preference-based one, but it ensures the finding of amenities along the complete route.

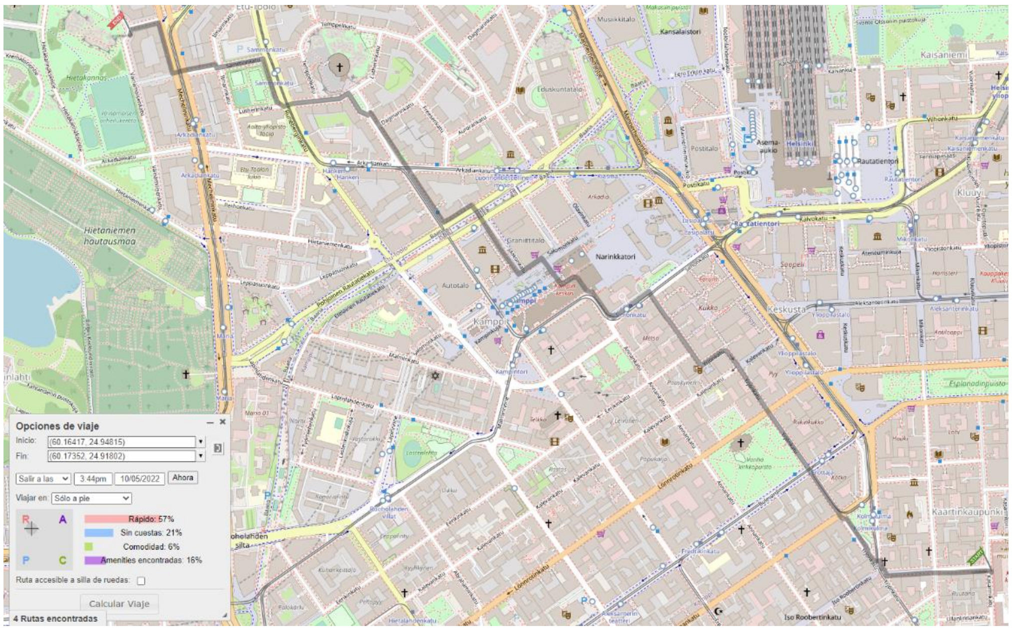

Opciones de viaje
Inicio: (60.16417, 24.94815)
Fin: (60.17352, 24.91802)
Salir a las 3:44pm 10/05/2022 Ahora
Viajar en: Sólo a pie
Rápido: 57%
Sin cuestas: 21%
Comodidad: 6%
Amenities encontradas: 16%
Ruta accesible a silla de ruedas: ☐
Calcular Viaje
4 Rutas encontradas

Figure 22: A Preference-based route in Helsinki, prioritizing quickness

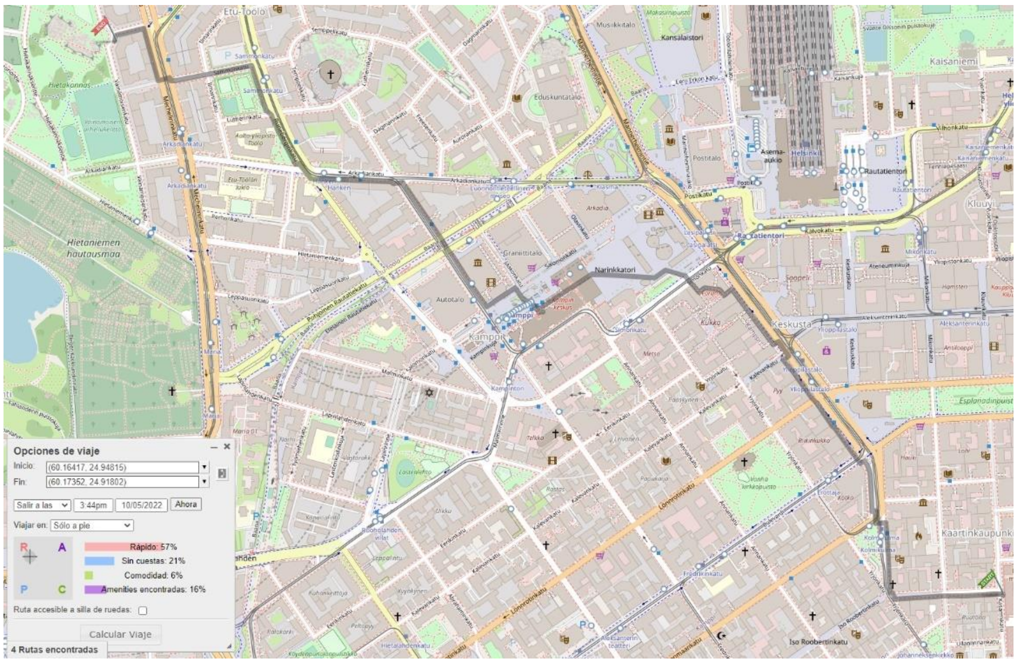

Opciones de viaje
Inicio: (60.16417, 24.94815)
Fin: (60.17352, 24.91802)
Salir a las 3:44pm 10/05/2022 Ahora
Viajar en: Sólo a pie
Rápido: 57%
Sin cuestas: 21%
Comodidad: 6%
Amenities encontradas: 16%
Ruta accesible a silla de ruedas: ☐
Calcular Viaje
4 Rutas encontradas

Figure 23: the ASF-based route in Santander, prioritizing the spreading of amenities along the route.

4.1.3.2 Public transportation routes

The Age-Friendly route planner developed in the context of URBANAGE project also allows the user to calculate public transportation-based routes. For properly building these paths, the route should be fed with the corresponding GTFS files, as described in 4.1.2.2.3.

This functionality does not provide much innovation to the Age-Friendly route planner, since lots of route planners can be found in the literature which consider these paths in a proper way. It should be said, in any case, that the preference-based concepts described in Figure 24: Public Transportation route in the city of Ghent, conducted by tram. Figure 24 is also applied for this kind of routes, just in this route-portions made by walking. In any case, usually, these portions are so short that it does not provide significant advantage for the user.

As a demonstration of the public transportation routes, in Figure 24 we depict a route placed in the city of Ghent, conducted in tram, while in Figure 25 we show the route with the same origin and destination, but performed by bus. In this regard, the Age-Friendly route planner provides the user with more than one route.

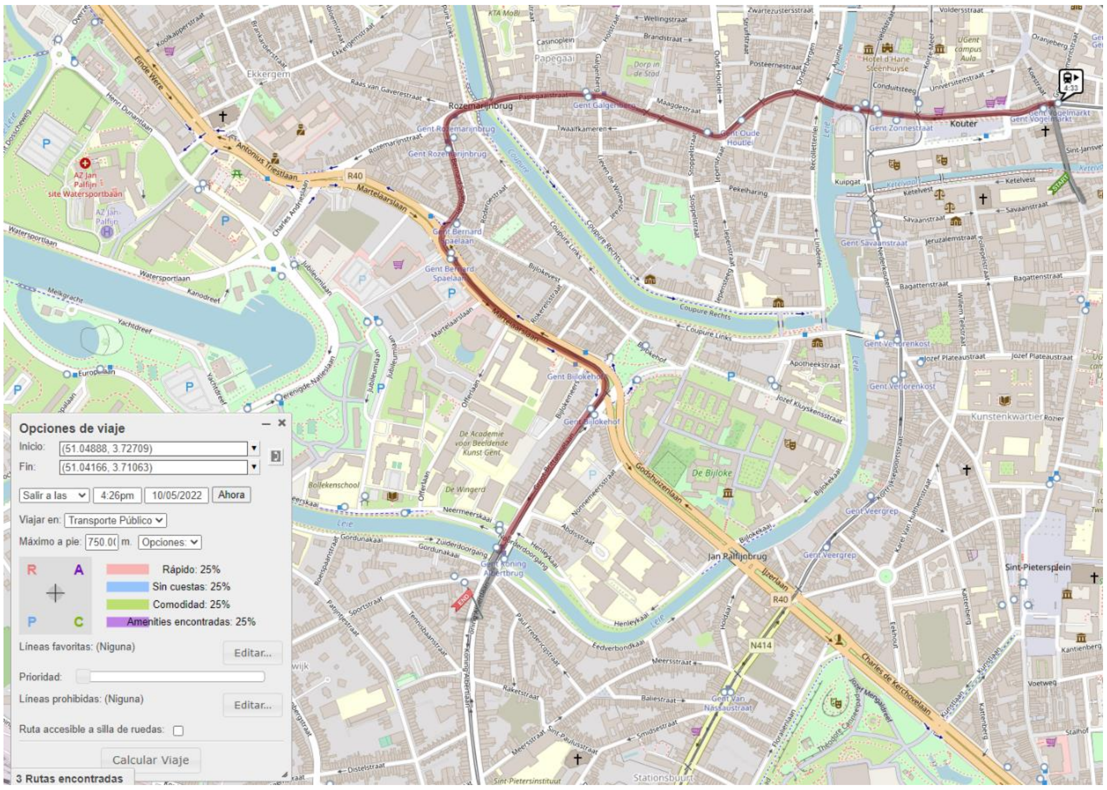
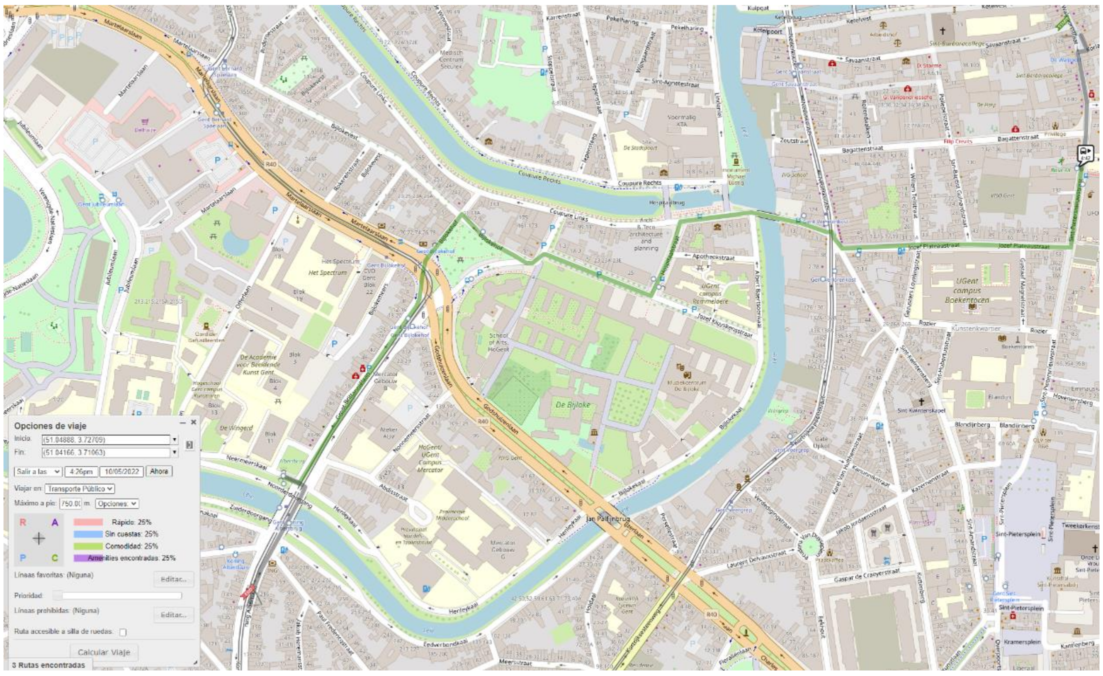

Opciones de viaje
Inicio: (51.04888, 3.72709)
Fin: (51.04166, 3.71063)
Salir a las | 4:26pm | 10/05/2022 | Ahora
Viajar en: Transporte Público
Máximo a pie: 750.0(m. Opciones:
🟥 Rápido: 25%
🟦 Sin cuestas: 25%
🟩 Comodidad: 25%
🟪 Amenities encontradas: 25%
Líneas favoritas: (Niguna)
Prioridad:
Líneas prohibidas: (Niguna)
Ruta accesible a silla de ruedas: ☐
3 Rutas encontradas

Figure 24: Public Transportation route in the city of Ghent, conducted by tram.



Opciones de viaje

Inicio: (51.04888, 3.72709)

Fin: (51.04166, 3.71063)

Salir a las | 4:26pm | 10/05/2022 | Ahora

Viajar en: Transporte Público

Máximo a pie: 750.0(m | Opciones:

🟥	Rápido: 25%
🟦	Sin cuestas: 25%
🟩	Comodidad: 25%
🟪	Amenities encontradas: 25%

Líneas favoritas: (Niguna) Editar...

Prioridad:

Líneas prohibidas: (Niguna) Editar...

Ruta accesible a silla de ruedas: ☐

Calcular Viaje

3 Rutas encontradas

Figure 25: Public Transportation route in the city of Ghent, conducted by bus.

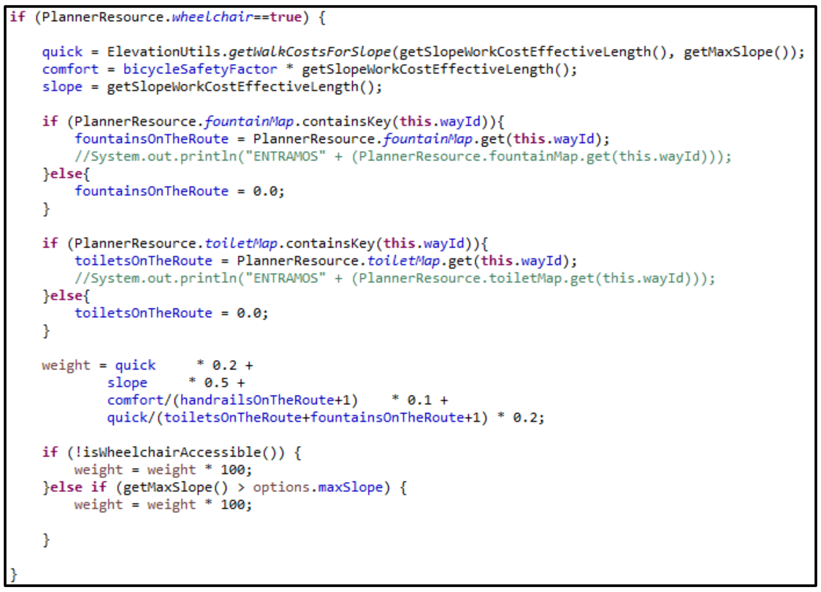
4.1.3.3 Wheelchair routes

An important feature of the Age-Friendly route planner is the possibility for the user to calculate routes conducted using a wheelchair. This kind of routes can be calculated both for pedestrian and public transportation routes,. This is the reason of highlighting this feature in this separated subsection. The calculation of this kind of routes involves the clear understanding and consideration of the elements that build the city maps. Aspects such as streets with high slopes, or stairs must be clearly faced by routes performed by people needing wheelchair.

It should be considered that this kind of routes are not often seen in general purpose route planners. Google Maps, for example, allows the calculation of public transportation routes with wheelchair accessibility, but it does not include wheelchair special needs for pedestrians. Additionally, the basic OTP supposedly allows the building of this kind of route. **In any case, in order for these routes to be calculated, the OSM files must be perfectly built, requiring that each street should have all the accessibility information, something that is virtually impossible considering that OSM is an open collaborative map. In any other case, the system fails to provide the route to the user**.

Considering that the route planner implemented in URBANAGE is specifically oriented for ageing people, we have considered this feature as crucial, adding some important amendments in the base code in order the user be able to plan routes conducted by wheelchair. **For this reason, this feature is an innovation for the Age-Friendly route planner.**

More specifically, it is also interesting to mention that the weight of traversing a street with high slopes and stairs has been modified in the method `doTraverse()` of the class `StreetEdge`, as can be seen inFigure 26. Additionally, in Figure 26we represent an additional except of the code, newly introduced for considering wheelchair routes. To achieve this, an additional Boolean parameter `wheelchair` has been introduced in the class `PlannerResource`, in package `org.opentripplanner.api.resource`.



```
if (PlannerResource.wheelchair==true) {

    quick = ElevationUtils.getWalkCostsForSlope(getSlopeWorkCostEffectiveLength(), getMaxSlope());
    comfort = bicycleSafetyFactor * getSlopeWorkCostEffectiveLength();
    slope = getSlopeWorkCostEffectiveLength();

    if (PlannerResource.fountainMap.containsKey(this.wayId)){
        fountainsOnTheRoute = PlannerResource.fountainMap.get(this.wayId);
        //System.out.println("ENTRAMOS" + (PlannerResource.fountainMap.get(this.wayId)));
    }else{
        fountainsOnTheRoute = 0.0;
    }

    if (PlannerResource.toiletMap.containsKey(this.wayId)){
        toiletsOnTheRoute = PlannerResource.toiletMap.get(this.wayId);
        //System.out.println("ENTRAMOS" + (PlannerResource.toiletMap.get(this.wayId)));
    }else{
        toiletsOnTheRoute = 0.0;
    }

    weight = quick     * 0.2 +
            slope     * 0.5 +
            comfort/(handrailsOnTheRoute+1)    * 0.1 +
            quick/(toiletsOnTheRoute+fountainsOnTheRoute+1) * 0.2;

    if (!isWheelchairAccessible()) {
        weight = weight * 100;
    }else if (getMaxSlope() > options.maxSlope) {
        weight = weight * 100;

    }
}
```

Figure 26: modifications introduced in `doTraverse()` method for calculating the weight of routes conducted by citizens using a wheelchair.

Demonstration routes considering wheelchair user will be shown in the following section.

4.1.3.4 Deepening in the use of public infrastructures (elevators, funiculars, escalators…)

Considering that the Age-Friendly route planner should be specially focused on embracing the needs of ageing citizens, **the use case leaders of URBANAGE recognised the necessity of evaluating public infrastructures while calculating age-friendly routes. For the route planner, elevators, funiculars, and escalators have been considered for the path building.**

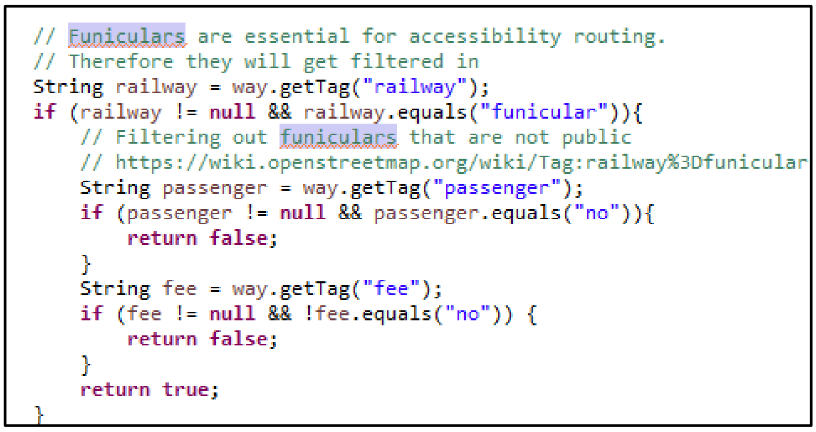
After several testing procedures, use case leaders and tool developers noticed that the basic version of OTP was uncapable of routing through public infrastructure elements in a regular fashion. This feedback was considered because the use of this type of infrastructure was identified as critical in the development of the planner tool; and both the data and the OTP engine was put through a deep analysis. From this analysis two conclusions were extracted:

- First, a particular circumstance was noticed in the Santander scenario, which made routing through elevator a rare occurrence. This circumstance was that the elevators that were being checked out near ‘Calle Santa Teresa de Jesus’ were isolated from the street by conveying stairs, and these were usually avoided for wheelchair users, or simply because they were being evaluated as too steep. The discovery of the tag ‘conveying’ for stairs lead to other ‘conveying’ paths and these segments were noted as segments that should be almost ‘free’ to walking through in terms of cost.
- The second conclusion was that basic OTP does not route over elevators that travel through funicular railways, because this street edges are not considered as routable even if the funicular is free and public. To put this in perspective, Google, the most popular router, solves this issue by transforming funicular railways as walkable streets. The solution of Google is not applicable for URBANAGE, given that our goal is to route ageing users through the most accessible streets. For this reason, funicular or conveying stairs - which are considered normal streets by Google - will be discarded because of their steepness, making these routes virtually unrouteable.

After this analysis, several modifications have been applied in the code of the basic OTP in order to account for the use of public infrastructures. **These modifications suppose a novelty and an important innovation for the Age-Friendly route planner, and it is a crucial step for reaching the goals of the project**.

Among the modifications introduced, we can highlight three of them, consisting of:

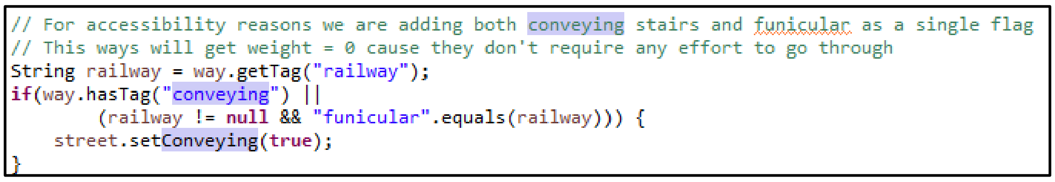
1. Filtering funicular railways as walkable routes when they accept passengers and are free. This is added in the class `OSMFilter`, in the package `org.opentripplanner.graph_builder.module.osm`. An excerpt of the code has been depicted in Figure 27.



```
// Funiculars are essential for accessibility routing.
// Therefore they will get filtered in
String railway = way.getTag("railway");
if (railway != null && railway.equals("funicular")){
    // Filtering out funiculars that are not public
    // https://wiki.openstreetmap.org/wiki/Tag:railway%3Dfunicular
    String passenger = way.getTag("passenger");
    if (passenger != null && passenger.equals("no")){
        return false;
    }
    String fee = way.getTag("fee");
    if (fee != null && !fee.equals("no")) {
        return false;
    }
    return true;
}
```

Figure 27: Filtering funicular railways as walkable routes

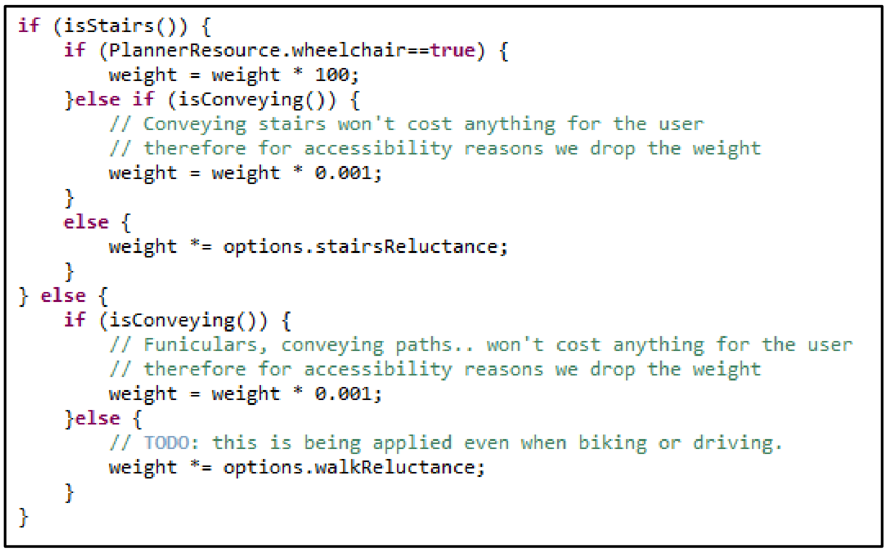
2. Both elevators and conveying paths have been tagged as *conveying* paths because their use will not cost much effort to the user to walk through them. This is added in `OpenStreetMapModule` class, in the package `org.opentripplanner.graph_builder.module.osm`. An excerpt of the code has been depicted in Figure 28.



```
// For accessibility reasons we are adding both conveying stairs and funicular as a single flag
// This ways will get weight = 0 cause they don't require any effort to go through
String railway = way.getTag("railway");
if(way.hasTag("conveying") ||
        (railway != null && "funicular".equals(railway))) {
    street.setConveying(true);
}
```

Figure 28: Tagging elevators and conveying as conveying paths in the code.

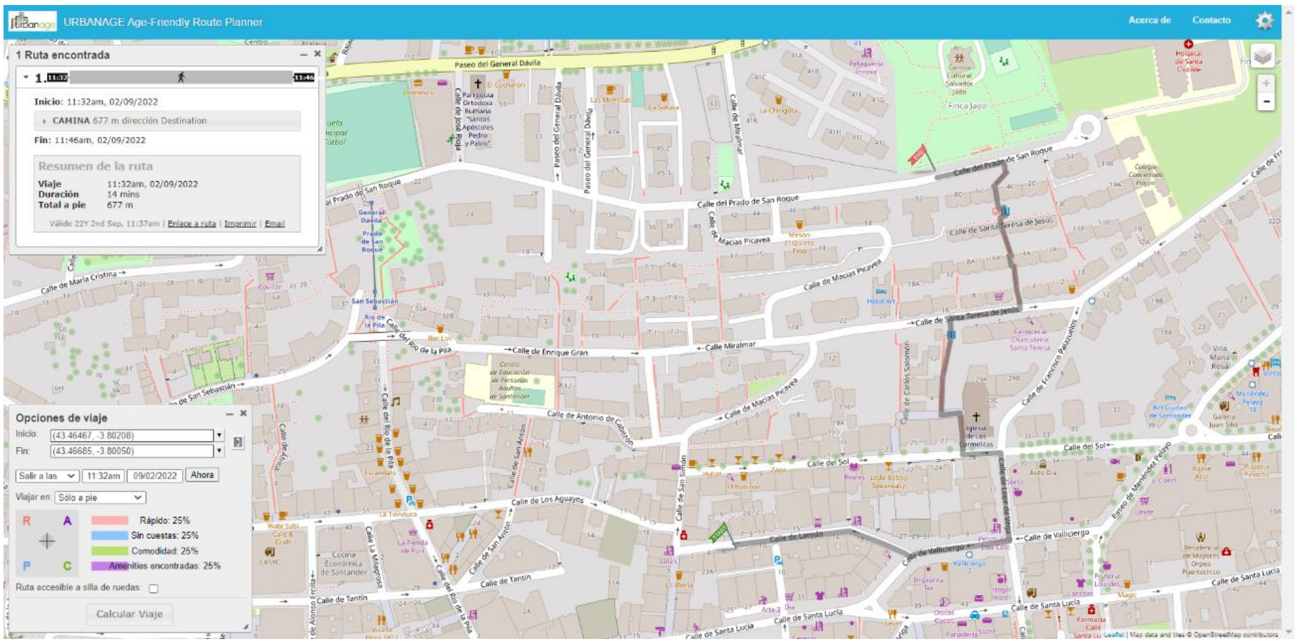
3. The weight function has been modified so paths that contains the *conveying* paths are *cost 0* paths. This is added in the `StreetEdge` class, in the package `org.opentripplanner.routing.edgetype`. A last consideration must be addressed in this regard: conveying stairs are not acceptable routes for wheelchair users. This exception is considered, so these routes are not suggested to wheelchair users but funiculars and conveying paths are. The excerpt of the code that represents this last point is shown in Figure 29.



```
if (isStairs()) {
    if (PlannerResource.wheelchair==true) {
        weight = weight * 100;
    }else if (isConveying()) {
        // Conveying stairs won't cost anything for the user
        // therefore for accessibility reasons we drop the weight
        weight = weight * 0.001;
    }
    else {
        weight *= options.stairsReluctance;
    }
} else {
    if (isConveying()) {
        // Funiculars, conveying paths.. won't cost anything for the user
        // therefore for accessibility reasons we drop the weight
        weight = weight * 0.001;
    }else {
        // TODO: this is being applied even when biking or driving.
        weight *= options.walkReluctance;
    }
}
```

Figure 29: Reducing the cost of going through a conveying, also considering the impossibility of using conveying stairs for citizens using a wheelchair.

As a demonstration of the routes using public infrastructure, and the functionality that permits the calculation of paths considering wheelchairs, we depict several routes placed in the city of Santander. Firstly, below Figure 30 represents a route that uses conveying stairs and elevators. Figure 31 depicts a route that uses a public funicular. Figure 32 shows a route for wheelchair users that goes through an elevator but avoids conveying stairs. Finally, Figure 33 represents a public transportation path for a user not needing a wheelchair, and in Figure 34 we depict the route with the same origin and destination but planned for a citizen using a wheelchair.



1 Ruta encontrada

1. 11:32 11:46

Inicio: 11:32am, 02/09/2022

CAMINA 677 m dirección Destination

Fin: 11:46am, 02/09/2022

Resumen de la ruta

Viaje	11:32am, 02/09/2022
Duración	14 mins
Total a pie	677 m

Válido 22Y 2nd Sep, 11:37am | Enlace a ruta | Imprimir | Email

Opciones de viaje

Inicio: (43.46467, -3.80208)

Fin: (43.46685, -3.80050)

Salir a las 11:32am 09/02/2022 Ahora

Viajar en: Sólo a pie

R A P C

Rápido: 25%

Sin cuestas: 25%

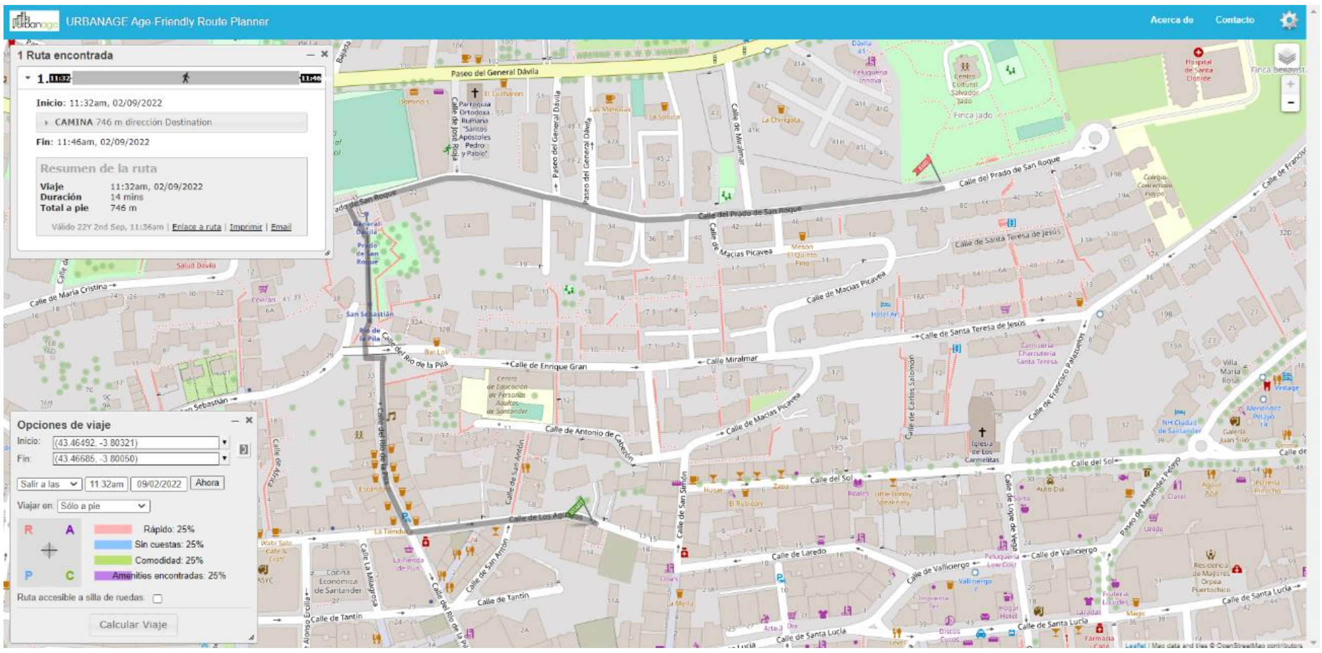
Comodidad: 25%

Amenities encontradas: 25%

Ruta accesible a silla de ruedas: ☐

Calcular Viaje

Figure 30: A route that goes through conveying stairs and elevators



1 Ruta encontrada

1. 11:32 11:46

Inicio: 11:32am, 02/09/2022

CAMINA 746 m dirección Destination

Fin: 11:46am, 02/09/2022

Resumen de la ruta

Viaje	11:32am, 02/09/2022
Duración	14 mins
Total a pie	746 m

Válido 22Y 2nd Sep, 11:36am | Enlace a ruta | Imprimir | Email

Opciones de viaje

Inicio: (43.46492, -3.80321)

Fin: (43.46685, -3.80050)

Salir a las 11:32am 09/02/2022 Ahora

Viajar en: Sólo a pie

R A P C

Rápido: 25%

Sin cuestas: 25%

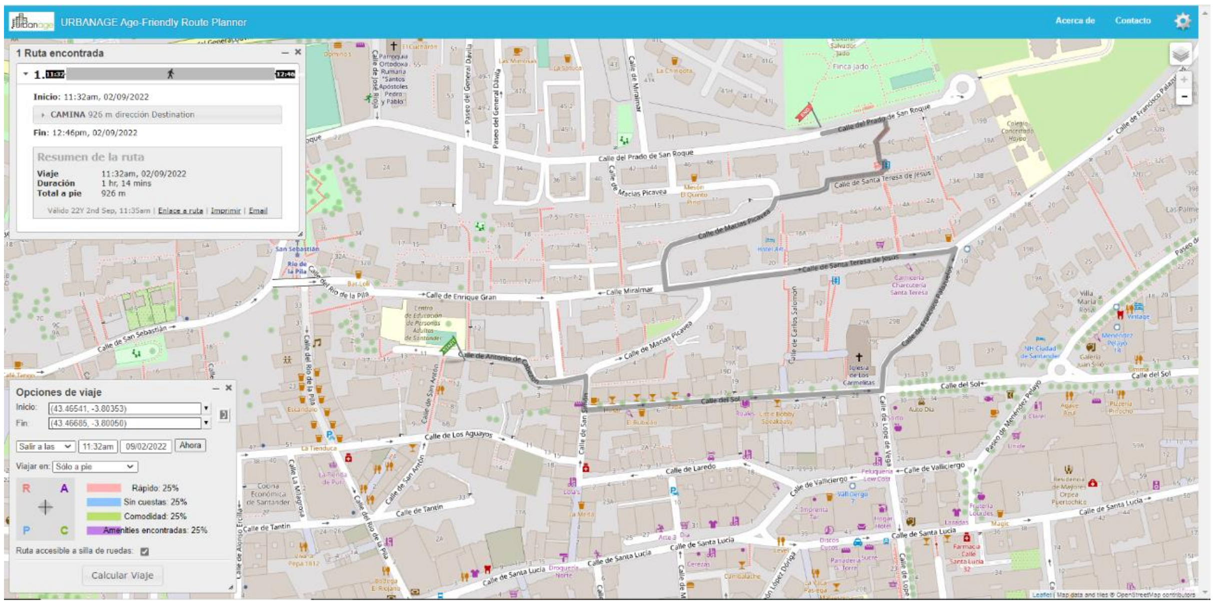
Comodidad: 25%

Amenities encontradas: 25%

Ruta accesible a silla de ruedas: ☐

Calcular Viaje

Figure 31: A route that goes through a public funicular.



1 Ruta encontrada

1. 11:32 12:46

Inicio: 11:32am, 02/09/2022

CAMINA 926 m dirección Destination

Fin: 12:46pm, 02/09/2022

Resumen de la ruta

Viaje	11:32am, 02/09/2022
Duración	1 hr, 14 mins
Total a pie	926 m

Válido 22Y 2nd Sep, 11:35am | Enlace a ruta | Imprimir | Email

Opciones de viaje

Inicio: (43.46541, -3.80353)

Fin: (43.46685, -3.80050)

Salir a las 11:32am 09/02/2022 Ahora

Viajar en: Sólo a pie

Rápido: 25%

Sin cuestas: 25%

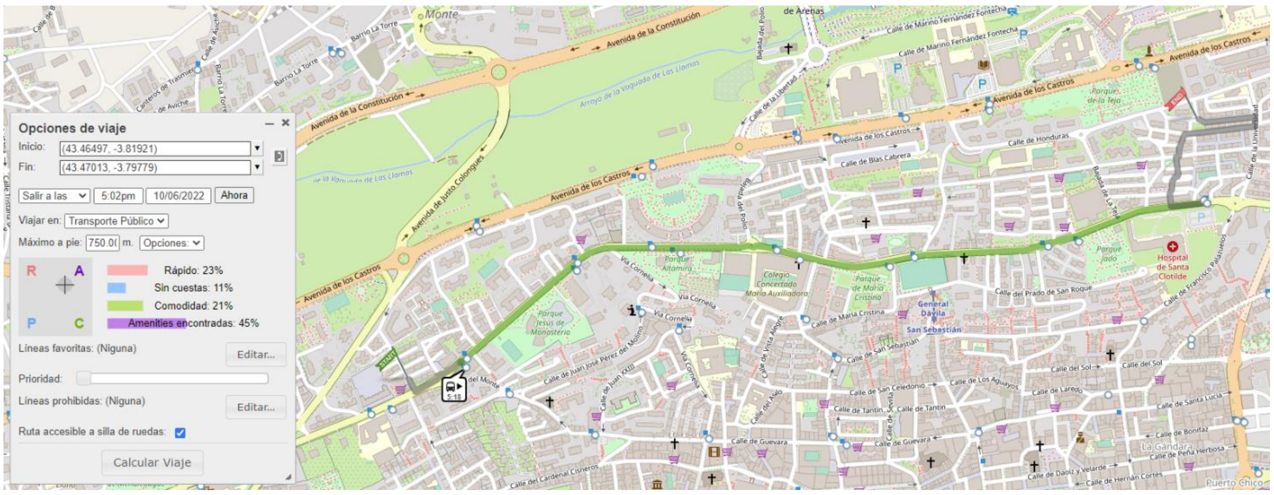
Comodidad: 25%

Amenities encontradas: 25%

Ruta accesible a silla de ruedas: ✓

Calcular Viaje

Figure 32: A route for wheelchair users that goes through an elevator but avoids conveying stairs.


Opciones de viaje

Inicio: (43.46497, -3.81921)

Fin: (43.47013, -3.79779)

Salir a las 5:02pm 10/06/2022 Ahora

Viajar en: Transporte Público

Máximo a pie: 750.0(m. Opciones:

Rápido: 23%

Sin cuestas: 11%

Comodidad: 21%

Amenities encontradas: 45%

Líneas favoritas: (Niguna) Editar...

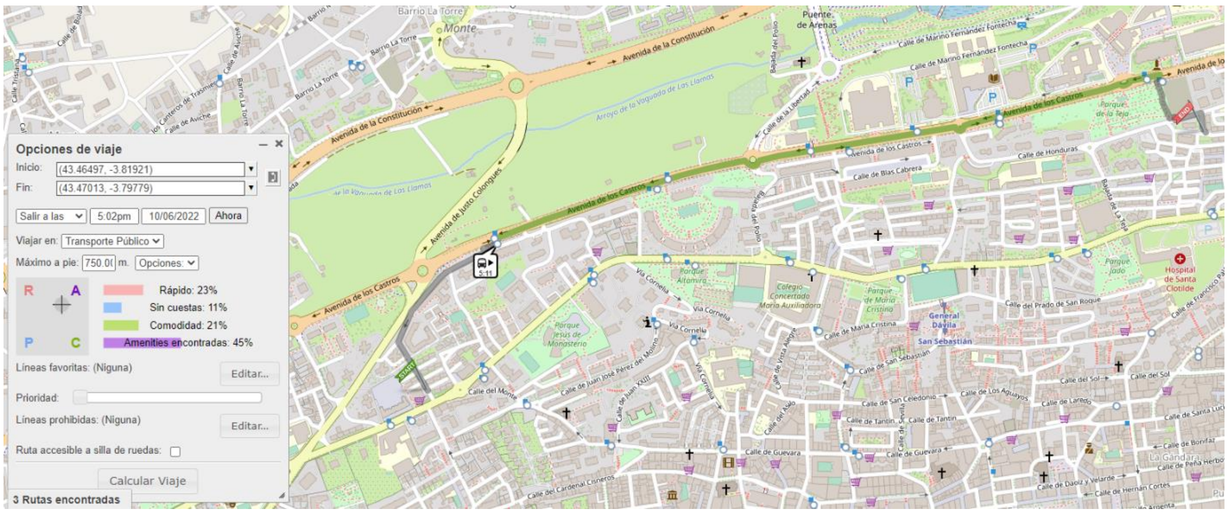
Prioridad:

Líneas prohibidas: (Niguna) Editar...

Ruta accesible a silla de ruedas: ✓

Calcular Viaje

Figure 33: Public transportation route for a user not needing a wheelchair.


Opciones de viaje

Inicio: (43.46497, -3.81921)

Fin: (43.47013, -3.79779)

Salir a las | 5:02pm | 10/06/2022 | Ahora

Viajar en: Transporte Público

Máximo a pie: 750.0(m. | Opciones:

🟥	Rápido: 23%
🟦	Sin cuestas: 11%
🟩	Comodidad: 21%
🟪	Amenities encontradas: 45%

Líneas favoritas: (Niguna)

Prioridad:

Líneas prohibidas: (Niguna)

Ruta accesible a silla de ruedas: ☐

3 Rutas encontradas

Figure 34: Public transportation route for a citizen using a wheelchair.

4.1.3.5 Route limitations and the amenity sweet spot

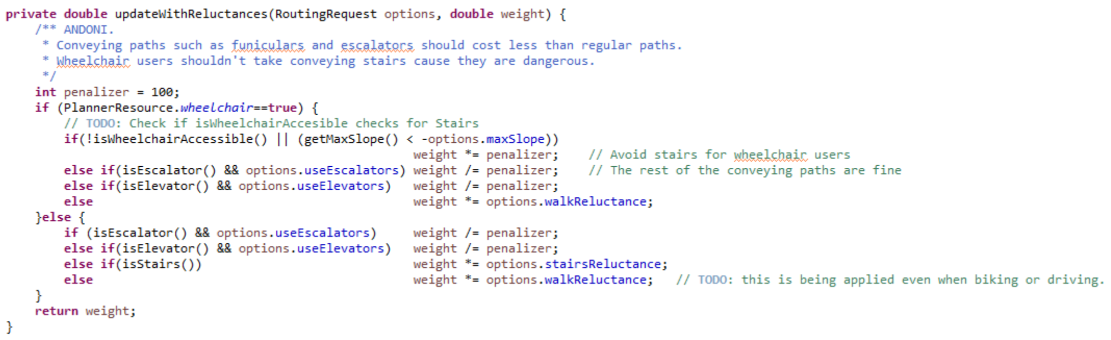
Along the previous section about the Age-Friendly route planner, we have highlighted the ability of the developed tool to account for amenities (benches, toilets, handrails, and water fountains) and public infrastructures (elevators, conveying stairs, and funiculars) in the calculation of friendly routes for ageing citizens. **Finally, in this section we describe an interesting feature implemented for the Age-Friendly route planner, which allows the users to select a point in the route where they want more amenities (from now on, sweet spot).**

Firstly, it is important to point out that a citizen using the route planner could have the desire of not using both elevators and escalators on the planned route. These mechanical conveyors might provide an easier to traverse route, but it is also a fact that users could consider them as frightening, not secure enough or simply don't want to hop into them. Thus, **in the Age-Friendly route planner, the user has the possibility to select which amenities and infrastructures should be prioritized or used.**

These both features have been developed to better define user mobility limitations, which is transversely essential for the Age-Friendly route planner. Thanks to these features, citizens can interact with the two key features of an accessible route: amenity presence on the route, and routing trough conveying paths. Both concerns were tackled at roughly the same moment, which was the end of the planner's development, and therefore are described together. **It should be highlighted that the development of these two features supposes an innovation for the Age-Friendly route planner.**

To accomplish both features two main changes where needed: *a)* new parameters for the API to characterize the requests and a modification of the cost function so the preferences are being considered, and b) the consideration of these preferences in the `doTraverse()` function of the `StreetEdge` Java class of package `org.opentripplanner.routing.edgetype`.

- **New parameters for the API**: first of all, in order to evaluate the using of elevators and escalators along the route two parameters were added to the API: `useElevators` an `useEscalators`. These two parameters are correctly collected and recorded in the `PlannerResource` class of OTP. Then, whenever a `StreetEdge` is traversed `updateWithReluctances()` function is called to penalize or prioritize the route taking into account if `useElevators` or `useEscalators` are set to false, and also previously developed features, such as if the user is using a wheelchair, if the slope is too high, if there are stairs present… We depict the code of this method inFigure 35. An additional step is required to be able to penalize these routes whenever `useElevators` or `useEscalators` are set to false, which is correctly flagging the `StreetEdge` as elevator or escalator edges. This step is done in `OpenStreetMapModule` class.



```java
private double updateWithReluctances(RoutingRequest options, double weight) {
    /** ANDONI.
     * Conveying paths such as funiculars and escalators should cost less than regular paths.
     * Wheelchair users shouldn't take conveying stairs cause they are dangerous.
     */
    int penalizer = 100;
    if (PlannerResource.wheelchair==true) {
        // TODO: Check if isWheelchairAccesible checks for Stairs
        if(!isWheelchairAccessible() || (getMaxSlope() < -options.maxSlope))
                                                    weight *= penalizer;    // Avoid stairs for wheelchair users
        else if(isEscalator() && options.useEscalators) weight /= penalizer;    // The rest of the conveying paths are fine
        else if(isElevator() && options.useElevators)   weight /= penalizer;
        else                                        weight *= options.walkReluctance;
    }else {
        if (isEscalator() && options.useEscalators)     weight /= penalizer;
        else if(isElevator() && options.useElevators)   weight /= penalizer;
        else if(isStairs())                         weight *= options.stairsReluctance;
        else                                        weight *= options.walkReluctance;   // TODO: this is being applied even when biking or driving.
    }
    return weight;
}
```

Figure 35: updateWithReluctances() code for considering the presence of public infrastructure along the route.

- **The amenity sweet spot:** this second feature required a more mathematical approach because characterizing a user preference that loosely describes at what point of the route they want more amenities to be found requires two information factors. The first factor is how to describe when the user wants that sweet spot for amenities. This is done with three string values: `First half`, `Half` and `Second half.` In order to contemplate the amenities sweet spots, these values are sent to the routes thank to a newly added API parameters: `distrBenches` for benches, `distrWater` for water fountains and toilets `distrToilets` for toilets. These are later on translated to percentages that represent the point of the route where they should happen. For example, `First half` would be the 25% of the route, and `Second half` the 75%. More options could be easily added. Elevators and escalators do not have a distribution API parameter because they weren't considered as an amenity that users would want to find at a specific point of the route. **Furthermore, it should be also spotlighted that these parameters are optional, meaning that if the user does not introduce the sweet spot, the amenities will have the same priority along the complete route. Furthermore, if the citizen set to false the newly added parameters `useBenches`, `useToilets` or `useWater`, the corresponding amenity will not have any influence in the calculation of the route.** The second information factor was knowing how long the route was going to be. To accomplish this, a preliminary route is calculated so an estimated route length can be recorded. After it is calculated, there is enough information to get the point where the user would want more amenities to be found and a mathematical solution can be established so OTP prioritizes those specific amenities. The used mathematical formula and how it behaves is shown in Figure 36 and Figure 37, respectively.

$$1 - 0.5 \cdot \exp\left(-\frac{\left(x - \left(p_{locpercent} \cdot m_{total}\right)\right)^2}{\left(2 \cdot \left(m_{acceptable}\right)^2\right)} \right)$$

Figure 36: mathematical formula to consider sweet spots in the route.

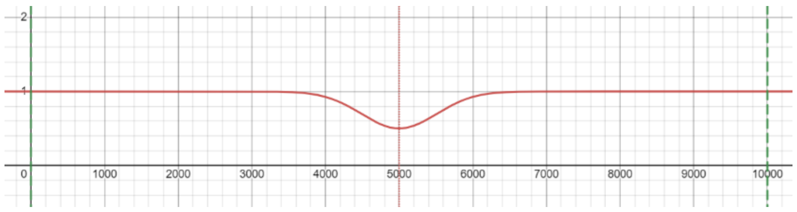
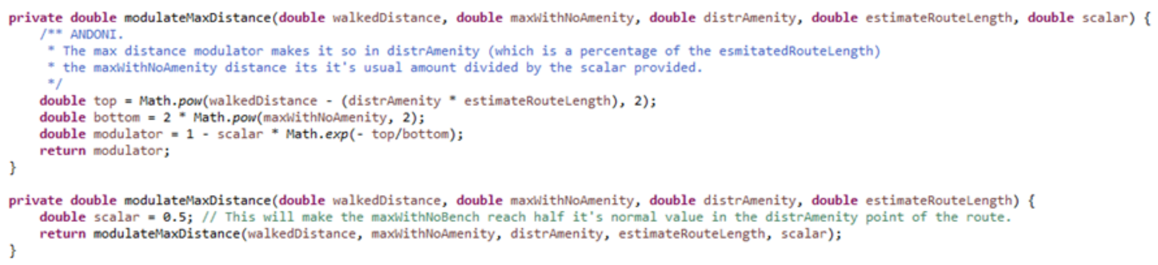


Figure 37: how the amenity sweet spot formula behaves in the user introduces `Half` as input.

The mathematical solution for this feature is designed in a way that the amenity spread function can be upgraded. This function is based on the Gaussian distribution but it is swift upside down so it can better suit the feature. **The approach is to create a variable that will modulate acceptable meters so there will be a segment of the route where acceptable meters is going to be way lower than usual.** This will make the routes to look for more amenities at that segment. By default, it will reduce the acceptable meters to its half and like such it will find double the number of amenities, but this could be adjusted if the scalar in the function above is set to another value. If the function is seen in a graph, the following is true:

- Whenever this amenity sweet spot is not at its location or it is deactivated, the modulator will return 1 and acceptable meters will be leave intact.
- When it is activated and the route is reaching the effect zone, the modulator will return the result an inversed gaussian formula with is, and acceptable meters will get half the acceptable meters by default.

Finally, we represent in Figure 38 an excerpt of the code added to `StreetEdge` class for contemplating this sweet spot formula.



```java
private double modulateMaxDistance(double walkedDistance, double maxWithNoAmenity, double distrAmenity, double estimateRouteLength, double scalar) {
    /** ANDONI.
     * The max distance modulator makes it so in distrAmenity (which is a percentage of the esmitatedRouteLength)
     * the maxWithNoAmenity distance its it's usual amount divided by the scalar provided.
     */
    double top = Math.pow(walkedDistance - (distrAmenity * estimateRouteLength), 2);
    double bottom = 2 * Math.pow(maxWithNoAmenity, 2);
    double modulator = 1 - scalar * Math.exp(- top/bottom);
    return modulator;
}

private double modulateMaxDistance(double walkedDistance, double maxWithNoAmenity, double distrAmenity, double estimateRouteLength) {
    double scalar = 0.5; // This will make the maxWithNoBench reach half it's normal value in the distrAmenity point of the route.
    return modulateMaxDistance(walkedDistance, maxWithNoAmenity, distrAmenity, estimateRouteLength, scalar);
}
```

Figure 38: an excerpt of the code added to `StreetEdge` class for contemplating this sweet spot formula.

4.1.4 Delivery and Usage

4.1.4.1 Package information

Because the Age-Friendly route planner is based on the OTP framework, the project package has the same structure as the basic OTP, with some components added. Additionally, a significant number of classes have been modified in order to properly adapt the planner to the URBANAGE needs. A detailed description can be found in the section 7.1 of Appendix A.

Each of these packages has its main objective and its context within the whole project. Furthermore, these packages are also comprised by several JAVA classes. Identifying the crucial packages, we have:

- `org.opentriplanner.routing.edgetype`: this package contains all the edge types that can built the complete graph that represents a city.
- `org.opentripplaner.api.resource`: This package contains crucial classes for properly dealing with the API call that make the route planner work.
- `org.opentripplaner.api.common`: This package contains few classes, which are in charge of taking the API call introduced by the user (such as the ones described in Section 7.1.2) and properly reading it in order to extract and traduce all the information into JAVA variables.
- `org.opentripplaner.routing.core`: This package contains several classes used to control the routing process.
- `org.opentripplaner.api.model`: This package contains several classes which are crucial for transforming the solution provided by the AI into the JSON that is then returned to the user.
- `org.opentripplaner.routing.genetic`: This is a newly generated package, which has been added to the OTP structure in order to contemplate the calculation of the AI-Enhanced routes described in Artificial Intelligence enhanced route 4.1.3.1.3.
- `org.opentripplaner.routing.genetic.de`: This newly generated package is a concretization of a differential evolution algorithm, employed for calculating the routes described in 4.1.3.1.4.

4.1.4.2 Installation instructions

The Age-Friendly route planner is provided in a compressed folder, which should be imported in any JAVA development framework such as Eclipse or NetBeans. Also, the project can be imported using Maven functionality. In a nutshell, Apache Maven is a software project management and comprehension tool. Inspired by a concept coined as Project Object Model (POM), Maven can manage a build, reporting and documentation in a project from a sole central piece of information. For properly importing the project using this mechanism, the `pom.xml` provided should be used as entry point, which includes, among others, all the dependencies needed for properly running the tool.

4.1.4.3 Download

The Age-Friendly Route Planner is available through the GitLab created for the URBANAGE Project. Furthermore, also the Amenity Projection Tool and OSM data enrichment tool are available in this JAVA project.

At the time of writing this manuscript, this link for downloading all the files needed for correctly running the Age-Friendly Route Planner is private and only accessible for all the partners of the project. For this reason, the source code will be available in the near future on a public repository, and it is going to be accessible through the project's website https://www.urbanage.eu/ . Until this code is available, the source code is offered under request by demand through the "Contact" section available on the website of the project (https://www.urbanage.eu/).

4.2 Simulation tool for long-term urban planning

4.2.1 Functional Description of the component

Urban planning actions often require extended periods of time with a specialized team focused on developing a strategy that is optimized to fulfil a single criterion. If multiple planning criteria are to be considered, the time and complexity of the planification strategy are extended. To do so, the urban development stakeholders, usually composed of a combination of civil servants and urbanism experts, need to consider a wide variety of scenarios and variables that are involved in the evolution of the urban environment. The simulation tool for long-term urban planning is aimed at integrating different aspects of the urban development to create possible scenarios that meet certain criteria and achieve certain objectives. The idea consists of assisting with IA tools in the exploration of scenarios that could be interesting. The Simulation tool is designed to be a visual aid for decision making by computing and visualizing the impact of different possible actuations on a predefined set of key indicators defined by experts. This is fulfilled by providing a dashboard that can be used as a playground in which each criteria value can be tinkered with individually. The Simulation tool is expected to provide two kinds of functionality: on one hand a what-if simulator that shows how changing, adding or defining urban equipment and infrastructure affects the different indicators. On the other hand, and once a set of budgetary and performance objectives are found, the simulation tool should be able to provide, through an AI-based optimization process, the most adequate actuations to perform.

The development of these tools relies entirely upon the definition of an Age Friendliness Index, which aggregates a set of sub-indices that reflect the impact of the urban equipment and services on the age friendliness of the city. Computing some of these indices requires in some cases the estimation of hundreds or thousands of routes, to determine the walking distance from some points to others. A key feature of the development of the simulation tool is its scalability, as this complexity in the computation of indices can affect severely the optimization time, and thus its usability. Hence, the development of the simulation tool has considered a scalable approach. This concept is noticeable in the design of the computations of the different age friendliness subindices (or Key Performance Indicators, KPIs) and the option to select roads in the dashboard.

First of all, the computation of several of those KPIs, is based on Manhattan distance to services, as this kind of distance is more accurate than straight line distance in terms of walking to a service. The computation of these distances from each building main entries to different locations and services is developed via isochrone lines. The isochrones are polygons from a point in the city that represent the area an individual can reach within a set time and given a certain traveling speed. Isochrones are computed using the URBANAGE Age-friendly route planner, so the areas are aware of slopes and other restrictions, such as escalators, stairs, or accessibility hindrances. All these polygons can be computed per address and don't need to be recomputed if the road network is not modified, so the routes do not need to be computed from each building entry to any destination instead, and once isochrones are computed just once, a service is checked whether it falls within the isochrone polygon, a much quicker procedure... This vastly improves the computation but also provides interesting polygon data that can be explored later on to provide new visualization, optimization and data.

In the current version of the Simulation tool, another functionality has been included, aimed at allowing its users the manual selection of locations to place a new infrastructure or equipment. Particularly, the functionality is designed to deploy new ramps or escalators that facilitate the mobility in sloppy streets. This kind of actuations can have a great impact in the walking distance mobility and thus increase considerably increase the area of isochrones, and access to different services, thus increasing the general Age Friendliness Index.

The development of this functionality has entailed a reverse geocoding API call to Nominatim which also uses OSM data. After the inclusion of new ramps or escalators, isochrones must be recomputed, and new scenarios can be explored in which not only adding services are considered but also modifying the road network. These newly selected conveying roads will be added to the Age-friendly route planner API call so they are taken into account and the new set of isochrones will be computed.

Summarizing, this simulation tool has been designed to improve performance, data-exploitation and, to be combined and further improve the Age-friendly route planner's application. Also, the future functionalities of the simulation tool based on optimization would highly benefit from the up to date developed features.

4.2.2 Technical Description of the component

4.2.2.1 Justification of the simulation framework used

This development includes two APIs which are the backbone of the results. These are the Age-friendly route planner and the reverse geocoding of Nominatim.

The age-friendly route planner is obviously chosen because the simulation is built upon the previously developed tools and its aim is to bring the accessibility routing knowledge even further, this time as a tool to explore scenarios. The route planner is essential because it has all the accessibility logic design to aid the ageing citizens and provides an isochrone calculation call, which is a powerful data structure to characterize walkability in the city.

Nominatim is a tool to search OSM data by name and address (geocoding) and to generate synthetic addresses of OSM points (reverse geocoding). Nominatim is part of the Open Street Maps foundation and its design as a search engine for OSM data. Therefore, it's an amazing tool to explore OSM maps. In the simulation tool's case, its purpose is to provide a reverse geocoding feature so streets can be identified just by clicking.

4.2.2.2 Architecture of the component (describing also the data used)

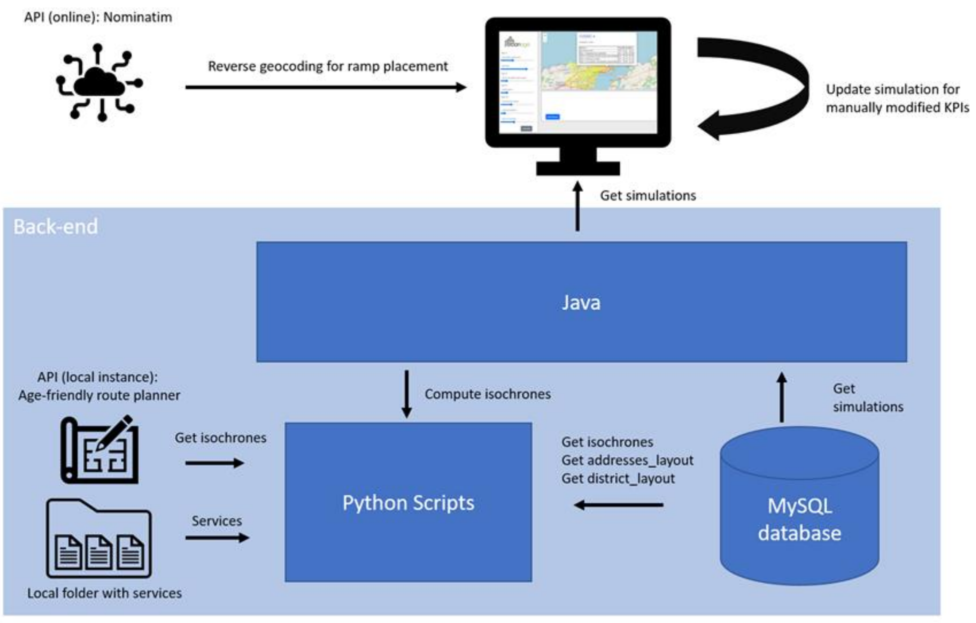


Figure 39: Architecture of the Age friendliness index simulation tool.

As can be seen in Figure 39 above, the simulation tool has its front-end almost completely separated from the rest of the architecture. The only exceptions are the calls to the Nominatim API, so each reverse geocoding request is launched locally. This way the calls are not always made from the back-end machine with would eventually be banned for making too many requests. Additionally, the front-end can compute the indexes on the client side so after it asks for a specific simulation no further calls are required. This will reduce the network load and speed up the simulations when tinkering with indicator values.

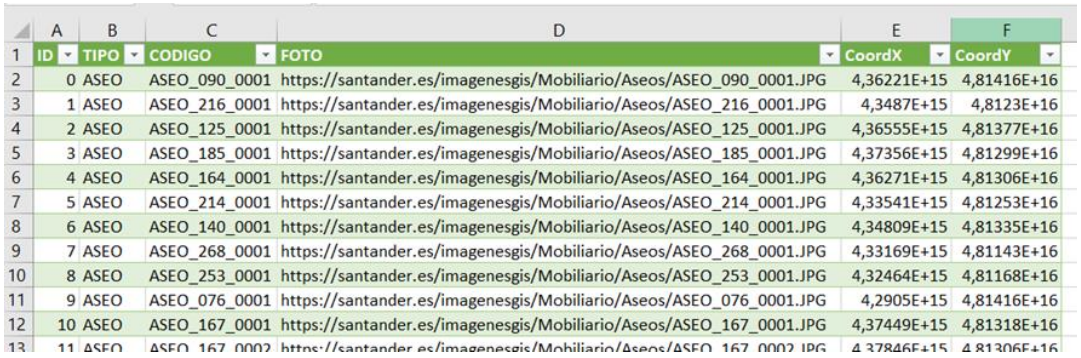
The back-end entry point is managed with Java. This entry point offers the robustness of making multiple calls to a Python API and the request from the database. The Python API uses the already developed Age-friendly route planner to compute isochrones. These scripts also require some additional data. Some are located inside the database and others are stored locally.

The **district layout** is a **geojson** with a FeatureCollection that contains a polygon for each district and properties of such district: Population, Area, CODDC (id), date of the properties.

The **address layout json** follows this structure:
{district_id_1: {address_id_1: {"lat": float, "lon": float}, address_id_2: {coords}, ...} district_id_2: {addresses}, …}
The isochrone json follows this structure:
{district_1: [isochrone_1, isochrone_2, isochrone_3],
district_2: [isochrones…], …}

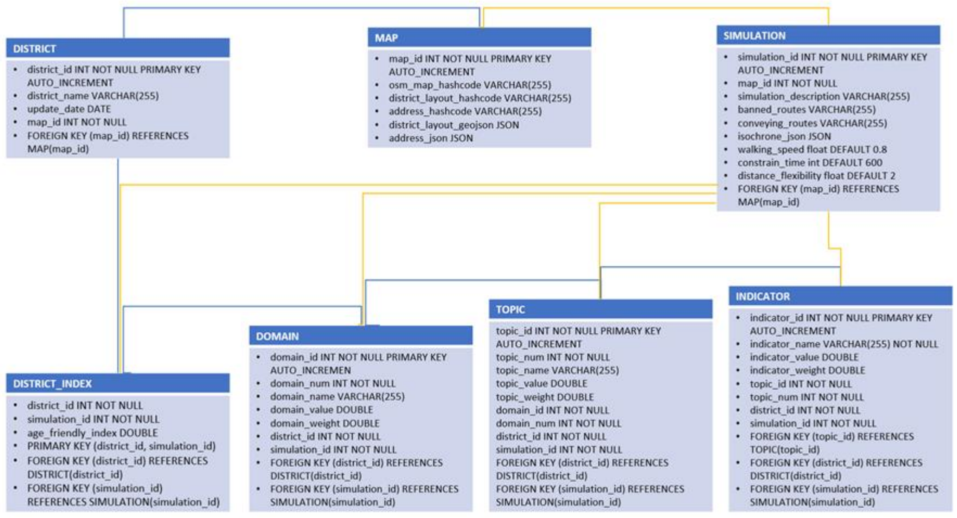
Each **isochrone** inside this isochrone json contains a response of the Age-friendly route planner when asked for an isochrone, which is a geojson that contains a Feature Collection with a MultiPolygon. MultiPolygon is returned because multiple isochrones can be computed at once if multiple cutoffSec parameters are provided but for the simulation only one is used. The structure of the following geojson can be obtained with this example.

The service data was provided by Digital Twin and was reduced to the minimum required information which are csv files containing services of the following types: toilets, benches, libraries, civic centers, health centers, local stores, worship places, entry points to open spaces, drinkable water fountain, bins and kiosks. These files have the following structure shown in the figure x below:



	A	B	C	D	E	F
1	ID	TIPO	CODIGO	FOTO	CoordX	CoordY
2	0	ASEO	ASEO_090_0001	https://santander.es/imagenesgis/Mobiliario/Aseos/ASEO_090_0001.JPG	4,36221E+15	4,81416E+16
3	1	ASEO	ASEO_216_0001	https://santander.es/imagenesgis/Mobiliario/Aseos/ASEO_216_0001.JPG	4,3487E+15	4,8123E+16
4	2	ASEO	ASEO_125_0001	https://santander.es/imagenesgis/Mobiliario/Aseos/ASEO_125_0001.JPG	4,36555E+15	4,81377E+16
5	3	ASEO	ASEO_185_0001	https://santander.es/imagenesgis/Mobiliario/Aseos/ASEO_185_0001.JPG	4,37356E+15	4,81299E+16
6	4	ASEO	ASEO_164_0001	https://santander.es/imagenesgis/Mobiliario/Aseos/ASEO_164_0001.JPG	4,36271E+15	4,81306E+16
7	5	ASEO	ASEO_214_0001	https://santander.es/imagenesgis/Mobiliario/Aseos/ASEO_214_0001.JPG	4,33541E+15	4,81253E+16
8	6	ASEO	ASEO_140_0001	https://santander.es/imagenesgis/Mobiliario/Aseos/ASEO_140_0001.JPG	4,34809E+15	4,81335E+16
9	7	ASEO	ASEO_268_0001	https://santander.es/imagenesgis/Mobiliario/Aseos/ASEO_268_0001.JPG	4,33169E+15	4,81143E+16
10	8	ASEO	ASEO_253_0001	https://santander.es/imagenesgis/Mobiliario/Aseos/ASEO_253_0001.JPG	4,32464E+15	4,81168E+16
11	9	ASEO	ASEO_076_0001	https://santander.es/imagenesgis/Mobiliario/Aseos/ASEO_076_0001.JPG	4,2905E+15	4,81416E+16
12	10	ASEO	ASEO_167_0001	https://santander.es/imagenesgis/Mobiliario/Aseos/ASEO_167_0001.JPG	4,37449E+15	4,81318E+16
13	11	ASEO	ASEO_167_0002	https://santander.es/imagenesgis/Mobiliario/Aseos/ASEO_167_0002.JPG	4,37846E+15	4,81306E+16

Figure 40: CSV file of a service that's going to be tested against the isochrones for the Age friendliness Index Simulation tool



DISTRICT
- district_id INT NOT NULL PRIMARY KEY AUTO_INCREMENT
- district_name VARCHAR(255)
- update_date DATE
- map_id INT NOT NULL
- FOREIGN KEY (map_id) REFERENCES MAP(map_id)

MAP
- map_id INT NOT NULL PRIMARY KEY AUTO_INCREMENT
- osm_map_hashcode VARCHAR(255)
- district_layout_hashcode VARCHAR(255)
- address_hashcode VARCHAR(255)
- district_layout_geojson JSON
- address_json JSON

SIMULATION
- simulation_id INT NOT NULL PRIMARY KEY AUTO_INCREMENT
- map_id INT NOT NULL
- simulation_description VARCHAR(255)
- banned_routes VARCHAR(255)
- conveying_routes VARCHAR(255)
- isochrone_json JSON
- walking_speed float DEFAULT 0.8
- constrain_time int DEFAULT 600
- distance_flexibility float DEFAULT 2
- FOREIGN KEY (map_id) REFERENCES MAP(map_id)

DISTRICT_INDEX
- district_id INT NOT NULL
- simulation_id INT NOT NULL
- age_friendly_index DOUBLE
- PRIMARY KEY (district_id, simulation_id)
- FOREIGN KEY (district_id) REFERENCES DISTRICT(district_id)
- FOREIGN KEY (simulation_id) REFERENCES SIMULATION(simulation_id)

DOMAIN
- domain_id INT NOT NULL PRIMARY KEY AUTO_INCREMEN
- domain_num INT NOT NULL
- domain_name VARCHAR(255)
- domain_value DOUBLE
- domain_weight DOUBLE
- district_id INT NOT NULL
- simulation_id INT NOT NULL
- FOREIGN KEY (district_id) REFERENCES DISTRICT(district_id)
- FOREIGN KEY (simulation_id) REFERENCES SIMULATION(simulation_id)

TOPIC

topic_id INT NOT NULL PRIMARY KEY AUTO_INCREMENT
topic_num INT NOT NULL
topic_name VARCHAR(255)
topic_value DOUBLE
topic_weight DOUBLE
domain_id INT NOT NULL
domain_num INT NOT NULL
district_id INT NOT NULL
simulation_id INT NOT NULL
FOREIGN KEY (district_id) REFERENCES DISTRICT(district_id)
FOREIGN KEY (simulation_id) REFERENCES SIMULATION(simulation_id)

INDICATOR
- indicator_id INT NOT NULL PRIMARY KEY AUTO_INCREMENT
- indicator_name VARCHAR(255) NOT NULL
- indicator_value DOUBLE
- indicator_weight DOUBLE
- topic_id INT NOT NULL
- topic_num INT NOT NULL
- district_id INT NOT NULL
- simulation_id INT NOT NULL
- FOREIGN KEY (topic_id) REFERENCES TOPIC(topic_id)
- FOREIGN KEY (district_id) REFERENCES DISTRICT(district_id)
- FOREIGN KEY (simulation_id) REFERENCES SIMULATION(simulation_id)

Figure 41: database structure for the age friendliness index simulation tool persistency and calculations

The database stores all the simulations, the values and weights for all KPIs, the map from which simulations are being created with its addresses and layout, the resulting isochrones for the simulation and its configuration.

4.2.2.3 Technical specifications

The simulation tool has been developed with worldwide standard web design technologies. These are, in the front-end of the tool: HTML, CSS and Bootstrap to create a responsive layout combined with JS and Ajax to manage dynamically updating the web and, lastly; Leaflet, to provide an interactive map in which polygons can be drawn. For the back-end Tomcat 8.5.82, Java EE 7 Web and MySQL 8.0 and were chosen because they are very well known, and offer all the functionalities needed to deploy a web service with a database. Also, to provide a strong data science backbone Python 3.8 was chosen because of its immense data science libraries and development speed.

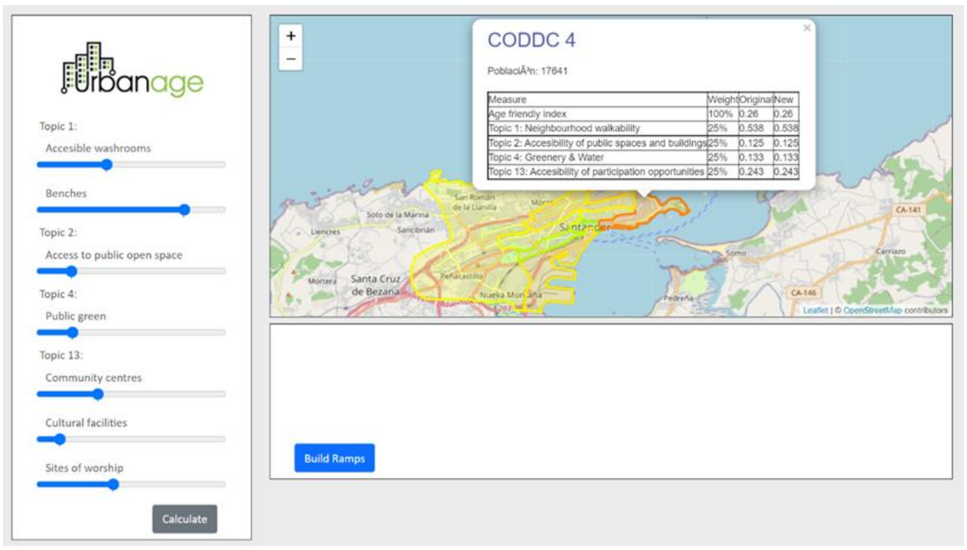
4.2.2.4 Technical contribution of the component (describe innovative aspects)

A prototype web service is developed so civil servants can quickly assess the district indexes and later on check how conveying paths (ramps, funiculars) change the indicators. This service combines client- and server-side computation to keep network traffic at its minimum and redundancy to verify results. It also includes a reverse geocoding feature to select ramps from an open-source platform (Nominatim) that also uses OSM data.

The isochrone approach provides new data exploitation capabilities, computation optimization and uses the already developed Age Friendly Route Planner which means that any further improvements to AFRP will translate to the Simulation.

The isochrone computation is managed by Python scripts that are placed behind the Python API. These scripts check if the isochrones are already computed for the specific simulation configuration. If they were not already computed Python will take the address layout json and launch requests to the Age-friendly route planner. When all isochrones are returned it will store them back in the database in the SIMULATION table. If they were already computed these isochrone json filled with will be taken from the database along with the geo-localized services that are stored locally in csv files. Then each service's coordinates will be matched against the isochrone polygons to check what is the number of services each address has access to.

4.2.3 Demonstration of the main functionalities

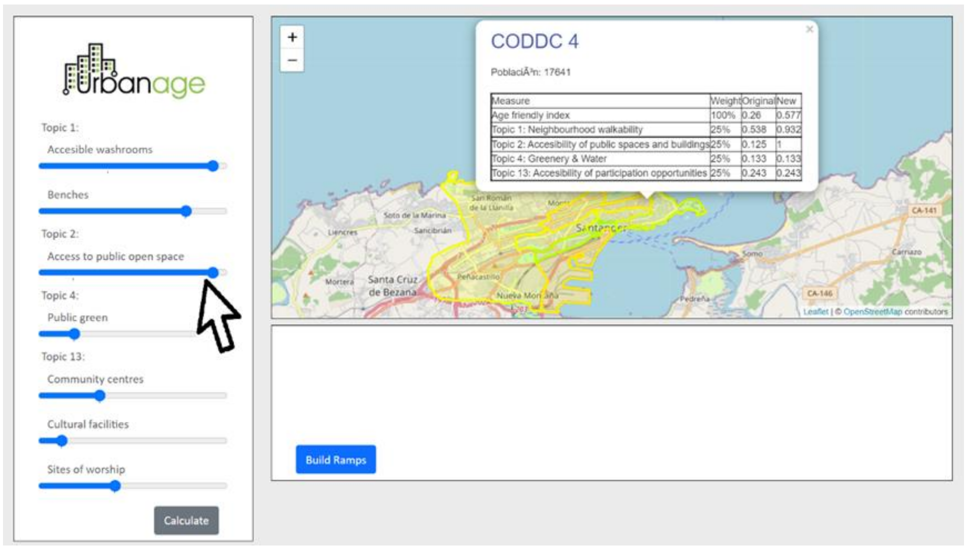


CODDC 4

PoblaciÃ³n: 17641

Measure	Weight	Original	New
Age friendly index	100%	0.26	0.26
Topic 1: Neighbourhood walkability	25%	0.538	0.538
Topic 2: Accesibility of public spaces and buildings	25%	0.125	0.125
Topic 4: Greenery & Water	25%	0.133	0.133
Topic 13: Accesibility of participation opportunities	25%	0.243	0.243

Figure 42:the simulation tool that showcases the districts of Santander and their Age Friendliness Index represented in a color gradient and all the indicator values to the left and on a table.



CODDC 4

PoblaciÃ³n: 17641

Measure	Weight	Original	New
Age friendly index	100%	0.26	0.577
Topic 1: Neighbourhood walkability	25%	0.538	0.932
Topic 2: Accesibility of public spaces and buildings	25%	0.125	1
Topic 4: Greenery & Water	25%	0.133	0.133
Topic 13: Accesibility of participation opportunities	25%	0.243	0.243

Figure 43: The simulation tool after the indicators are changed resulting in the index value being updated and the color of the district changed.

The first one to the left is a list of sliders with each indicator defined in the database for the simulation selected. These sliders can be moved and will update the index of the corresponding district that is shown on the map as can be seen in Figure 43. The "Calculate" bottom doesn't trigger any action because the indexes are updated every time the sliders are moved. Also, the original indicator values can be checked with the little line below the sliders.

The second panel to the top-right is the leaflet map with Santander's district layout and for each district, a color that matches the Age-friendly index value 0 being red, around 0.5 being orange and 1 being red. When exploring the indicators clicking the different district polygons will update the indicators to match those of the selected district and when the indicators are changed the colors will also update as can be seen in Figure 44. All the indicator modifications can be compared with the original values with the table that popups up.

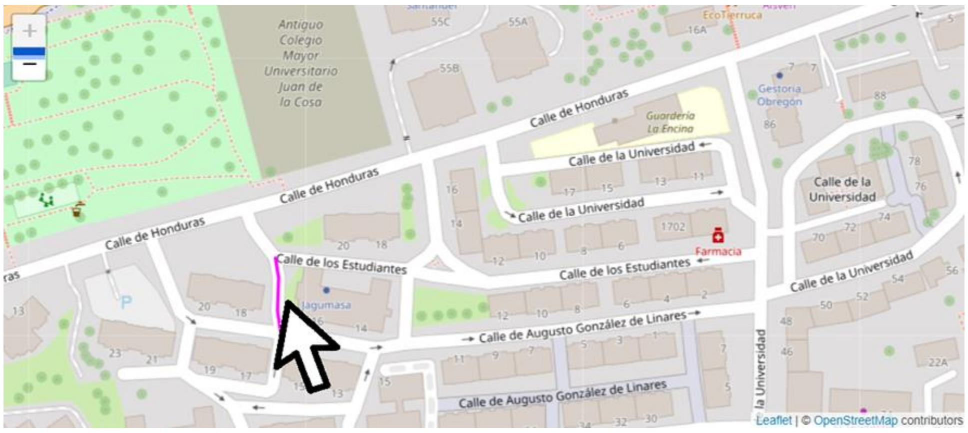

Calle de Honduras
Calle de la Universidad
Calle de los Estudiantes
Calle de Augusto González de Linares
Antiguo Colegio Mayor Universitario Juan de la Cosa
Guardería La Encina
Farmacia
Leaflet | © OpenStreetMap contributors

Figure 44: the simulation tool screenshot that shows a ramp being built after clicking into Build Ramps and then clicking on a street.

Lastly, the panel to the bottom-right is designed to contain all the simulations that are stored in the database. No simulations are shown because currently, it just loads the mockup values from the database and that's just a single simulation. In this panel, a blue button can be found. This button grants the ability to select roads where ramps are going to be built. When the button is clicked, it hides the district layout and lets the user click on the map. When clicking on the map it makes a reverse geocoding request to Nominatim to find the closest available road geometry and its osm id, as can be seen in FIGURE 3. This will later on let the user ask to create new simulations that take those newly created ramps.

4.2.4 Delivery and Usage

4.2.4.1 Package information

The repository has the following structure:

- /data
    - afis-mysql -> Scripts to initialize the MySQL database
- /dc -> Docker Compose files
- /python
    - /data -> CSV files with services and district layout
    - /src
        - /simulation -> Python Scripts for the simulation
        - app.py -> Python API that orchestrates the Python module
        - constants.py -> A configuration file that stores constants
        - Utils.py -> Mixed of functions
    - requirements.txt -> dependencies of the project that need to be installed
- /src/java -> The java webapp code

4.2.4.2 Installation instructions

First, MySQL should be installed on the machine. Start MySQL and create a new database called Urbanage. Launch "create_ddbb.sql" script inside mysql, then "create_mockup.sql". This will create a mockup database that will provide a showcase of the functionalities.
Secondly, a virtual environment for Python 3.8 should be built. After its built, install all the dependencies in requirements.txt and then, start the Python API.
Lastly, the Java service can be started.

4.3 Street Furniture Detection

4.3.1 Functional description

The Green Comfort Index is calculated from POI's which are derived from several potential sources such as OSM and municipal open data portals. However, for many municipalities, these POI's are either not available, not shared publicly or badly outdated. These so-called *data deserts* have very low green comfort scores, introducing a bias which is rooted in lack of funding for digitalisation efforts or lack of initiative to perform this effort. This however reflects badly towards elderly citizens as these locations may seem inhospitable from the perspective of the Green Comfort Index calculations.

To alleviate these shortcomings in a cost-effective way, street view datasets such as Google's Streetview and Mapillary Vistas can offer high quality imagery along a street network. Using object detection models, one can detect the presence of street furniture in these photographs, resulting in an alternative data layer to correct the green comfort score biases.

Pro's and con's of both alternatives:

Open source data (OSM, open data)

➕

- accurate coordinates
- no double counting

❓

- data only good if it is well maintained

➖

- not available in data deserts

Street furniture model on streetview data

➕

- works in data deserts (if streetview imagery is available)

❓

- data is only good if it is well maintained

➖

- less accurate coordinates
- potentially double-counting
- doesn't discriminate between street furniture from public space or private street furniture
- street furniture that is blocked from view (e.g. by a passing van) or not visible from the street (e.g. behind vegetation) is not included

4.3.2 Technical description

4.3.2.1 Justification

Models that allow for building an inventory in urban scenery exist but are not always adapted to the use-case that is called for. Some of them are used for semantic segmentation, while what we really need is object detection model. Furthermore, they are often not pre-trained on specific elements contributing to (elderly) comfort. The specific street furniture required for running the green comfort index need to be attuned for, furthermore street furniture can be vastly different looking depending on the city that is being considered.

Some detail on existing models are given below:

Model	Architecture	Task	Training dataset	Cost	Issues
YOLO v7	CNN with bounding box regression	object detection	COCO	Open source	has a lot of unwanted objects and some missing objects (such as bus stops and drinking water fountains)
CityScapes models	CNN	semantic segmentation	Cityscapes	Open source or code not available	wrong task, wrong object classes

Table 7: Existing models for urban scenery inventory – architecture and tasks

To tackle the documented shortcomings, we can benefit from transfer learning to finetune an existing model architecture (with pre-trained weights) to the object classes that we're interested in. This allows us to combine state of the art performance with homemade annotations, using small(er) sample sizes and less computation than would be needed when training a model from scratch.
The object detection models that were initially under consideration are:

- YOLOv7
- facebookresearch's DETR[10]
- IDEA-Research's DINO[11]

The choice for YOLOv7 fell due to:

- ease of retraining (it having it's own - rather intuitive - labelling format & directory structure)
- high (self-)reported Average Precision
- good documentation
- good reputation of YOLO model architecture
- (relatively) low resource requirements

[10] https://github.com/facebookresearch/detr
[11] https://github.com/IDEA-Research/DINO

We tabulate some important technical characteristics of the selected model and training dataset increments:

Characteristic	Description	Source
Model name	YOLOv7	source code & paper
Model license	GPL3	license
Model class	AI > Image recognition > Convolutional neural network	
Task	Object detection	
Training environment	GPU enabled HPC clusters: • 2 GPU's (2x NVIDIA GeForce GTX 1080 Ti, 11178.375MB) 8cpu's, 32gb RAM per CPU	
Training dataset v1	Google StreetView	source code & report
Training dataset v2	previous + Mapillary	source code & report
Training dataset v3	previous + Internet Search Results	google image search + batch image downloader & report

Table 8: Existing models for urban scenery inventory - sources

Note that the characteristics of the datasets can be found in the source code and reports.

4.3.2.2 Architecture

4.3.2.2.1 Model training workflow

The model training workflow is illustrated in the diagram below:

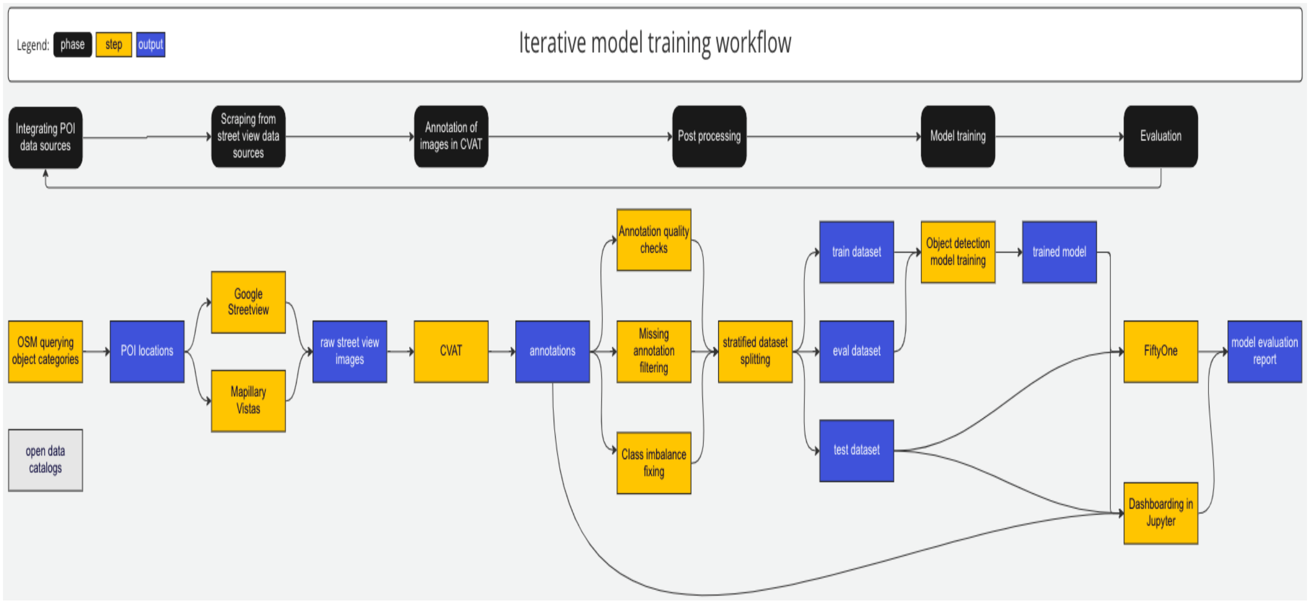

Legend: phase step output
Iterative model training workflow
Integrating POI data sources
Scraping from street view data sources
Annotation of images in CVAT
Post processing
Model training
Evaluation
OSM querying object categories
open data catalogs
POI locations
Google Streetview
Mapillary Vistas
raw street view images
CVAT
annotations
Annotation quality checks
Missing annotation filtering
Class imbalance fixing
stratified dataset splitting
train dataset
eval dataset
test dataset
Object detection model training
trained model
FiftyOne
Dashboarding in Jupyter
model evaluation report

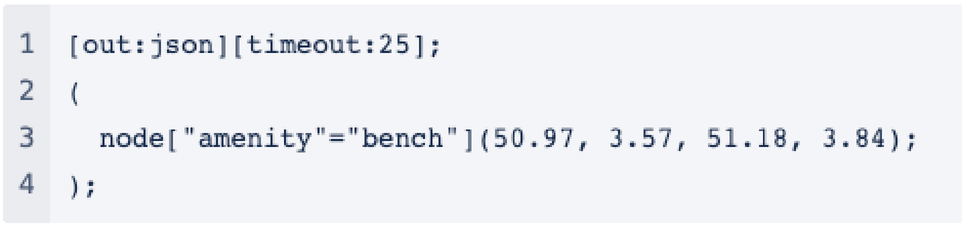
Figure 45: Workflow diagram

We briefly dive deeper into some elements of this workflow:

4.3.2.2.1.1 OpenStreetMap (OSM) querying

OpenStreetMap is a reliable, unique and open source which is heavily relied upon by the GIS community for qualitative and accessible location-based data. Using the Overpass Turbo API[12], one can easily query a given area of interest for specific POI, using the Overpass QL (querying language).

An example query for benches:



```
[out:json][timeout:25];
(
  node["amenity"="bench"](50.97, 3.57, 51.18, 3.84);
);
```

Figure 46: Example query for benches

The bounding on the bottom part is focussed on the city of Ghent. The results are visualized in map form as such:

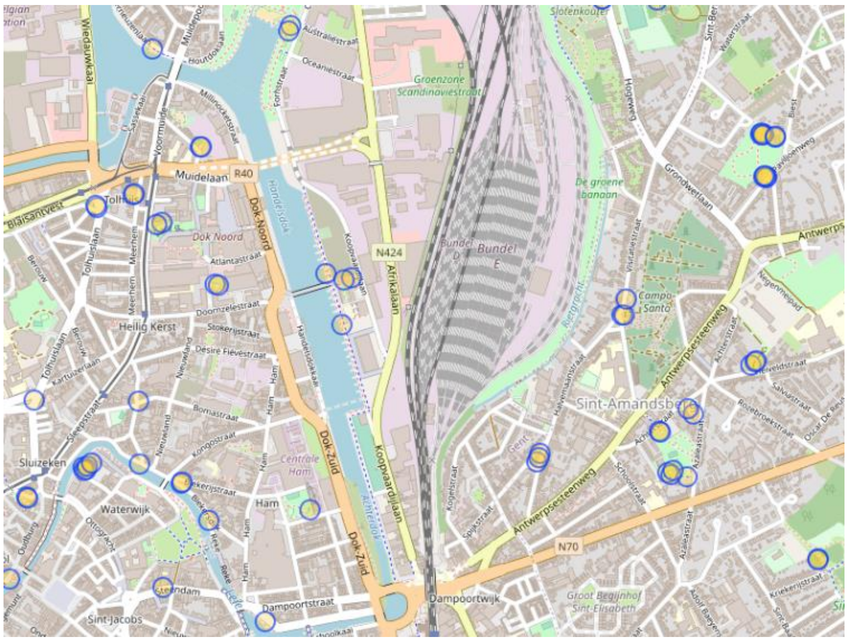
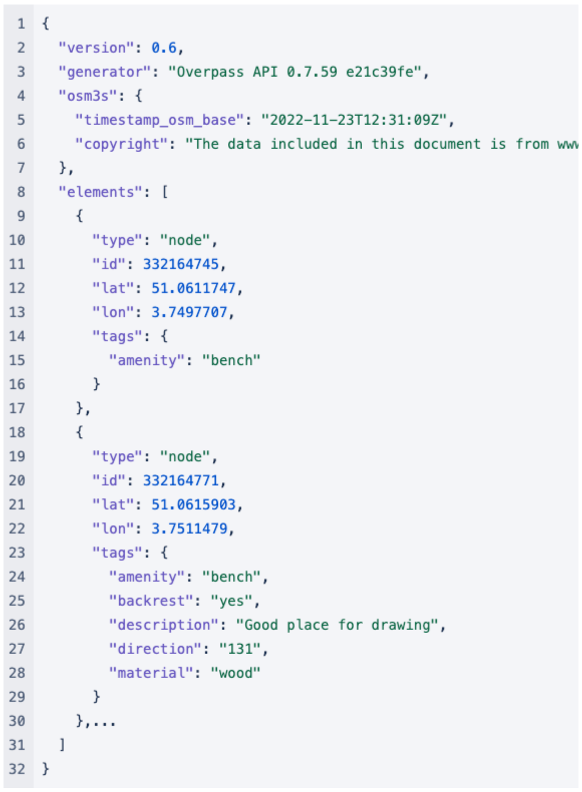


Figure 47: Visualization of query results

and look like this in JSON format:

12 https://overpass-turbo.eu/



```
{
  "version": 0.6,
  "generator": "Overpass API 0.7.59 e21c39fe",
  "osm3s": {
    "timestamp_osm_base": "2022-11-23T12:31:09Z",
    "copyright": "The data included in this document is from www
  },
  "elements": [
    {
      "type": "node",
      "id": 332164745,
      "lat": 51.0611747,
      "lon": 3.7497707,
      "tags": {
        "amenity": "bench"
      }
    },
    {
      "type": "node",
      "id": 332164771,
      "lat": 51.0615903,
      "lon": 3.7511479,
      "tags": {
        "amenity": "bench",
        "backrest": "yes",
        "description": "Good place for drawing",
        "direction": "131",
        "material": "wood"
      }
    },...
  ]
}
```

Figure 48: Query results in json format

4.3.2.2.1.2 Google Streetview data

The generation of street network points and downloading of google streetview data was easily facilitated with the use of Streetviewdownloader's[13] python AP. This tool provides a powerful high-level interface for downloading streetview imagery.

4.3.2.2.1.3 Mapillary Vistas data

Mapillary offers a viable high-quality image alternative to Google Streetview. The downloading of imagery can be done for free and without limitations.

4.3.2.2.1.4 Computer Vision Annotation Tool for dataset annotation

To facilitate multi-user image annotation, the online CVAT (computer vision annotation tool) was set up for performing annotation on the street view dataset.

[13] https://svd360.istreetview.com/

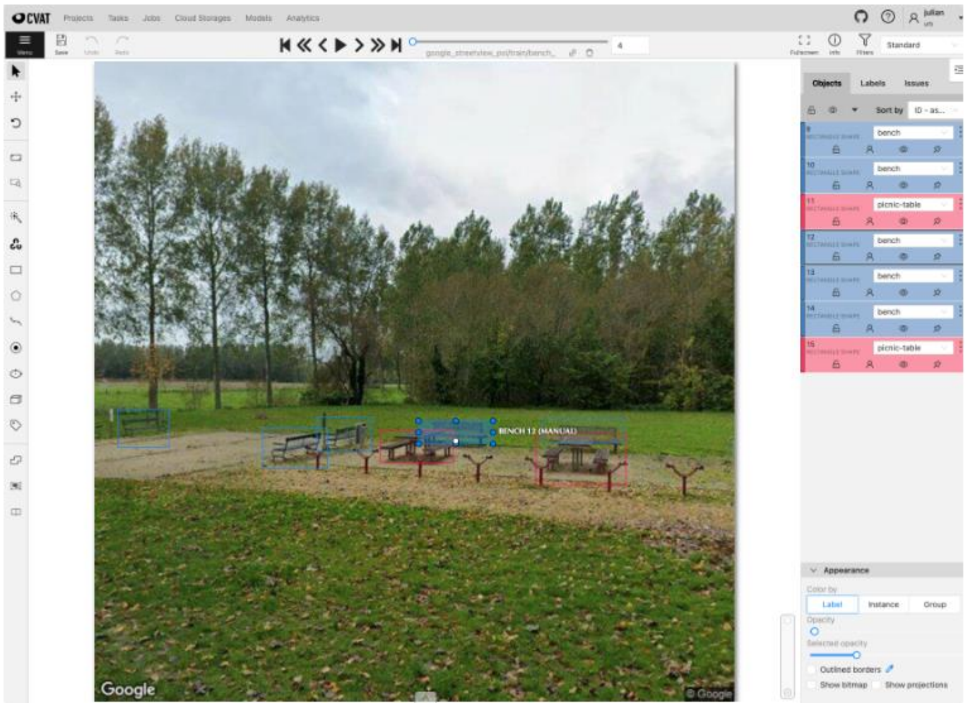


Figure 49: The Computer Vision Annotation Tool in action

A detailed report of the labelthon demo (an image labelling hackathon) is available[14].
Following from this, an image quality recommendation report[15] was created. This report details considerations when dealing with a modelling effort on the source data quality, such as recommended practices, sample sizes, workflows and noteworthy issues to check for.

4.3.2.2.1.5 Object detection model training

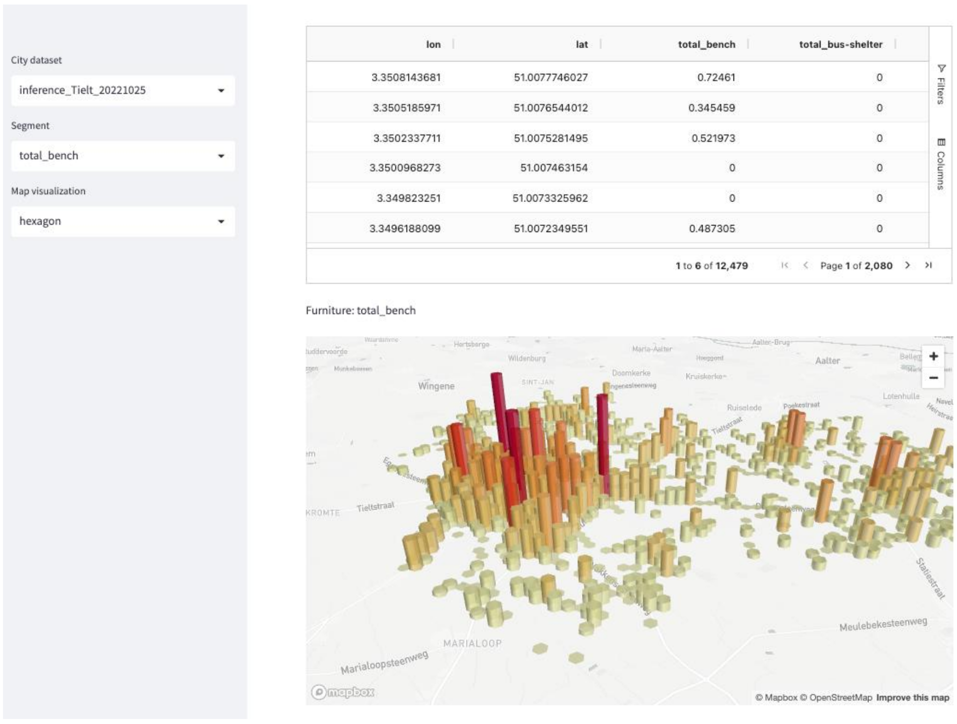
For the (re)training of the object detection model, the YOLOv7 architecture was used. With transfer learning, one can use the model weights of a previous training effort, to finetune the model to our specific task: detecting street furniture in the Flemish context. The training was done in a GPU-powered JupyterLab environment. A notebook was used for model retraining details the necessary steps for performing data gathering, configuration setup, weights downloading, model loading & retraining and outputting and securing the resulting model weights.

[14] https://imec.atlassian.net/wiki/spaces/URB/pages/4820598968/Demo+27+09+2022+-+How+to+detect+street+furniture+from+images+in+the+wild+using+transfer+learning+on+computer+vision+models+and+manually+annotated+data
[15] https://imec.atlassian.net/wiki/spaces/URB/pages/4850450481

Training the combined (Google Streetview + Mapillary + Internet Search Results) dataset for 500 epochs takes about 24.5 hours (87 425 seconds) on a CUDA enabled GPU lab environment with 2 NVIDIA GeForce GTX 1080 Ti (11 178MB) GPU's.

4.3.2.2.1.6 Streamlit dashboard

A dashboard was developed using streamlit, to visualize the output of inference on the city-wide datasets.



City dataset: inference_Tielt_20221025

Segment: total_bench

Map visualization: hexagon

lon	lat	total_bench	total_bus-shelter
3.3508143681	51.0077746027	0.72461	0
3.3505185971	51.0076544012	0.345459	0
3.3502337711	51.0075281495	0.521973	0
3.3500968273	51.007463154	0	0
3.349823251	51.0073325962	0	0
3.3496188099	51.0072349551	0.487305	0

1 to 6 of 12,479 Page 1 of 2,080

Furniture: total_bench

Figure 50: Streamlit dashboard output for the city of Tielt

An example of the output for the city of Tielt is visible below. In the dashboard, the chose object class to be visualized in "bench".

4.3.2.2.1.7 Inference workflow

The inference workflow is illustrated in the diagram below:

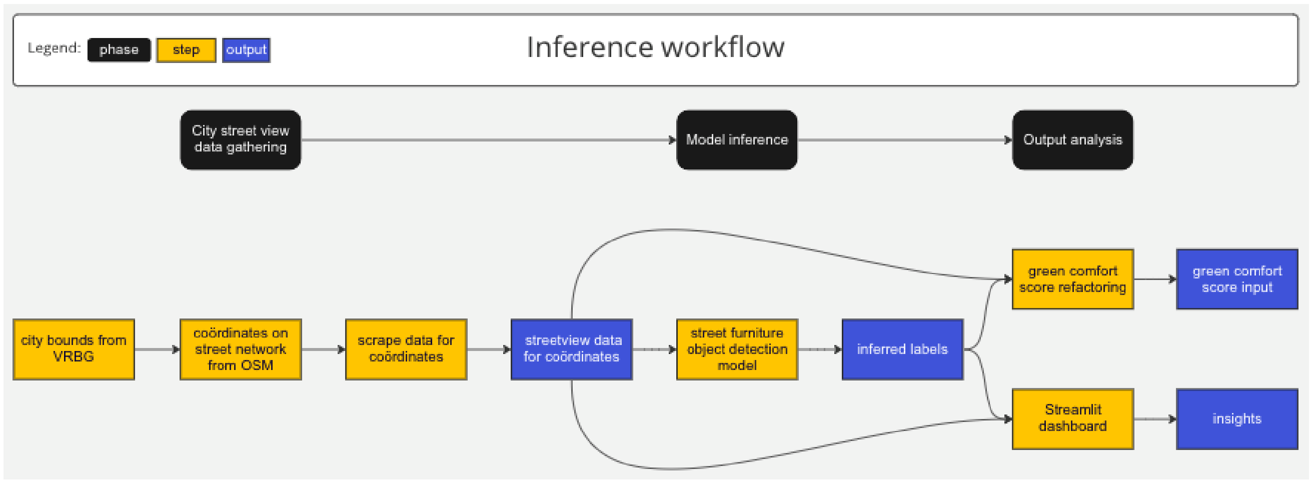

Legend: phase step output
Inference workflow
City street view data gathering
Model inference
Output analysis
city bounds from VRBG
coördinates on street network from OSM
scrape data for coördinates
streetview data for coördinates
street furniture object detection model
inferred labels
green comfort score refactoring
green comfort score input
Streamlit dashboard
insights

Figure 51: The inference workflow of the street furniture detection model

VRGB for city bounds. The VRGB file is a temporary register for city borders, and is made available in the shapefile format. You can easily filter for the desired city by its' name or postal code. A simplified WKT version (using geopandas' "simplify" function) is used to make the WKT representation smaller. This compressed representation is then fed into the streetviewdownloader point generator.

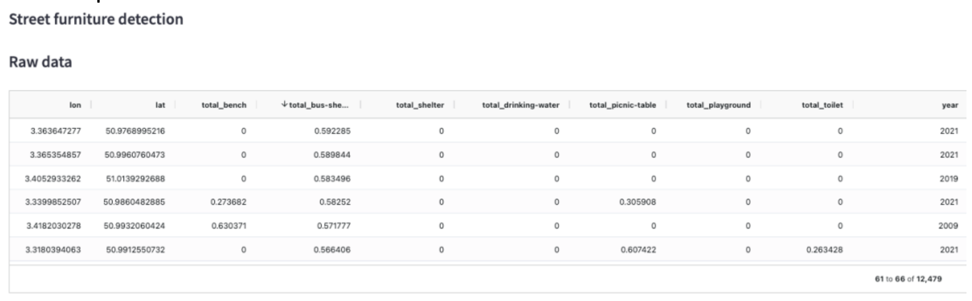
4.3.2.2.1.8 Streetviewdownloader point generator

This component generates points on the street network, set at a given distance from eachother. These coordinates are then used to feed to the Mapillary of streetviewdownloader python API, after which the desired panorama's are downloaded.

4.3.2.2.2 Green comfort score refactoring

The object detections were aggregated to the total of their confidence values per coördinate / panorama. This is an output that can serve as a (partial) replacement for green comfort score computation
The resulting dataset looks like this:

Format specification:

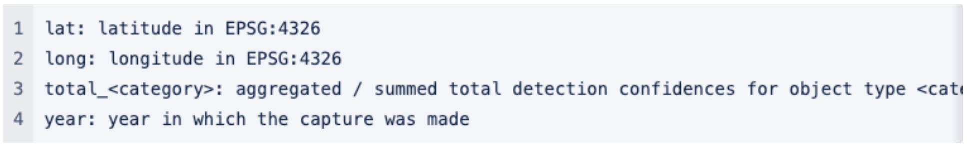


Street furniture detection

Raw data

lon	lat	total_bench	↓total_bus-she...	total_shelter	total_drinking-water	total_picnic-table	total_playground	total_toilet	year
3.363647277	50.9768995216	0	0.592285	0	0	0	0	0	2021
3.365354857	50.9960760473	0	0.589844	0	0	0	0	0	2021
3.4052933262	51.0139292688	0	0.583496	0	0	0	0	0	2019
3.3399852507	50.9860482885	0.273682	0.58252	0	0	0.305908	0	0	2021
3.4182030278	50.9932060424	0.630371	0.571777	0	0	0	0	0	2009
3.3180394063	50.9912550732	0	0.566406	0	0	0.607422	0	0.263428	2021

61 to 66 of 12,479



```
lat: latitude in EPSG:4326
long: longitude in EPSG:4326
total_<category>: aggregated / summed total detection confidences for object type <cat
year: year in which the capture was made
```

Figure 52: Resulting dataset

4.3.2.3 Data used

The following data sources were consulted for model training, testing and evaluation

4.3.2.3.1 Mapillary

Mapillary[16] is a free alternative that offers vistas (street view images) and analytics on top of them.

4.3.2.3.2 Google Street View

Google StreetView has a pay per use model and offers panorama's of high quality.

4.3.2.3.3 Image Search results

If quality annotated data is lacking, a simple Image search can yield promising results to enrich you existing datasets.

4.3.2.3.4 Streetview datasets on complete cities

Full-blown city wide coverage datasets were downloaded for the following cities:

- Brugge (38.884 coordinates; 18,83gb zipfile)
- Gent (23.812 coordinates; 10,75 gb zipfile)
- Leuven (22.588 coordinates; 10,27 gb zipfile)
- Roeselare (21.905 coordinates; 8,51 gb zipfile)
- Tielt (12.479 coordinates; 5,03 gb zipfile)

4.3.2.3.5 Data quality concerns

Several quality issues may exist to do with the image itself, which is notably the case with many of the Mapillary images.

Some typical quality issues are:

- rotations (for instance 45 degrees upwards or downwards)
- warping (panoramic cameras do this)
- artifacts such as blurry spots, cracks
- variable resolution

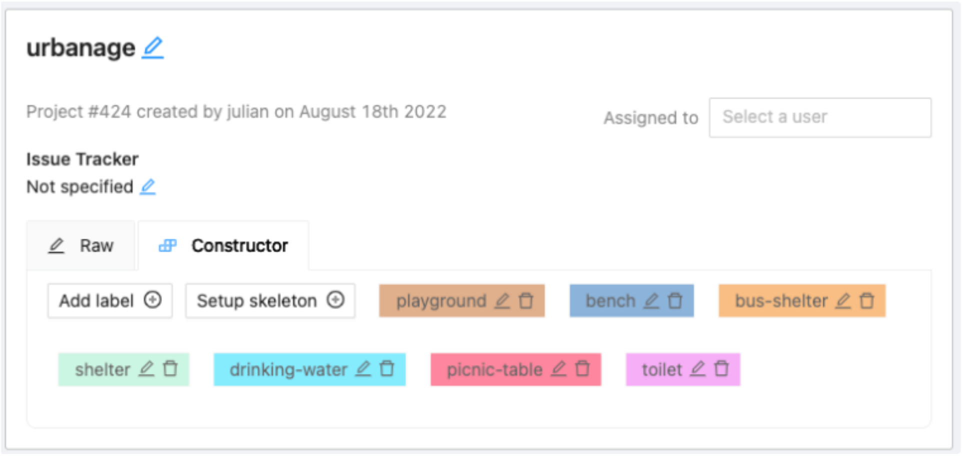
[16] https://www.mapillary.com/?locale=es_ES

- grainy image due underexposure (typically when taken during night or evening with low exposure time)
- grainy image due to movement of camera at time of capture
- grainy / blurry image due to out-of-focus
- …

4.3.2.4 Technical specifications

4.3.2.4.1 Annotation process

Within CVAT.ai[17] you can create a project to which the desired object categories (labels) are assigned.



urbanage

Project #424 created by julian on August 18th 2022

Assigned to Select a user

Issue Tracker

Not specified

Figure 53: Details within Computer Vision Annotation Tool for the Urbanage project

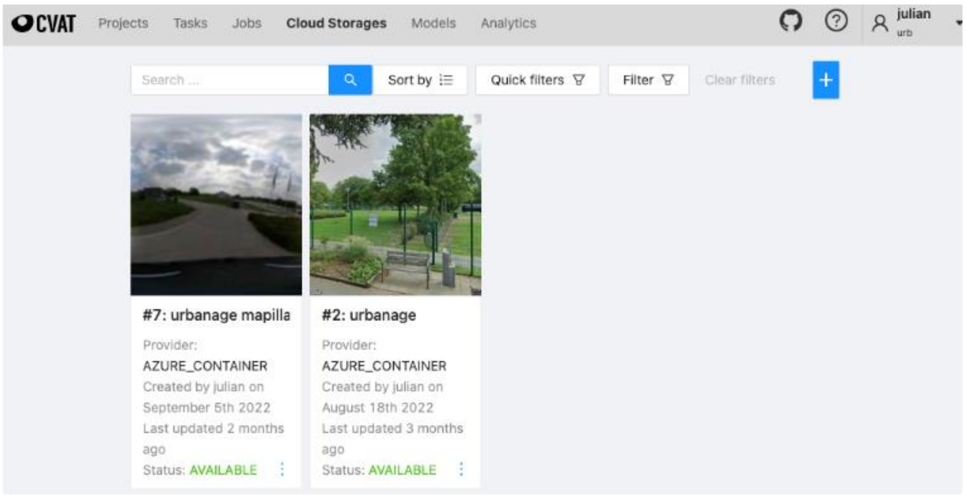
The labels / object categories are as follows (number serves as codification within annotation files):

```
bench (0)
shelter (1)
bus-shelter (2)
toilet (3)
drinking-water (4)
playground (5)
picnic-table (6)
```

Figure 54: Labels / object categories for the annotations

[17] https://www.cvat.ai/

All the selected images are uploaded to cloud storage. This cloud storage is then attached to the CVAT workspace. A manifest file is created and uploaded to the same Azure Blob Container. This manifest will allow us to annotate the images.

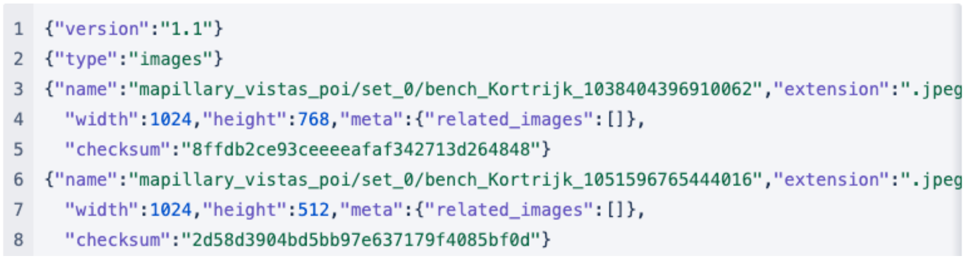


#7: urbanage mapilla
Provider:
AZURE_CONTAINER
Created by julian on September 5th 2022
Last updated 2 months ago
Status: AVAILABLE

#2: urbanage
Provider:
AZURE_CONTAINER
Created by julian on August 18th 2022
Last updated 3 months ago
Status: AVAILABLE

Figure 55: Cloud storages as they are registered in the Computer Vision Annotation Tool

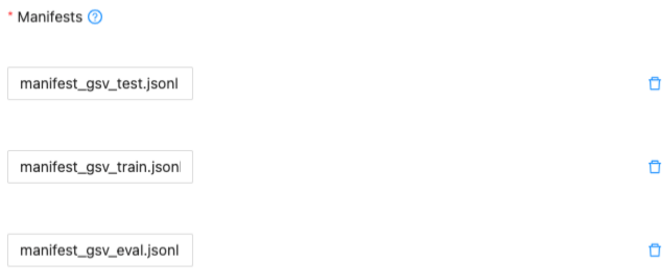
The manifests in this cloud storage reflect the dataset slices that are used for model training: train, eval & test. A manifest file is - as the extension suggests - a jsonl (json lines) file, containing the necessary metadata to perform the annotation task. Below a manifest fragment was added for reference.



```
{"version":"1.1"}
{"type":"images"}
{"name":"mapillary_vistas_poi/set_0/bench_Kortrijk_1038404396910062","extension":".jpeg
  "width":1024,"height":768,"meta":{"related_images":[]},
  "checksum":"8ffdb2ce93ceeeeafaf342713d264848"}
{"name":"mapillary_vistas_poi/set_0/bench_Kortrijk_1051596765444016","extension":".jpeg
  "width":1024,"height":512,"meta":{"related_images":[]},
  "checksum":"2d58d3904bd5bb97e637179f4085bf0d"}
```

Figure 56: Format of a manifest file

In the CVAT annotation workspace, for each manifest, a task is created within the urbanage project -which has a predetermined set of labels.

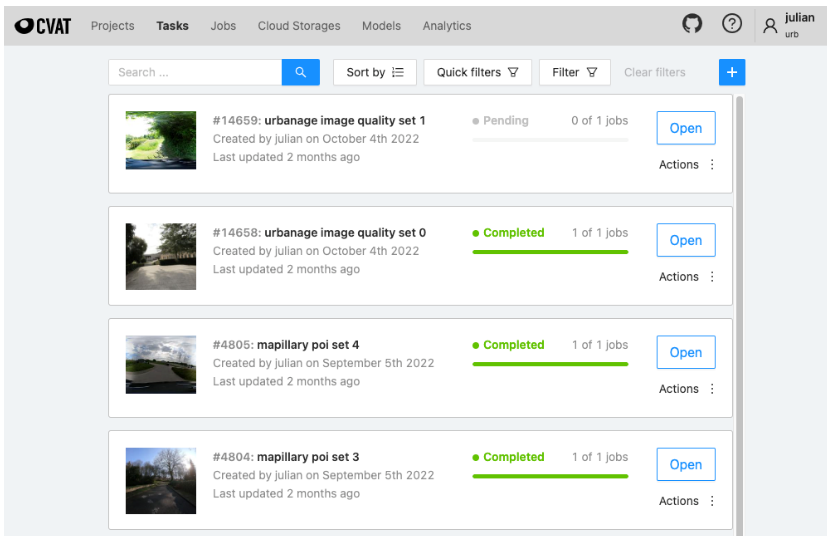


* Manifests

manifest_gsv_test.jsonl

manifest_gsv_train.jsonl

manifest_gsv_eval.jsonl



#14659: urbanage image quality set 1 Created by julian on October 4th 2022 Last updated 2 months ago	Pending	0 of 1 jobs	Open Actions
#14658: urbanage image quality set 0 Created by julian on October 4th 2022 Last updated 2 months ago	Completed	1 of 1 jobs	Open Actions
#4805: mapillary poi set 4 Created by julian on September 5th 2022 Last updated 2 months ago	Completed	1 of 1 jobs	Open Actions
#4804: mapillary poi set 3 Created by julian on September 5th 2022 Last updated 2 months ago	Completed	1 of 1 jobs	Open Actions

Figure 57: Annotation tasks within the Computer Vision Annotation Tool

These tasks can then be assigned to different users, which can get to annotating the images. Once the annotations are done, these can be exported in a desired format, such as YOLO5, Pascal VOC of COCO.

A "labelthon" - featuring pizza - was organized to speed up the labelling process. Thanks a lot Jan, Maarten, David, Ismail, Mircea & Jef for participating in the labelling effort!

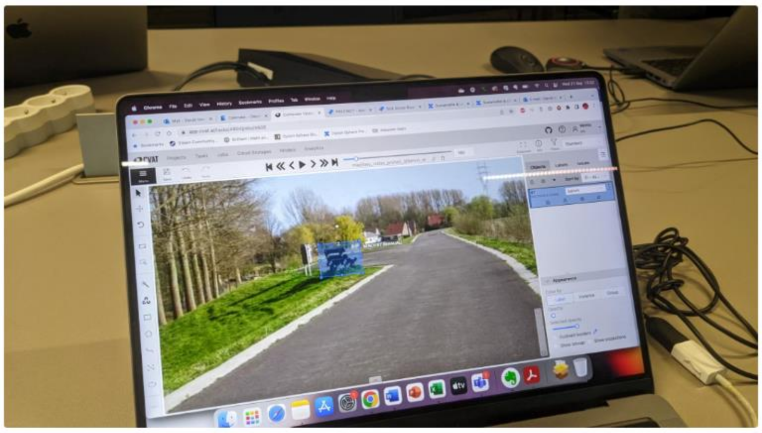


Figure 58: Hard at work during the labelthon - a close up

For our use-case, YOLO5 suffices as an export format. This format saves one TXT file per image file. these TXT files contain the following format, where **bb** means "bounding box", the rectangle around the object to be detected:

```
<object id> <bb x mid> <bb y mid> <bb width> <bb height>
```

Figure 59: YOLOv5 labelling format

`<bb x mid>`, `<bb y mid>`, `<bb width>` and `<bb height>`

are all numbers between 0 and 1, indicating the relative size with respect to the image width and image hight. The object id is a number from 0 up to the number of object types available. (0=bench, 1=shelter, 2=...)
Example:

```
0 0.034646 0.627701 0.069292 0.076792
5 0.486750 0.492403 0.756604 0.577361
```

Figure 60: Example of 2 labels

A simple preparation is needed to match the desired / simplified document structure for retraining object detection models. For the YOLOv7 model this comes down to creating a directory structure looks as follows:

```
train>
  images
    image_1.jpg
    image_2.jpg
    ..
  labels
    image_1.txt
    image_2.txt
    ..
eval
  images
    image_102.jpg
    image_202.jpg
    ..
  labels
    image_102.txt
    image_202.txt
    ..
test
  images
    image_1111.jpg
    image_22222.jpg
    ..
  labels
    image_1111.txt
    image_22222.txt
    ..
```

Figure 61: Folder layout for retraining a YOLO model

Note that the image and label files can be random, but the names within a set (train, test or eval) should be coupled, as in each <image_name>.jpg should have a corresponding <image_name>.txt file.

4.3.2.4.2 Model selection

A number of models was taken into consideration for the object detection task. These are listed in the table below:

Model	Full title	Originator	Comments	Dataset format	Licence
YOLO v7[18]	YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors (YOLO stands for **Y**ou **O**nly **L**ook **O**nce)	Institute of Information Science, Academia Sinica, Taiwan	easy to get started, code available, pre-trained weights available	YOLOv5	GPL3
DETR[19]	DE⋮TR: End-to-End Object Detection with Transformers	facebook / meta	somewhat easy to get started	COCO	Apache 2.0
DINO[20]	DINO: DETR with Improved DeNoising Anchor Boxes for End-to-End Object Detection	IDEA, the International Digital Economy Academy	somewhat easy to get started	COCO	Apache 2.0
EVA[21]	Exploring the Limits of Masked Visual Representation Learning at Scale	BAAIvision	not available at start of the project, less easy to get started	Detectron custom format or COCO	MIT

Table 9: Existing models for the object detection task

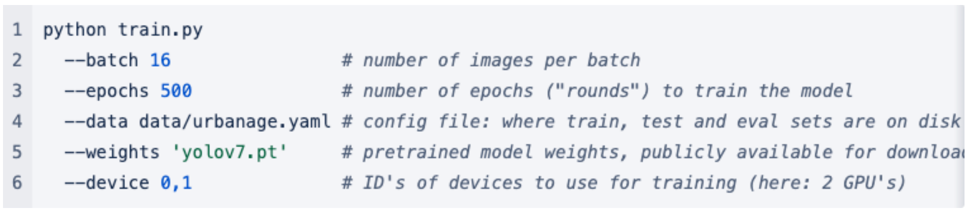
The choice of a model is one of personal taste, and completely interchangeable. All of these solutions were developed in Python, using the Pytorch framework, and require GPU enabled systems for training and inference. The list of SOTA models can be reviewed on the website[22], where leaderboards exist for a number

[18] https://github.com/WongKinYiu/yolov7
[19] https://github.com/facebookresearch/detr
[20] https://github.com/IDEA-Research/detrex/tree/main/projects/dino
[21] https://github.com/baaivision/EVA/tree/master/eva
[22] https://paperswithcode.com/sota/object-detection-on-coco

of benchmark datasets. The model itself typically doesn't need any changing, simply creating an input dataset of the correct format and define the required labels will suffice.

4.3.2.4.3 Model training

Model training was performed on the Imec / UGent GPU lab environment which offers powerful GPU's in a HPC (high-performance computing) environment. As the workflow suggests, the data science workflow is iterative (data gathering, data preparation, model training, evaluation, data or transformation logic fixing, repeat). About 6 different model runs were initiated with each their own dataset and parametrisation. Eventually it was discovered that training the model for 500 epochs suffices to achieve maximum performance. For illustration, a bash command to perform model training in YOLOv7 is explained in more detail. For a more exhaustive set of options, you can consult the source-code[23].



```
python train.py
  --batch 16                  # number of images per batch
  --epochs 500                # number of epochs ("rounds") to train the model
  --data data/urbanage.yaml # config file: where train, test and eval sets are on disk
  --weights 'yolov7.pt'       # pretrained model weights, publicly available for downloa
  --device 0,1                # ID's of devices to use for training (here: 2 GPU's)
```

Figure 62: Example parametrization of the training python script

The weights of a pre-trained model were used to start from an already very good performance. In our case, we used the YOLOv7 weights as obtained after training on the COCO 2017 dataset.

4.3.2.4.4 Model evaluation

An object detection model is typically evaluated using the mean Average Precision measure. This is a combined measure that is very well explained in an article[24]. This number on its' own does however not give you a very detailed view of the hits and misses of the model.

A confusion matrix can show you how "confused" the model was between classes of objects:

[23] https://github.com/WongKinYiu/yolov7/blob/main/train.py#L528

[24] https://www.v7labs.com/blog/mean-average-precision#:~:text=let%27s%20dive%20in!-,What%20is%20Mean%20Average%20Precision%20(mAP)%3F,values%20from%200%20to%201.

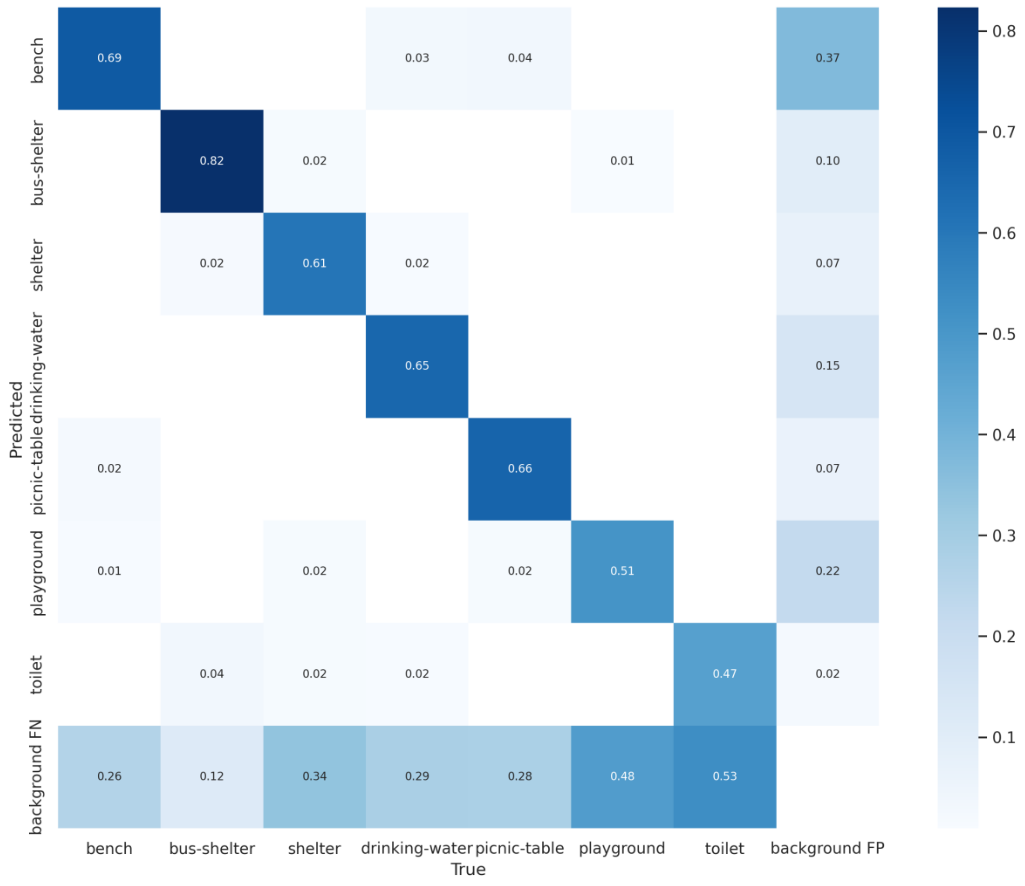


Figure 63: Confusion matrix of the trained street furniture detection model

In the confusion matrix above, we see that the quality of detection of objects is rather good, but there are many background-false negatives, meaning that a percentage of objects are undetected. Likewise, a (smaller) number of objects are detected where there should be none. A common pitfall is that many benches are detected where picnic-tables were expected. This confusion is quite understandable. To get some sen se of how the model performs on the lower level, you can use an open-source tool such as fiftyone[25]. In the screenshot below you can see the tool in action for the final dataset version.

[25] https://voxel51.com/docs/fiftyone/

FiftyOne is a tool that allows for detailed analysis and error fixing of computer vision models.

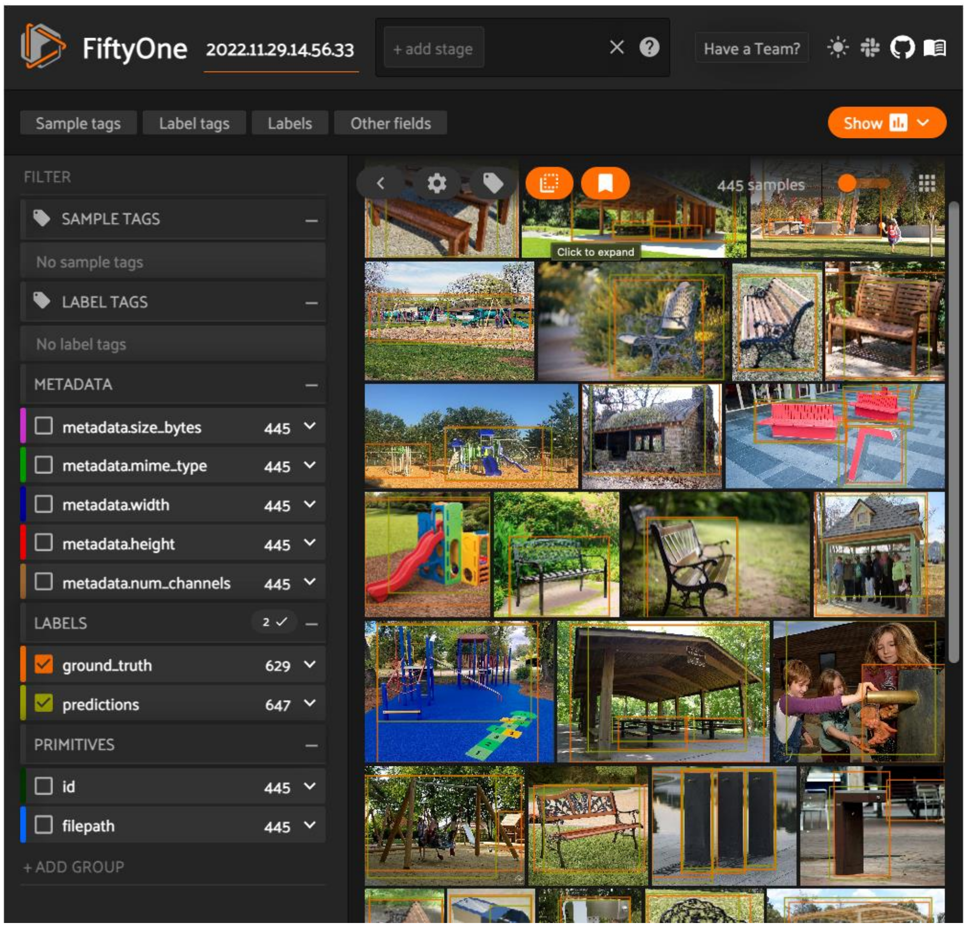


Figure 64: FiftyOne, a computer vision evaluation dashboard

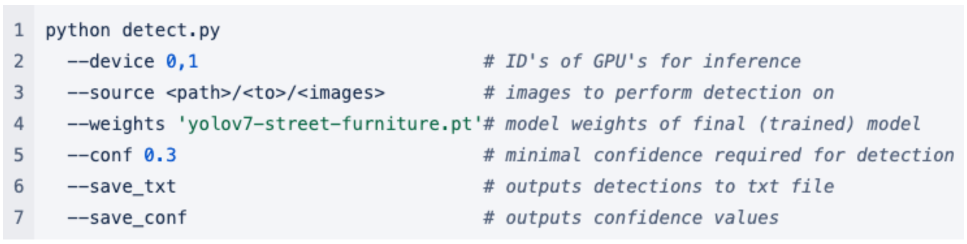
4.3.3 Demonstration of main functionalities

As explained in the functional description at the start of this chapter, we are using the street furniture object detection model to fill in the blanks of so-called *data deserts*. The input needed for calculation of Green Comfort Indices are basically locations in which we find POI's (benches, shelters, …).
The output of the images are objects, and the confidence value of their detection. Simply returning for each coordinate the list of detected objects and the sum of the confidence values allows for substituting alternative / missing data source. Below is an example of the prepared output for feeding the Green Comfort model.

```
poi_type, lat, lng, num_detected
bench, 51.0123, 4.512, 2.4
shelter, 52.0123, 4.123, 0.3
```

4.3.4 Delivery and usage

To perform detections for a given set of coordinates, you need their images scraped. You can then simply perform the detection script as illustrated below. The full set available options for YOLOv7 are documented[26].



```
python detect.py
  --device 0,1                        # ID's of GPU's for inference
  --source <path>/<to>/<images>       # images to perform detection on
  --weights 'yolov7-street-furniture.pt'# model weights of final (trained) model
  --conf 0.3                          # minimal confidence required for detection
  --save_txt                          # outputs detections to txt file
  --save_conf                         # outputs confidence values
```

4.4 Green Comfort Model

4.4.1 Functional description

4.4.1.1 Creation of the Green Comfort Index

An important outcome of the co-design workshop in WP2 was the identification of the elements that contribute to the mental and physical wellbeing and comfort of elderly inhabitants of a city. These elements were:

- Effects of heat and heat stress
- Availability of green (vegetation) and blue (water) infrastructure elements
- The accessibility and comfort levels in a broad sense

These elements were refined further into individual contributors to elderly citizen comfort:

- Presence of shade and shady areas
- Presence of vegetation and greenery in general, as separate elements or organized in parks, forests...
- Presence of water elements (ponds, rivers, fountains...)
- Good accessibility to the area
- Availability of street furniture (benches, tables, public bathrooms...)

Using the parameters above, we created a **Green Comfort Index** to allow citizens and city experts to quickly gauge the comfort levels of an area, specifically as perceived by elderly.

The Green Comfort Index or GCI was defined as:
An indication of how comfortable certain locations in the public domain of a city feel, specifically to a member of the elderly community.

[26] https://github.com/WongKinYiu/yolov7/blob/main/detect.py#L167

Citizens that will look for areas with a high Green Comfort are e.g.:

- Nursing home inhabitants who want to spend a sunny afternoon outside close to the home
- Active senior citizens who want to meet up and chat for an hour or so before going to the theater
- A senior citizen looking for a cool spot to sit and relax in the early evening after a very hot day

For the GCI values we chose an easy-to-read scale from zero to five, where 0 means extremely uncomfortable and 5 is extremely comfortable. The GCI model will round final scores to 2 decimal places. This scale allows for easy visual dissemination of the score – using color codes - while still allowing for nuanced detailed analysis.

We introduced the following color scale and legend for visualizations:

Score	Meaning	Info	Color
0-1	Very uncomfortable	Very unpleasant space to be in or to walk through. Absolutely no reasons for senior citizens to spend any leisure time here	Red 🟥
1-2	Uncomfortable	Not a comfortable place to stay in for more than an hour	Orange 🟧
2-3	Reasonably comfortable	It’s possible to spend an hour here in a reasonably comfortable way	Orange-Yellow 🟨
3-4	Very comfortable	You can comfortably spend several hours in this area, sitting, sight-seeing and chatting	Yellow-Green 🟩
4-5	Extremely comfortable	This is an extremely comfortable spot for senior citizens to spend a lot of time here	Green 🟩

Table 10: Color legend for the Green Comfort Index visualization

4.4.1.2 Capabilities of the Green Comfort Model

The Green Comfort Model allows cities and their (elderly) citizens to quantify and visualize the Green Comfort Index, which represents the general comfortableness of the public space. The model does this by calculating a Green Comfort Index for areas in a grid-like structure, using the nearby availability of street furniture, city amenities, presence of vegetation... as inputs.

In our research, we have limited the scope of Green Comfort Model executions to specific cities, or even to their inner-city limits. This allows for fairly quick model executions, with a limited amount of contributors, and a valuable result to city experts and citizens living in or nearby the city center. The model can however also be executed for larger areas such as large urban agglomerates, or even entire regions.

4.4.1.2.1 Grid system

In order to assign GCI scores to areas in the city, the model will first need to divide the city into a grid structure. Each area making up a section of the grid will get assigned a GCI score. We elaborate on the advantages of the chosen grid system further along in this document. However – functionally – it is important that the grid is visually easy to interpret and aesthetically interesting. At the same time, the grid system should allow for maximum interoperability and follow well-known standards.

When selecting the grid system, the options are limited to 3 polygon shapes: squares, hexagons and triangles. These 3 shapes are the only polygons that can tesselate[27] (i.e., be repeated over and over to cover an entire area without gaps).

We eliminated the triangle grid systems because the general public is not accustomed to the type of visualization these generate. Triangles have a large perimeter and small area which would cause a larger number of indices and therefore calculations, but a limited resulting surface area. Visually, orientation of triangles “flips” which can results in distortion of the areas (one area would “point up”, the area south of it would “point down”).

[27] https://en.wikipedia.org/wiki/Tessellation

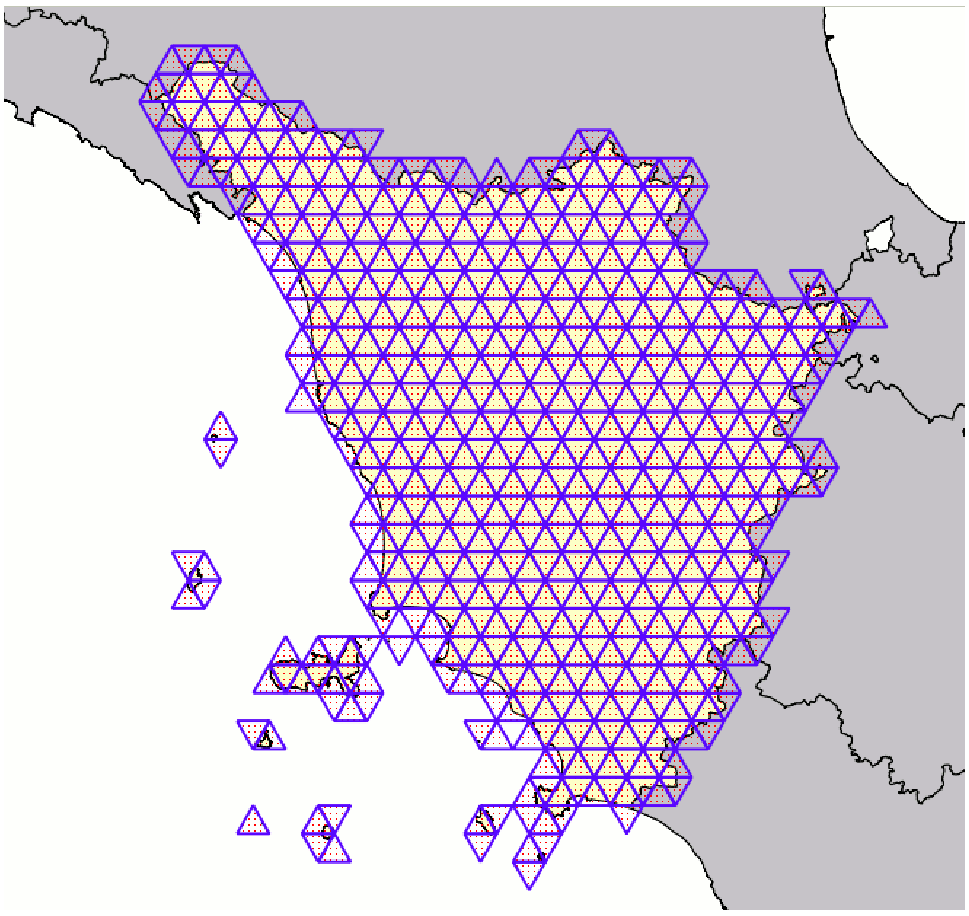


Figure 65: Triangle cell grid covering Tuscany

Although the square grid is most commonly used and is the most recognizable, we did choose to eliminate it in favor of a hexagon-based grid. Visually, hexagons will allow to detect any natural curvature of the Green Comfort Index more easily, because of their lower perimeter to area ratio and reduced sampling bias at the edges compared to squares[28]. This means a citizen can more easily see the comfortable areas on a map as a more accurate, naturally shaped geospatial space.

[28] https://www.gaia-gis.it/fossil/libspatialite/wiki?name=tesselations-4.0

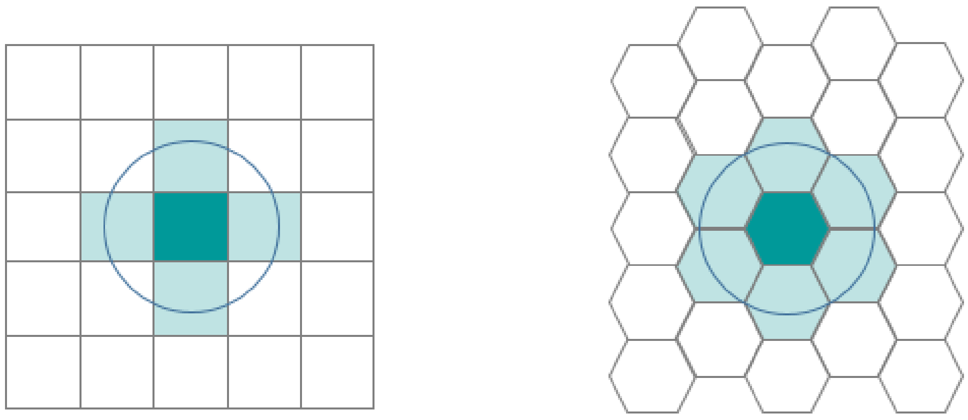


Figure 66: Comparing square and hexagon cell grid[29]

We aim for each cell in the grid to represent an area of (roughly) 300 sqm. For a hexagon shape this would mean its radius is around 10 meter or 20 meters across. In a typical Flemish city, such a shape can cover a small city center street, both of the sidewalks and parts of the building facades. In an open space, this covers an area where several people can sit and chat, without the space feeling cramped or crowded.

[29] https://pro.arcgis.com/en/pro-app/2.9/tool-reference/spatial-statistics/h-whyhexagons.htm

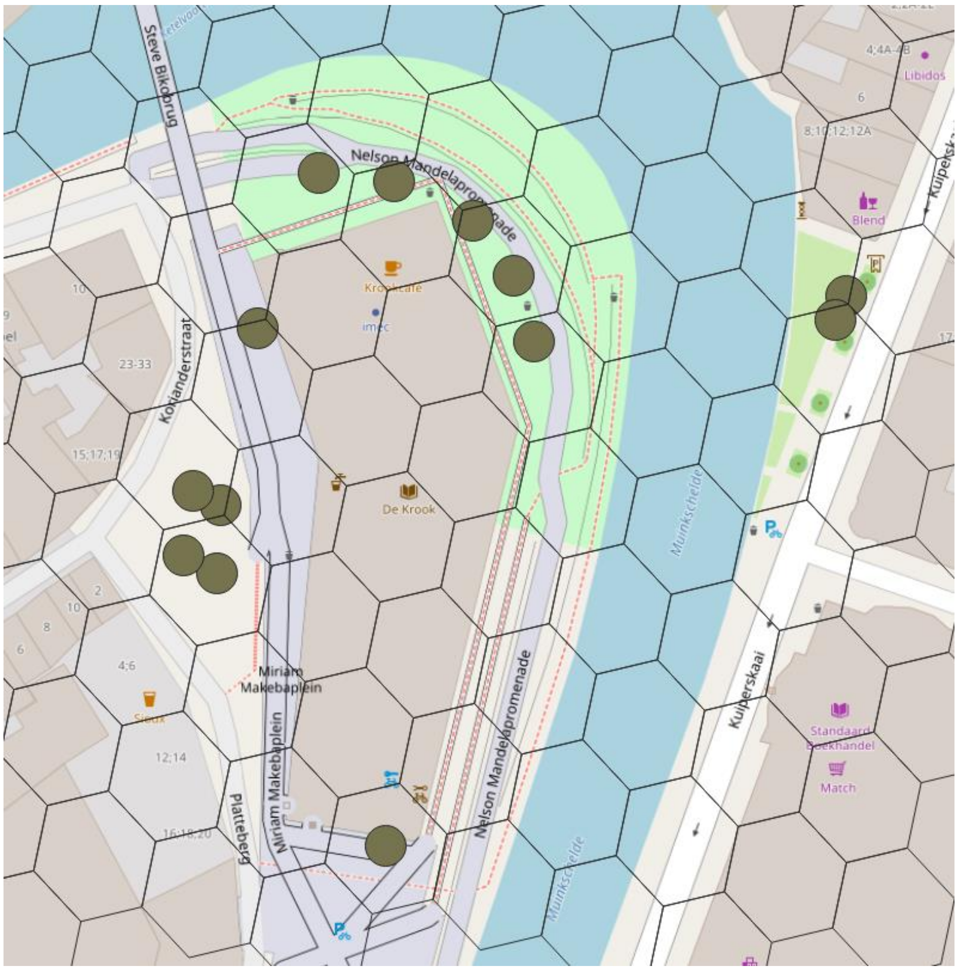


Figure 67: H3 grid with resolution 12 covering the Krook area in Ghent

4.4.1.2.2 Indicator scores

Once a grid structure has been defined, the Green Comfort model will use the different inputs provided to calculate each individual indicators contribution to the final score.

To evaluate the GCI in a given grid cell, different contributors can have a different effect. In a first iteration, the Green Comfort model takes into account the presence or vicinity of different city objects in a given cell, to determine the comfort score.

These objects will be used as static indicators, and each will contribute with a certain weight to the comfort score in the area.
For example, the score is positively impacted by:

- The presence of benches and trees in the area
- A public bathroom is located relatively close-by

- A drinking water fountain is located relatively close-by
- The area is located in a city park

Similarly, the score of the area is negatively impacted by:

- The absence of benches and trees in the area
- The nearest public bathroom is located considerably far away
- The nearest drinking water fountain is located considerably far away

The model will need to detect those indicators in relation to the hex area it's calculating a score for. Then, the model will need to score each indicator by comparing it to a value representing a perfect score (called the "**limit**"). Finally, the model will use the individual indicator **weights** to calculate the final Green Comfort Index.

The output of the calculation is a list of hexagons, each with a Green Comfort Index score assigned to it. For each GCI, the contributing indicator scores are included. E.g., the area corresponding with hex id X has a GCI score of 3,5 out of 5. The presence of benches scored 4 out of 5; the vicinity of a public bathroom scored 2 out of 5 etc.

At this point, the GCI model takes into account mainly the presence of certain amenities and vegetation. In a second iteration, the model is extended to also work with heat stress indicators. In the case of heat stress, an area where heat stress levels are high should get a lower GCI score. To achieve this behavior in the model, we introduced a penalty system, where high heat stress values cause the GCI score to lose a significant number of points, medium heat stress values cause deduction of a limited number and low heat stress levels leave the score unchanged.

4.4.1.3 Green Comfort Simulations

The GCI model allows different types of users to simulate changes to city infrastructure and evaluate the effects on the Green Comfort Index score in the affected areas.

We can distinguish different kinds of scenarios:

4.4.1.3.1 Citizen scenarios

Authenticated citizens will be able to do small-scale simulations, where they suggest the addition of a certain feature to the area, see the effects on Green Comfort, and where other users will be able to see and comment on the suggestion.

Typical simulation flow would be:

1. User navigates to certain area in the city UI (e.g., an existing park)
2. User picks a location and suggests adding a bench there (imagine dragging and dropping a bench element from a sidebar to the map)

3. A simulation for the scenario is triggered and the GCI model is fed all necessary inputs (changes to indicators)
4. New GCI scores are calculated for the immediate area (for practical reasons, we can limit this “affected area” to support large quantities of these small-scale citizen simulations).
5. The user can compare the before (= the “baseline” GCI) and after (the simulation results)
6. The user can decide to persist the scenario, at which point it becomes also visible to other users who can add comments or likes/dislikes to the suggestion

4.4.1.3.2 Expert scenarios

Experts (such as urban planners) will create larger scale scenarios, where they will start by selecting an area (e.g., the possible location for a brand-new park, or an existing park about to be remodeled). Within that location they add and/or remove different elements and will want to see the new Green Comfort scores for the area and its surroundings. Experts might create a few different scenarios for the same area, representing different lay-outs (or budgets) for the future park, and ask citizens to vote for their favorite.

Typical simulation flow would be:

1. Expert user draws area on map and chooses to create a scenario for the area
2. Expert removes certain existing POIs from the map (image bulldozing the area flat before landscaping start)
3. Expert adds different new elements to the area (trees, benches, facilities, water elements…)
4. A simulation for the scenario is triggered and the GCI model is fed all necessary inputs (changes to indicators)
5. New GCI scores are calculated for the used area and its surroundings (the new park will have an effect on areas outside the park)
6. The expert can create additional scenarios for the same area, changing the layout of the elements and seeing the effect on GC.
7. The scenarios as a whole (the area, the added elements, the GC effects) can be published and disseminated via a 3rd party application (such as, a case shared on citytwins.eu, a webpage hosted by a city requesting citizen participation…)

4.4.1.3.3 Cascading effects simulations

The simulations above work mainly with scenarios where locations and presence of POIs have been modified and how those changes directly impact the Green Comfort score.

However, there are also plenty of scenarios imaginable where an expert would modify the city landscape itself (add a building, a bike road, a water element…). These changes could potentially have an impact on the heat stress in the area (a water element results in slightly cooler, humid air; a building provides shade…), and those changing heat stress levels have an impact on the Green Comfort Index in the area.

Currently models calculating heat stress or discomfort based on changing city elements and layout accessible to Flanders are still in development. To support two-tier scenarios like the above the simulation would need to allow for:

1. Expert creates a new scenario by indicating the area in which he/she will work
2. Expert makes changes to city POIs but also city landscape characteristics like adding a bike road, changing the surface to grass or concrete, adding a 5-story building with a façade mainly consisting of glass window panes…
3. A simulation workflow for the scenario is triggered
4. The impact on heat stress of these changes is calculated by an external model
5. The impact of the changes directly (POIs) and the simulated heat stress changes on the GCI is calculated by the GCI model

4.4.2 Technical description

4.4.2.1 Technical specifications

Technology

The model is written in Python, because it is a popular programming language and the many libraries available for it. We've used popular libraries to support the model, some examples are h3-py, geojson, geopandas and shapely.

Motivation of H3

H3 is a geospatial "indexing" system created by Uber, it allows for a way to refer to coordinates (in this case, a hexagonal grid). It offers an easy API for indexing coordinates into a hexagonal grid, down to square meter resolution.[30]

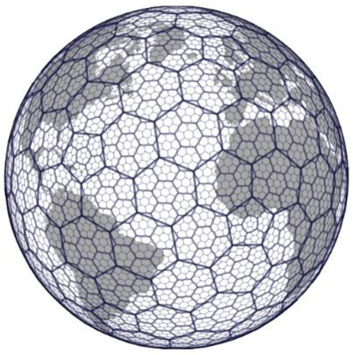


Figure 68: H3 enables users to partition the globe into hexagons for more accurate analysis.

The reason we selected it, is because the GCM calculations are not for a point, but for an area. When other layers are to be combined, we will need a way to always match the same areas together. For example, if a partner calculates "average temperature" or another model outputs "heat stress" metrics for a given area, we need a way to match it against specific regions for which we have GCM calculations.

"H3 is hierarchical: Indexed data can be quickly joined across disparate datasets and aggregated at different levels of precision"

source: https://h3geo.org/

An H3 cell index can be converted back to a coordinate which represents the center of the hex.

Anyone generating H3 cells, regardless of resolution, will have a hex grid that is "compatible" - meaning we can put our hex grid output next to a hex grid output from another system and the hexes will match.

About the model

[30] https://h3geo.org/

First, the model expects some input parameters to run for a certain city, as follows:

- Area to be calculated (format: GeoJSON: Polygon or MultiPolygon)
- POIs (format: GeoJSON)
- Weights & perfect scores per POI (format: simple mapping)
- output H3 resolution you desire (https://h3geo.org/docs/core-library/restable/)

If the H3 resolution is 12 (15 being the highest) will output 7x more hexes in comparison with 11. We've chosen for 12, which corresponds to ~300m^2 which is an acceptable area for Green Comfort Index.

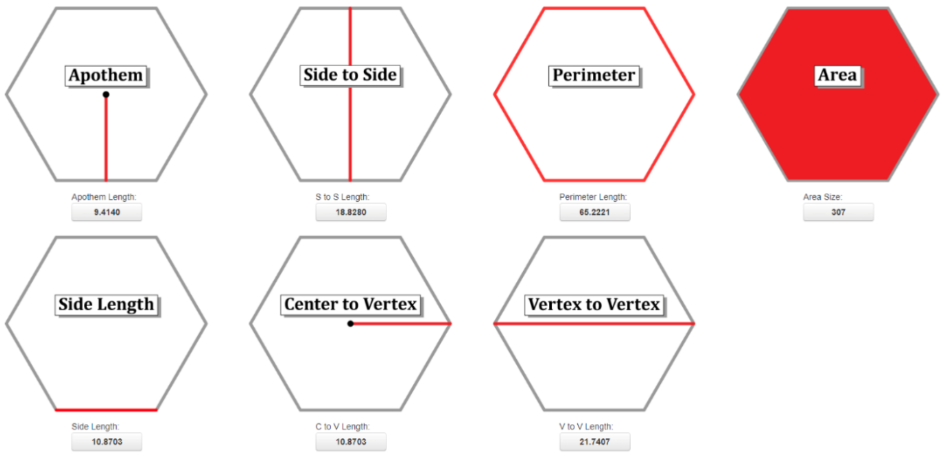


Figure 69: Hexagon geometry calculations for a resolution 12

Particularly for the POIs we used GeoJSON as the standard for the input. There is an endpoint in Geospatial API to get all the POIs for a certain area, we use that endpoint to feed the model with. The same area is also used to extract the H3 cells. Optionally, you may also supply a path to GeoTiff file as input for heat stress. This file is not processed into another format, we keep it as a GeoTIFF and based on pixel values in a single H3 cell. Then we apply the penalty score to the end score. There are 3 types of weights, which are: score (1st image), radius (2nd image) and coverage (3rd image) based.

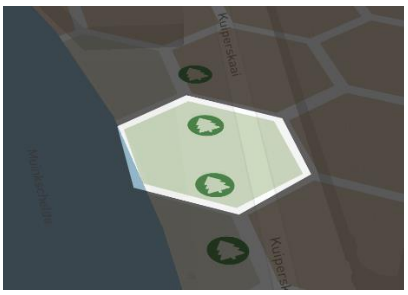
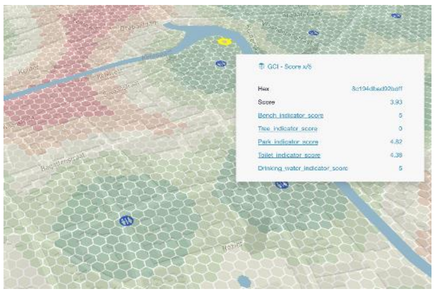
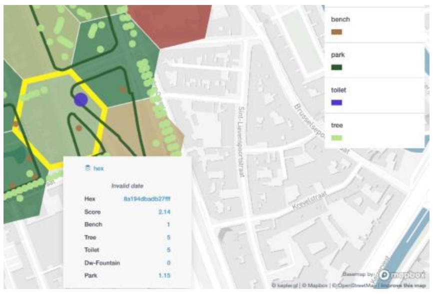


Figure 70: Score (type of weight)


GCI - Score x/5

Hex	8c194dbad02bdff
Score	3.93
Bench_indicator_score	5
Tree_indicator_score	0
Park_indicator_score	4.82
Toilet_indicator_score	4.38
Drinking_water_indicator_score	5

Figure 71: Radius (type of weight)



hex

Invalid date

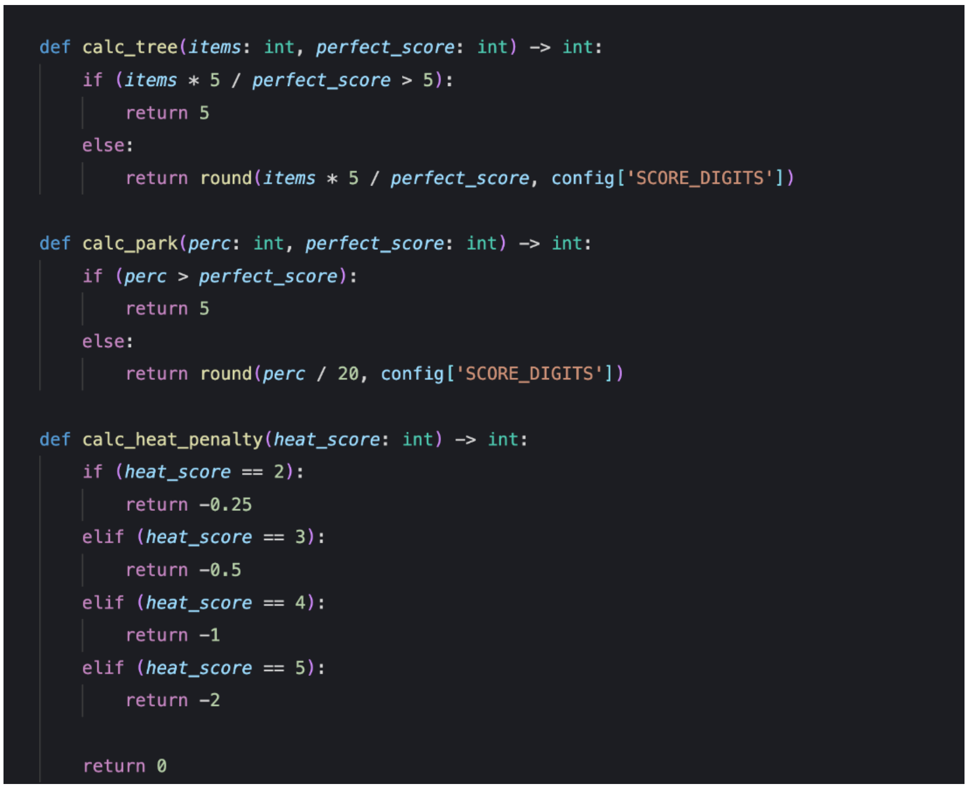
Hex	8a194dbadb27fff
Score	2.14
Bench	1
Tree	5
Toilet	5
Dw-Fountain	0
Park	1.15

Figure 72: Coverage (type of weight)

Score-based calculation is used for benches and trees, it is just summing up all the POIs inside the H3 cell. It does not consider other H3 cells. Radius-based calculation is used for toilets and drinking water fountains. For example, the perfect score being 50 meters for a H3 cell, this will result in a score of 5. If a H3 cell is 200 meters far from the toilet, this cell will result in a score of 2, because it is 4 times further than the perfect score. In the picture you can clearly see the steps of the radius calculation. The coverage-based calculation is used for parks, this will basically get the percentage of park coverage within a H3 cell.

Secondly, the area will be converted into H3 cells covering the entire area. If a Multi-polygon is provided with inner holes, these will also be ignored when converting to H3. This technique was applied to Ghent's use case, omitting all the private spaces.

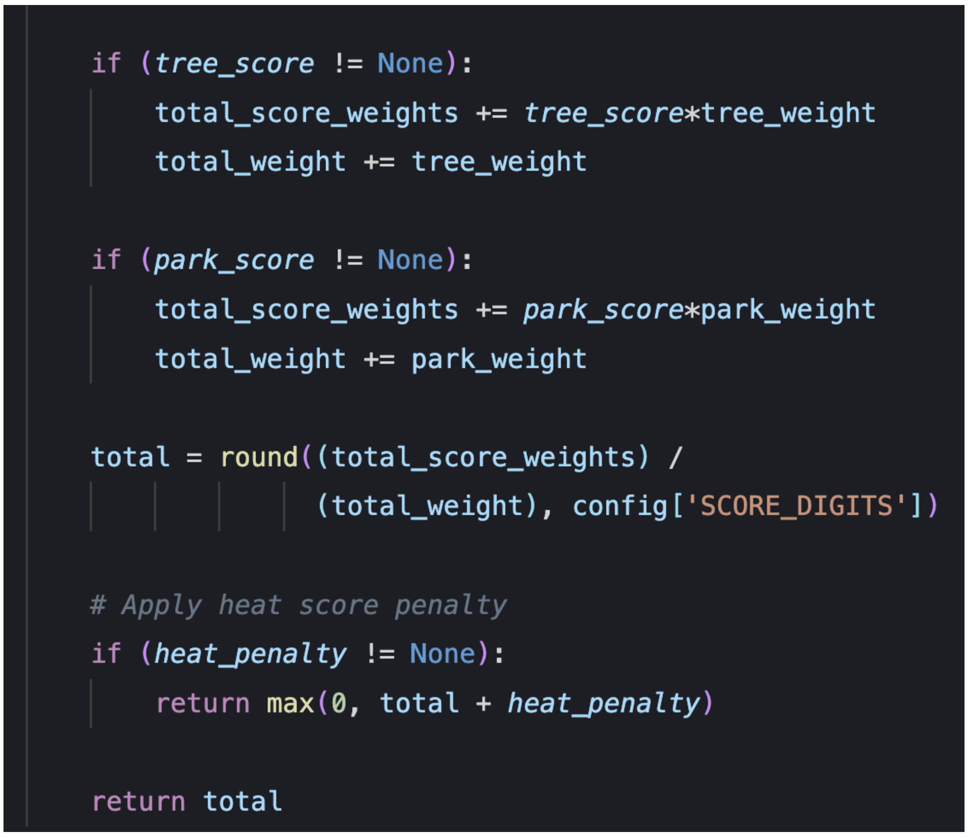
The model then starts the actual calculation, looping through each POI for every H3 cell and assigning indicator scores with the help of perfect scores per H3 cell.



```python
def calc_tree(items: int, perfect_score: int) -> int:
    if (items * 5 / perfect_score > 5):
        return 5
    else:
        return round(items * 5 / perfect_score, config['SCORE_DIGITS'])

def calc_park(perc: int, perfect_score: int) -> int:
    if (perc > perfect_score):
        return 5
    else:
        return round(perc / 20, config['SCORE_DIGITS'])

def calc_heat_penalty(heat_score: int) -> int:
    if (heat_score == 2):
        return -0.25
    elif (heat_score == 3):
        return -0.5
    elif (heat_score == 4):
        return -1
    elif (heat_score == 5):
        return -2

    return 0
```

Figure 73: Code extract - Calculating indicator scores

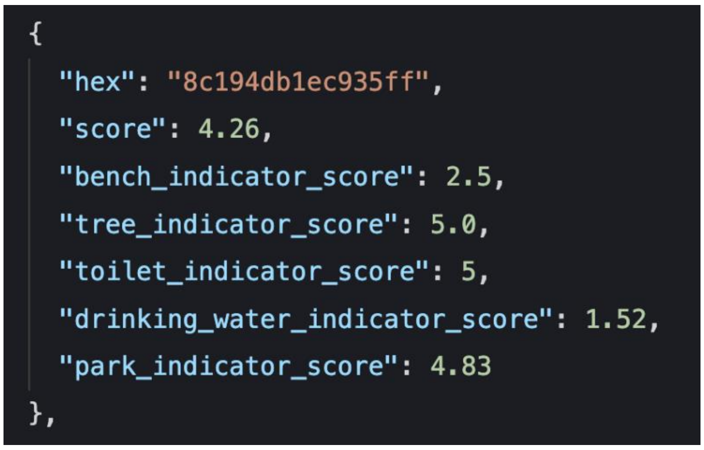
Last but not least, the green comfort score is determined based on the indication scores along with weights for each H3 cell.



```python
if (tree_score != None):
    total_score_weights += tree_score*tree_weight
    total_weight += tree_weight

if (park_score != None):
    total_score_weights += park_score*park_weight
    total_weight += park_weight

total = round((total_score_weights) /
              (total_weight), config['SCORE_DIGITS'])

# Apply heat score penalty
if (heat_penalty != None):
    return max(0, total + heat_penalty)

return total
```

Figure 74: Code extract - Calculating final GCI score

The H3 Index, the actual green comfort score, and all of the indicator scores for each POI are included in the output, which is a simple flat JSON file.



```json
{
  "hex": "8c194db1ec935ff",
  "score": 4.26,
  "bench_indicator_score": 2.5,
  "tree_indicator_score": 5.0,
  "toilet_indicator_score": 5,
  "drinking_water_indicator_score": 1.52,
  "park_indicator_score": 4.83
},
```

Figure 75: GCI output example for a single hex

4.4.2.2 Data used

When generating the GCI baselines for the cities seen in this document, we used mainly OSM data for the city amenities and a VMM example output of the new Heat Score. The GCI model itself, however, supports any data-source, as long as the data is provided according to the model requirements described in the sections above. We can demonstrate for example the use of local data-sources containing city infrastructure managed by the city administrations themselves, or the output of a street image object detection model such as our own Street Furniture model described in this very document.

4.4.2.2.1 OSM

In the case of GCI, we made use of OSM to kickstart GCI calculations, by querying the necessary inputs for GCI from OSM.

Specifically, these datasets were:

4.4.2.2.1.1 Benches

In OSM, the main way to identify benches is with the tag **amenity=bench**. Additionally, we also used node tagged with **leisure=picnic_table** and any amenity with a key **bench=yes** (which indicates a feature that includes benches, e.g. a shelter with benches within).

4.4.2.2.1.2 Drinking fountains

Drinking water sources are identified with **amenity=drinking_water**

4.4.2.2.1.3 Public toilets

Toilets open to the public are identified with **amenity=toilets**

4.4.2.2.1.4 Trees

Trees are identified as nodes with tag **natural=tree**

4.4.2.2.1.5 Parks

And finally, parks were identified with **leisure=park**

4.4.2.2.2 Local open data

Many Flemish cities maintain an open data portal where certain amenities relevant to GCI can be found. These datasets can be used as inputs for a GCI model run or simulation. In case of the city of Ghent for example, we used the **publiek-sanitair-gent** dataset available as open data[31] .

4.4.2.2.3 Street Furniture model output

The output of the Street Furniture model contains the locations of GCI indicators and can be used as input for the GCI model

4.4.2.2.4 Urbanage City Information Model (CIM)

Cities with their amenities modelled in CIM, can also use that data as inputs for GCI.

[31] https://data.stad.gent/explore/dataset/publiek-sanitair-gent/information/

4.4.2.2.5 Heat score

For the heat stress indicators, we received early access to heat score maps by the Flemish Environment Agency (VMM). They have created a heat-score index, which uses a max and min WBGT (Wet Bulb Globe Temperature) to come up with a heat score between 1 and 5, where 5 represents unbearable and problematic heat stress.

For both the measured max WBGT and min WBGT during a T20 heat-day, VMM uses international thresholds, and assigns a score to both. We received access to GeoTIFF maps for all 3 values (max, min WBGT and heat score). The resolution of the GeoTIFF maps is 1 square meter.

4.4.3 Demonstration of main functionalities

In a very first phase developing the GCI, we focused on an area visited frequently by elderly and very well-known to our imec EDiT team; the area surrounding De Krook[32] in the city of Ghent. De Krook is a popular meeting spot and landmark in the very city center, the building itself containing the library, various institutions and services and reading café. There is also a pleasant waterfront promenade with lots of facilities to sit and relax.

Figure 76: De Krook front view - ©Stad Gent - Dienst Toerisme

We started by visualizing the different OSM amenities, made available via the Geospatial Analysis API, and generated different GCI outputs, exploring different weight and limit parameters for the indicators.

[32] https://visit.gent.be/en/see-do/de-krook

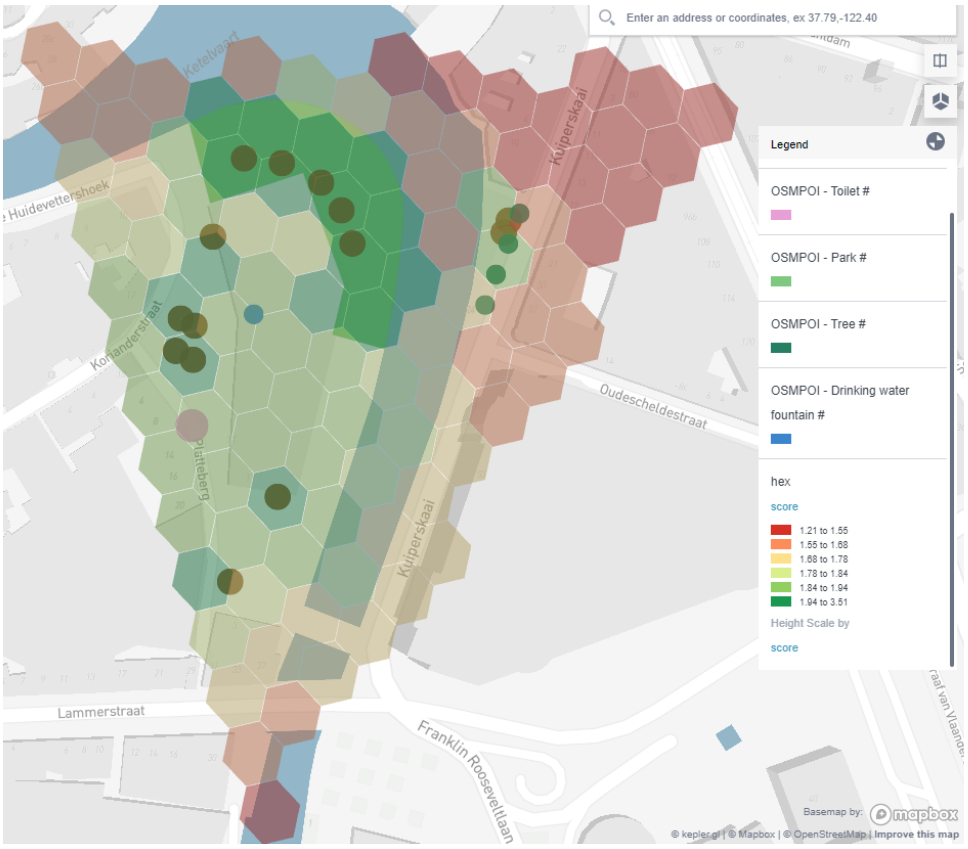
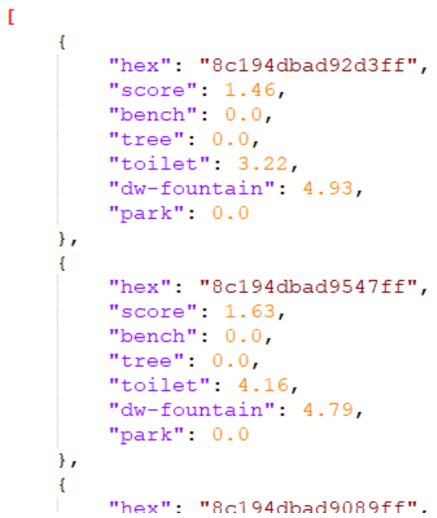


Figure 77: A first GCI output for Krook area

In order to get this first result, the model had to support the following:

- Generates an H3 grid for the requested area with resolution 12
  - Using the Krook area for quick testing
- For each hex, the model counts the number of benches and trees found inside the hex. The score for each is calculated based on those number and the limits provided (e.g. 2 benches => 5/5; 4 trees => 5/5)
- For each hex, the distance to the closest toilet and water fountain is calculated. The closer it is, the higher its score.
  - For each hex, the distance from the hex center point to each POI (toilet or water fountain) is calculated. If it is higher than the limit * 6, it is ignored (because it would result in a score of 0. If it is the lowest distance encountered so far, it is withheld. If it is lower or equal to the limit, the loop stops and results in a maximum score of 5 (because we already encountered a POI close enough to get 5/5). Otherwise, the loop continues until all distances have been calculated and the lowest has been found.
- For each hex, the area of the hex that overlaps with park areas is calculated. If a hex has overlapping areas with more than 1 park, the total is used. The percentage of the hex area that overlaps with park(s) is used to calculate the score.

- All indicator scores and their weights are used to calculate the final score.
- The model generates a flat file with the hex id, the green comfort score and the individual indicator scores



```
[
    {
        "hex": "8c194dbad92d3ff",
        "score": 1.46,
        "bench": 0.0,
        "tree": 0.0,
        "toilet": 3.22,
        "dw-fountain": 4.93,
        "park": 0.0
    },
    {
        "hex": "8c194dbad9547ff",
        "score": 1.63,
        "bench": 0.0,
        "tree": 0.0,
        "toilet": 4.16,
        "dw-fountain": 4.79,
        "park": 0.0
    },
    {
        "hex": "8c194dbad9089ff",
```

Figure 78: GCI output fragment

Pretty soon afterwards we also created GCI results using the Gent Low Emission Zone[33] for user validation. With a H3 resolution of 12 the entirety of Ghent would take more than 700,000 hexes, which the model performance could not handle at the moment. The LEZ needs around 30,000 hexes, which is doable.

[33] https://stad.gent/nl/mobiliteit-openbare-werken/lage-emissiezone/waar-ligt-de-lage-emissiezone-precies

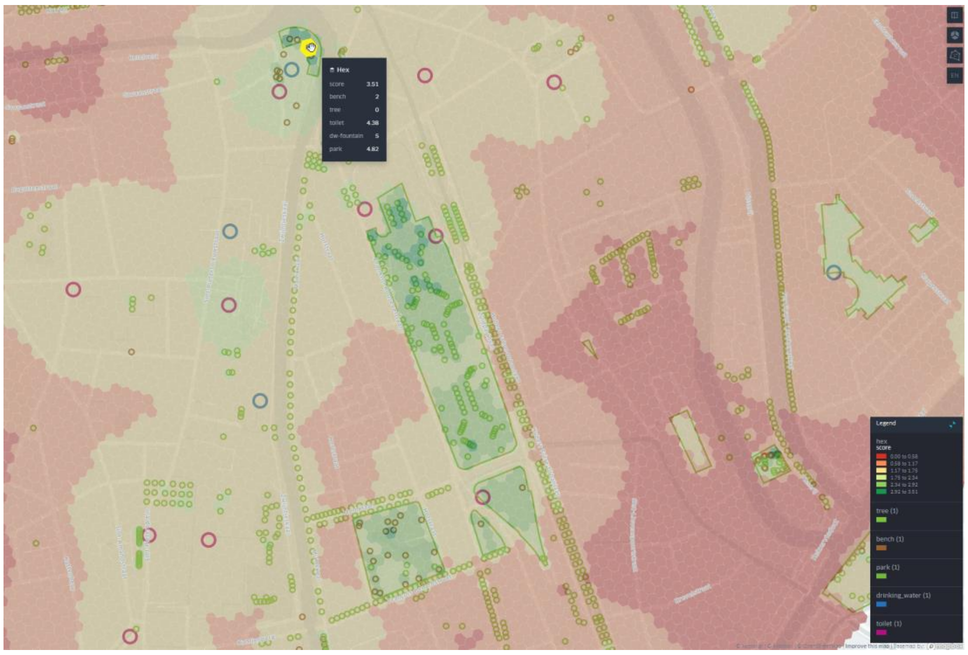


Figure 79: GCI results for Ghent LEZ

To understand how the different variables in the model work and can be tweaked, we defined these different calculation "types":

Calculation type	Indicator examples	Description	Use of limit (perfect score)
Count	Trees, benches	The number of POIs present in the cell determines the score.	If the count exceeds the limit, the score is set to 5. Any count below the limit results in a score between 0 and 5.
Nearest	Public toilets, drinking water fountains	The distance to the nearest POI (calculating from the cell center point) determines the score.	If the nearest is equal to or lower than the limit, the score is set to 5. If the nearest is greater than 6*limit, the score is set to 0. Any nearest between those values results in a score between 0 and 5.
Coverage	Parks	The percentage of cell area covered by the POI area determines the score.	If the coverage exceeds the limit, the score is set to 5. Any coverage below the limit results in a score between 0 and 5.

Table 11: Model variables in the model

At this point, we are able to generate GCI model outputs for different Flemish cities. An example for the City of Leuven.

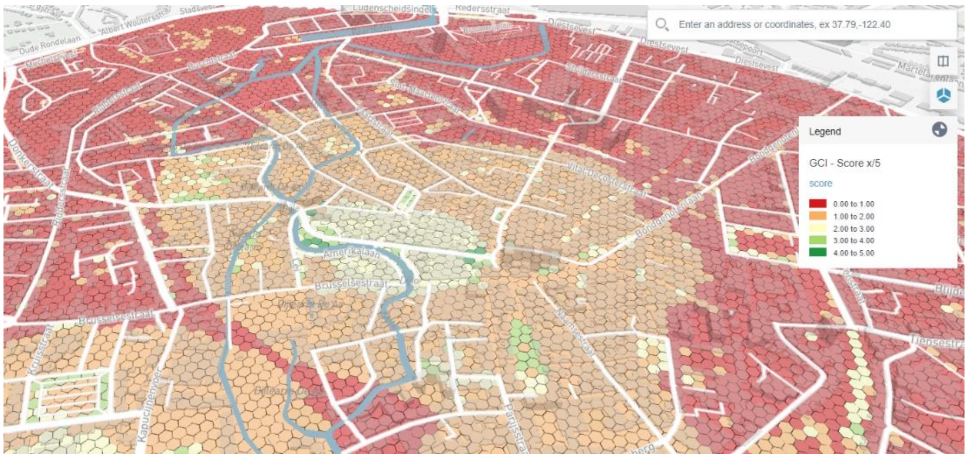


Figure 80: GCI results for Leuven

Thanks to the polygon operations added in the Geospatial Analysis API (described in this document below in paragraph 4.5) we were able to eliminate non-public spaces from the GCI calculations, making the results even more recognizable and useful for validation.

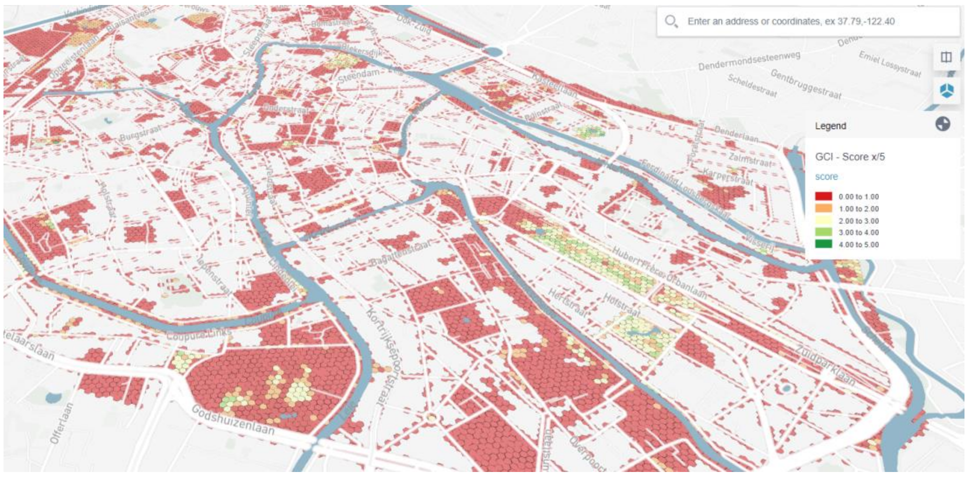


Figure 81: GCI results for public spaces in the Ghent LEZ

When we extended the GCI model with the heat stress indicators, we can also introduce a new calculation type for the model to handle:

Calculation type	Indicator examples	Description	Penalty
Highest_Most_Common	Heat Score	The value with most occurrences in the area. In case of a tie,	Each contributor value corresponds with a penalty - a value that gets deducted from the Green Comfort score result. In

		the highest value wins	case of heat score, a high value causes the highest penalty

Table 12: Model calculation types

As mentioned before, high heat stress result in a penalty, meaning the Green Comfort score will be reduced by a certain number. To figure out the heat score penalty for a hex, we look for the heat score that is most common inside the hex. In case of a tie, the worst heat score (i.e. the highest) wins.

To visualize the impact of the heat score on the new Green Comfort score, we can add both the heat score layer and the GC scores on a map of Ghent:

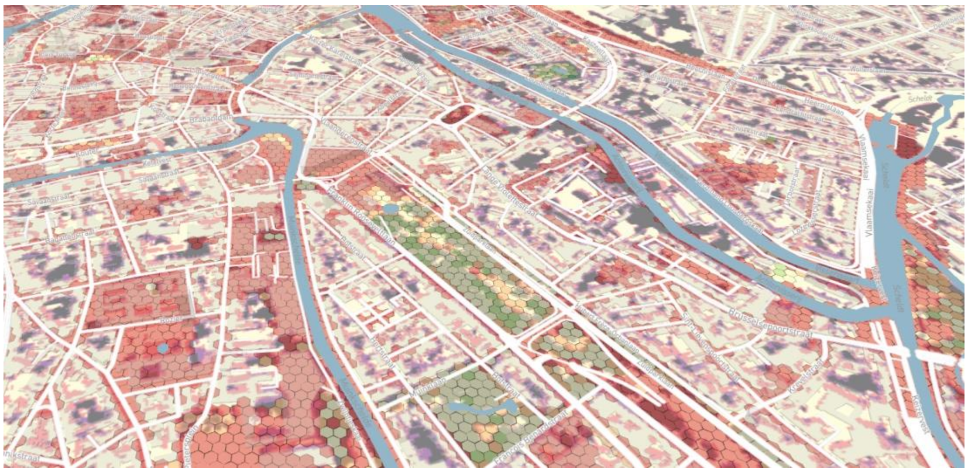
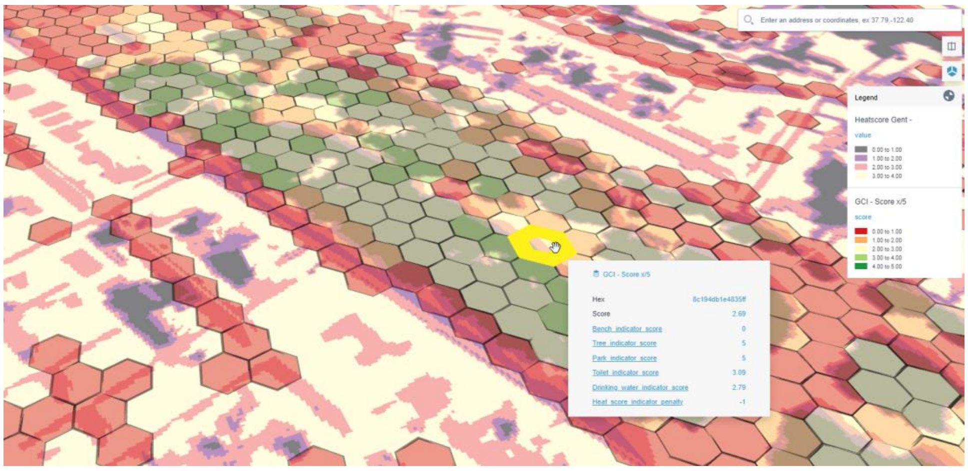


Figure 82: A visualization of both GCI (hexes) and heat score in area around Albert I park

Looking at the Albert I park, most areas have low heat scores (dark blue, purple) and are not affected by a penalty to their GC score. However, some areas (e.g. the outside borders of the park neighboring a wide concrete street) have high heat scores, and their GC score gets penalized, resulting in a lower score then expected when looking at presence of trees, benches…



GCI - Score x/5

Hex	8c194db1e4835ff
Score	2.69
Bench_indicator_score	0
Tree_indicator_score	5
Park_indicator_score	5
Toilet_indicator_score	3.09
Drinking_water_indicator_score	2.79
Heat_score_indicator_penalty	-1

Figure 83: GCI and heat score details for a hex on the edge of the Albert I park

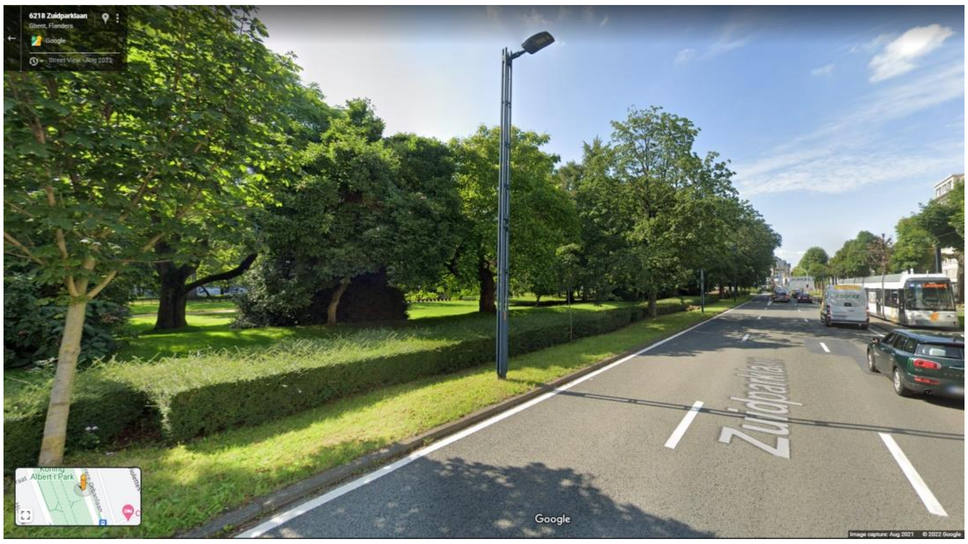


Figure 84: A street view image of the area at the edge of the Albert I park

Using the results of the co-design workshop and our own knowledge of the environment, we can tweak the parameters and create different versions of the GCI outputs to support validation activities.

Different GCI versions can be achieved by:

- Using different resolutions: changing the hex resolutions will result in different GCI scores. We decided against introducing this variability for now, because the current surface area of approximately 300 m^2 is well received.
- Creating outputs with different indicator limits: changing how many benches is considered perfection, what's the ideal distance to a public restroom...

- Creating outputs with different indicator weights: changing how important the presence of benches is for the final GCI score, how important is greenery.
- Leaving out certain indicators: basically, setting the weight of the indicator to zero

Because the GCI model takes these limit and weight parameters as inputs, the client requesting a GCI run already has some flexibility available. For the validation workshops we created 2 pre-defined parameters sets, each with a different priority:

Calculation type	Indicator examples	Description
Bench	2	1
Drinking water	100 m	0.2
Park	100 %	0.5
Toilet	50 m	0.4
Tree	2	0.5

Table 13: Set 1 – Priority on sitting comfortability

Calculation type	Indicator examples	Description
Bench	2	0.6
Tree	2	1
Toilet	50 m	0.4
Drinking water	100 m	0.2
Park	100 %	1

Table 14: Set 2 –Priority on green environment

Both sets have been used to create GCI results for different cities and for use during validation workshops. For the heat stress penalties, the following table is used:

Hex Heat Score	Heat indicator penalty
1 (best)	0
2	-0.25
3	-0.5
4	-1
5 (worst)	-2

Table 15: Heat stress penalties

4.4.4 Delivery and usage

The user should prepare and submit the input dataset to GCM model, then request GCM calculation to start. Delivery and usage info can also be found on public page[34].

4.4.4.1 Prepare dataset(s) - inputs

POST Request to /dataset endpoint to set your datasets
Below is an example for drinking water dataset.

```
{
  "type": "FeatureCollection",
  "features": [
    {
      "type": "Feature",
      "id": "6100950764",
      "geometry": {
        "type": "Point",
        "coordinates": [3.7405577, 51.0463066992415]
      },
      "properties": {
        "distance": null,
        "type": "ur:drinking water"
      }
    },
    {
      "type": "Feature",
      "id": "6544184928",
      "geometry": {
        "type": "Point",
        "coordinates": [3.7199965000000006, 51.0545479992401]
      },
      "properties": {
        "distance": null,
        "type": "ur:drinking water"
      }
    }
  ]
}
```

4.4.4.2 Start model - request

POST Request to /green-comfort-score endpoint to request GCM calculation.

```
{
  "calculationArea": "https://example.org/calculation-area.json",
  "bench": "https://example.org/bench.json",
  "drinkingWater": "https://example.org/drinking-water.json",
  "park": "https://example.org/drinking-park.json",
  "toilet": "https://example.org/toilet.json",
  "tree": "https://example.org/tree.json"
```

[34] https://artifact-hub.odt.imec-apt.be/packages/container/ur-api-rest-gcm/ur-api-rest-gcm

```
}
```

Once the calculation is completed the result, sample below, will be published to the topic which the user can receive the message and retrieve the output from dataset endpoint with the given reference

```
{
  "id": "641a16df-acfd-4405-a2bd-18a0dbc2a113",
  "calculationArea": "https://example.org/calculation-area.json",
  "bench": "https://example.org/bench.json",
  "drinkingWater": "https://example.org/drinking-water.json",
  "park": "https://example.org/drinking-park.json",
  "toilet": "https://example.org/toilet.json",
  "tree": "https://example.org/tree.json",
  "status": "started",
  "createdOn": "2022-11-09T11:01:45.172861139Z"
}
```

4.4.4.3 Model report - output

Request the dataset from the URL which is in the output field of the GCM calculation result. Below is an example GCM result output.

```
[
  {
    "hex": "8c194db132b05ff",
    "score": 0.87,
    "bench-indicator-score": 0.0,
    "tree-indicator-score": 0.0,
    "toilet-indicator-score": 2.41,
    "drinking-water-indicator-score": 4.21,
    "park-indicator-score": 0.0
  },
  {
    "hex": "8c194dbad1055ff",
    "score": 0.13,
    "bench-indicator-score": 0.0,
    "tree-indicator-score": 0.0,
    "toilet-indicator-score": 0.44,
    "drinking-water-indicator-score": 0.52,
    "park-indicator-score": 0.0
  },
  {
    "hex": "8c194db1354b3ff",
    "score": 0.75,
    "bench-indicator-score": 0.0,
    "tree-indicator-score": 0.0,
    "toilet-indicator-score": 3.99,
    "drinking-water-indicator-score": 0.8,
    "park-indicator-score": 0.0
  }
]
```

5 Conclusion

This document has been devoted to present the main advances performed up to M24 on the task T3.2 of URBANAGE - *AI Algorithms and Simulation*, part of the WP3 – *Data & Intelligence*. The main objective of the manuscript has been the deep description of all the AI and Data Science related components implemented along the project, which will be deployed in the different pilot cases that compose the project.

Along with the functional description of each tool, we have also emphasized on spotlighting the main novelties and research contributions of each functionality. Furthermore, we have offered information about the implementation aspects of the tools, also demonstrating their principal features. It should be highlighted that, despite each component has been tested on a single pilot case, we have put a special attention on stressing the replicability of each developed functionality.

The main task that involved the work described in this deliverable, which is T3.2, finishes its activity in month 24 of the project. In any case, this is the last version of the *AI Algorithms and Simulation* deliverable. For this reason, further advances in the context of T3.2 will be reported in the upcoming D3.7 - *Data & Intelligence framework* (delivered in M30). Further work on task T3.2 will contemplate fine-adaptations to the different uses case of URBANAGE, and fine-tunning on some specific functionalities.

6 References

[1] Liao, X., Zhou, T., Wang, X., Dai, R., Chen, X., & Zhu, X. (2022). Driver route planning method based on accident risk cost prediction. Journal of advanced transportation, 2022.

[2] Zhu, S. (2022). Multi-objective route planning problem for cycle-tourists. Transportation Letters, 14(3), 298-306.

[3] Storn, R., & Price, K. (1997). Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces. Journal of global optimization, 11(4), 341-359.

[4] Khaparde, A., Alassery, F., Kumar, A., Alotaibi, Y., Khalaf, O., Pillai, S., & Alghamdi, S. (2022). Differential evolution algorithm with hierarchical fair competition model. Intelligent Automation & Soft Computing, 33(2), 1045-1062.

[5] Abdelkader, E. M., Moselhi, O., Marzouk, M., & Zayed, T. (2022). An exponential chaotic differential evolution algorithm for optimizing bridge maintenance plans. Automation in Construction, 134, 104107.

[6] Sethanan, K., & Jamrus, T. (2020). Hybrid differential evolution algorithm and genetic operator for multi-trip vehicle routing problem with backhauls and heterogeneous fleet in the beverage logistics industry. Computers & Industrial Engineering, 146, 106571.

[7] Kaur, M., Gianey, H. K., Singh, D., & Sabharwal, M. (2019). Multi-objective differential evolution based random forest for e-health applications. Modern Physics Letters B, 33(05), 1950022.

[8] Mahmoodjanloo, M., Tavakkoli-Moghaddam, R., Baboli, A., & Bozorgi-Amiri, A. (2020). Flexible job shop scheduling problem with reconfigurable machine tools: An improved differential evolution algorithm. Applied Soft Computing, 94, 106416.

[9] Khonjun, S., Pitakaso, R., Sethanan, K., Nanthasamroeng, N., Pranet, K., Kaewta, C., & Sangkaphet, P. (2022). Differential Evolution Algorithm for Optimizing the Energy Usage of Vertical Transportation in an Elevator (VTE), Taking into Consideration Rush Hour Management and COVID-19 Prevention. Sustainability, 14(5), 2581.

[10] Wu, X., Liu, X., & Zhao, N. (2019). An improved differential evolution algorithm for solving a distributed assembly flexible job shop scheduling problem. Memetic Computing, 11(4), 335-355.

[11] Potthuri, S., Shankar, T., & Rajesh, A. (2018). Lifetime improvement in wireless sensor networks using hybrid differential evolution and simulated annealing (DESA). Ain Shams Engineering Journal, 9(4), 655-663.

[12] Meenachi, L., & Ramakrishnan, S. (2020). Differential evolution and ACO based global optimal feature selection with fuzzy rough set for cancer data classification. Soft Computing, 24(24), 18463-18475.

[13] Dash, J., Dam, B., & Swain, R. (2020). Design and implementation of sharp edge FIR filters using hybrid differential evolution particle swarm optimization. AEU-International Journal of Electronics and Communications, 114, 153019.

[14] Opara, K. R., & Arabas, J. (2019). Differential Evolution: A survey of theoretical analyses. Swarm and evolutionary computation, 44, 546-558.

[15] Pant, M., Zaheer, H., Garcia-Hernandez, L., & Abraham, A. (2020). Differential Evolution: A review of more than two decades of research. Engineering Applications of Artificial Intelligence, 90, 103479.

[16] Yang, X. S., & He, X. (2013). Bat algorithm: literature review and applications. International Journal of Bio-inspired computation, 5(3), 141-149.

[17] Yang, X. S., & Deb, S. (2014). Cuckoo search: recent advances and applications. Neural Computing and applications, 24(1), 169-174.

[18] Karaboga, D., Gorkemli, B., Ozturk, C., & Karaboga, N. (2014). A comprehensive survey: artificial bee colony (ABC) algorithm and applications. Artificial Intelligence Review, 42(1), 21-57.

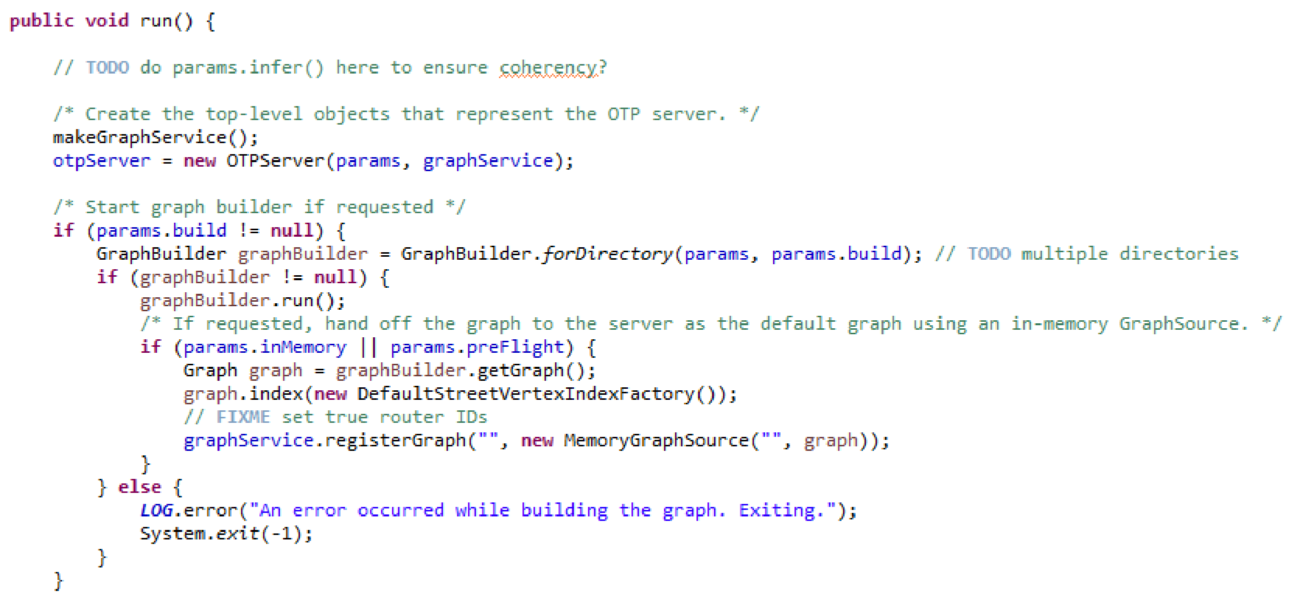
7 Appendix A

7.1 Age-friendly route planner.

7.1.1 User Manual – Running the tool

Three different alternatives are available for properly testing the Age-Friendly Routing Planner. **The first one** is acceding to the openly different accessible endpoints at: https://afrp.santander.ecosystem-urbanage.eu/, https://afrp.flanders.ecosystem-urbanage.eu/ and https://afrp.helsinki.ecosystem-urbanage.eu/. Thanks to this link, a user can consume the service using both the API, which will be detailed later, and the testing-purpose webpage. It should be considered that the version of the router deployed in this endpoint is focused on the use case of Santander.

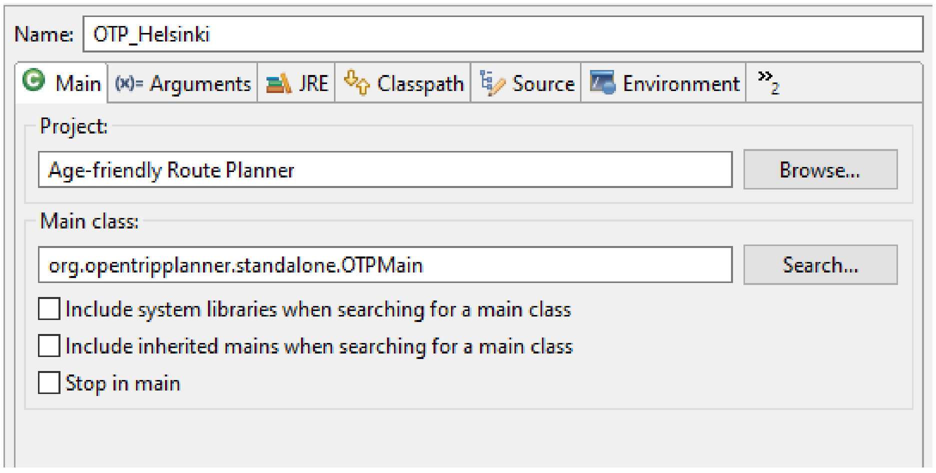
The second alternative resorts to the execution of the JAVA project using a JAVA framework and an ordinary test. The entry point of the planner if the user wants to run it in this manner is the class named as `OTPmain.java`, which is part of the package `org.opentripplanner.standalone`, within the main branch of the code. We depict in the following an excerpt of the class `OTPmain.java`. More concretely, we show the part of the code in which the planner is called through the `run()` method.



```
public void run() {

    // TODO do params.infer() here to ensure coherency?

    /* Create the top-level objects that represent the OTP server. */
    makeGraphService();
    otpServer = new OTPServer(params, graphService);

    /* Start graph builder if requested */
    if (params.build != null) {
        GraphBuilder graphBuilder = GraphBuilder.forDirectory(params, params.build); // TODO multiple directories
        if (graphBuilder != null) {
            graphBuilder.run();
            /* If requested, hand off the graph to the server as the default graph using an in-memory GraphSource. */
            if (params.inMemory || params.preFlight) {
                Graph graph = graphBuilder.getGraph();
                graph.index(new DefaultStreetVertexIndexFactory());
                // FIXME set true router IDs
                graphService.registerGraph("", new MemoryGraphSource("", graph));
            }
        } else {
            LOG.error("An error occurred while building the graph. Exiting.");
            System.exit(-1);
        }
    }
```

Figure 85: An excerpt of the OTPmain.java class

In addition to that, in order to run the tool completely adapted to a test pilot, a specific running configuration must be built. As an example, using the Eclipse IDE, this configuration can be made in Run -> Run configurations. We depict Figure 86 and Figure 87, a possible running configuration and parameterization for a Helsinki test pilot.

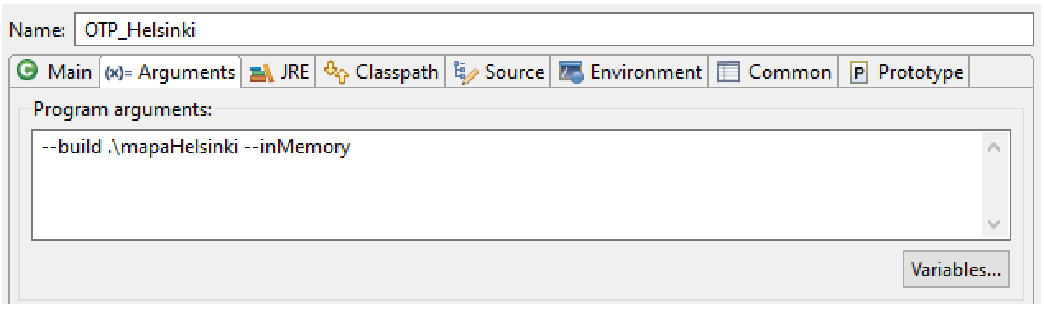

Name: OTP_Helsinki

Project: Age-friendly Route Planner

Main class: org.opentripplanner.standalone.OTPMain

☐ Include system libraries when searching for a main class
☐ Include inherited mains when searching for a main class
☐ Stop in main

Figure 86: main run configuration for a possible Helsinki Pilot

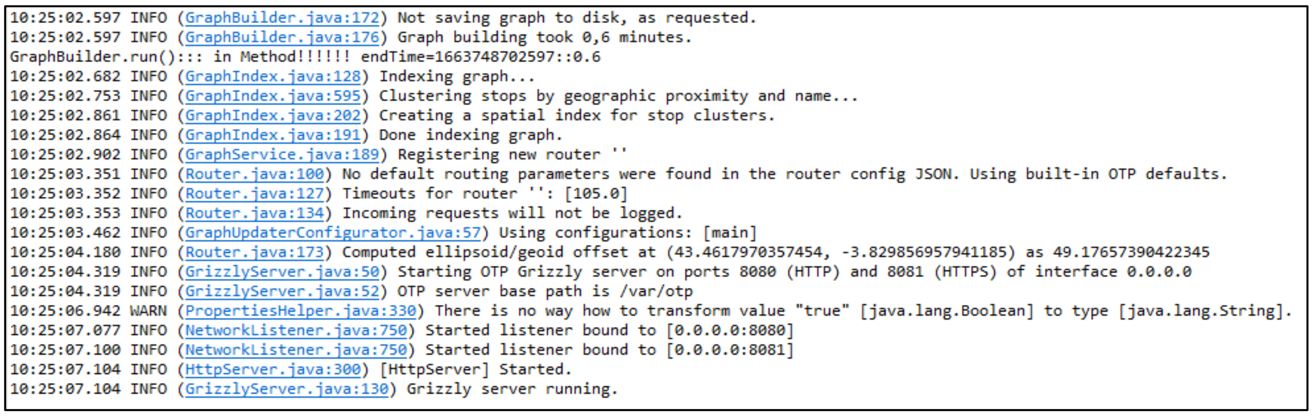

Name: OTP_Helsinki

Program arguments:

```
--build .\mapaHelsinki --inMemory
```

Figure 87: arguments configuration for a possible Helsinki Pilot

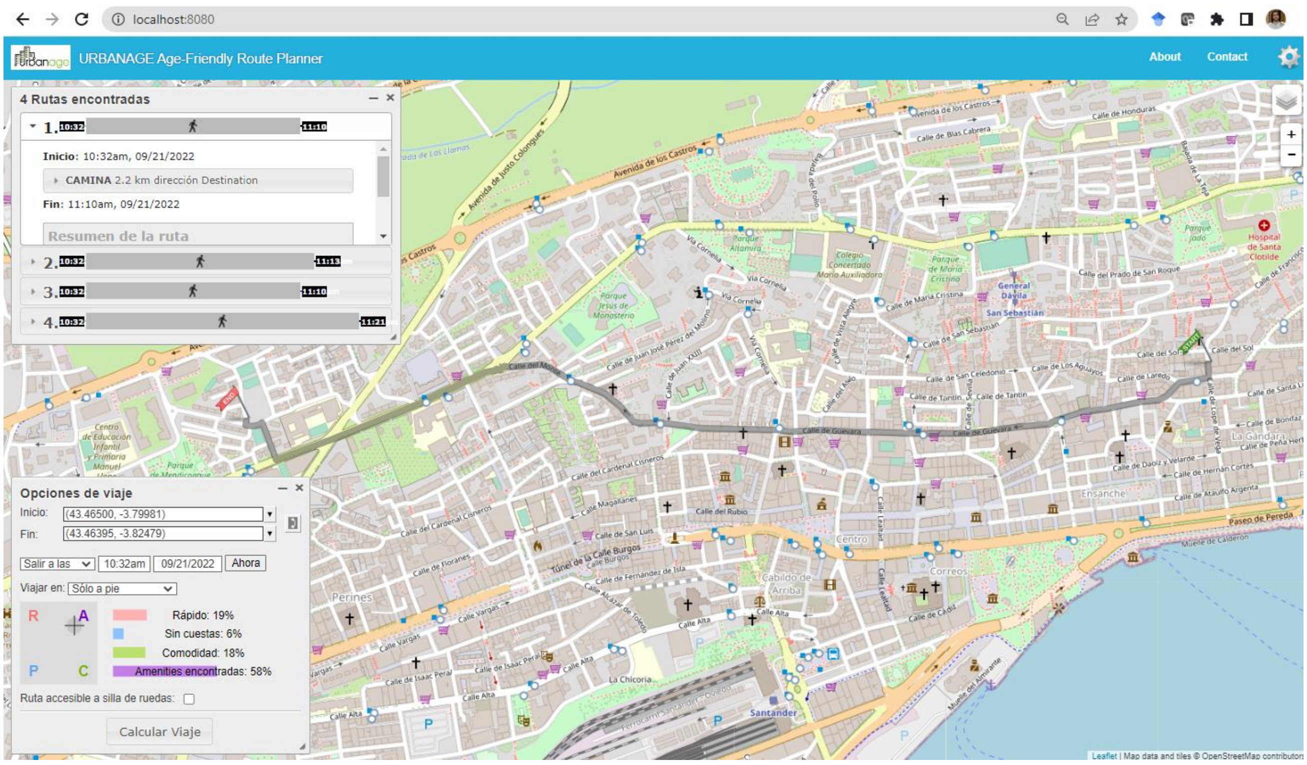
Once this run configuration is introduced, the service can be run, and it is deployed locally. This deployment can take some minutes, depending on the amount and size of data that the planner should consider. In other words, depending on the size of inputs such as the OSM map, the GTFS, the Geotiff or the amenities file. Once the Age-Friendly Routing Planner is deployed, the message `Grizzly server running` is returned to the user, as depicted in Figure 88.



```
10:25:02.597 INFO (GraphBuilder.java:172) Not saving graph to disk, as requested.
10:25:02.597 INFO (GraphBuilder.java:176) Graph building took 0,6 minutes.
GraphBuilder.run()::: in Method!!!!!! endTime=1663748702597::0.6
10:25:02.682 INFO (GraphIndex.java:128) Indexing graph...
10:25:02.753 INFO (GraphIndex.java:595) Clustering stops by geographic proximity and name...
10:25:02.861 INFO (GraphIndex.java:202) Creating a spatial index for stop clusters.
10:25:02.864 INFO (GraphIndex.java:191) Done indexing graph.
10:25:02.902 INFO (GraphService.java:189) Registering new router ''
10:25:03.351 INFO (Router.java:100) No default routing parameters were found in the router config JSON. Using built-in OTP defaults.
10:25:03.352 INFO (Router.java:127) Timeouts for router '': [105.0]
10:25:03.353 INFO (Router.java:134) Incoming requests will not be logged.
10:25:03.462 INFO (GraphUpdaterConfigurator.java:57) Using configurations: [main]
10:25:04.180 INFO (Router.java:173) Computed ellipsoid/geoid offset at (43.4617970357454, -3.829856957941185) as 49.17657390422345
10:25:04.319 INFO (GrizzlyServer.java:50) Starting OTP Grizzly server on ports 8080 (HTTP) and 8081 (HTTPS) of interface 0.0.0.0
10:25:04.319 INFO (GrizzlyServer.java:52) OTP server base path is /var/otp
10:25:06.942 WARN (PropertiesHelper.java:330) There is no way how to transform value "true" [java.lang.Boolean] to type [java.lang.String].
10:25:07.077 INFO (NetworkListener.java:750) Started listener bound to [0.0.0.0:8080]
10:25:07.100 INFO (NetworkListener.java:750) Started listener bound to [0.0.0.0:8081]
10:25:07.104 INFO (HttpServer.java:300) [HttpServer] Started.
10:25:07.104 INFO (GrizzlyServer.java:130) Grizzly server running.
```

Figure 88: message returned by the JAVA framework once the planner is deployed.

After this first deployment step, the user can access the Age-Friendly route planner both on its test-purpose webpage through the just mentioned endpoints (as depicted in Figure 89) or using an API call (as shown in 7.1.2).



4 Rutas encontradas

1. 10:32 — 11:10

Inicio: 10:32am, 09/21/2022

CAMINA 2.2 km dirección Destination

Fin: 11:10am, 09/21/2022

Resumen de la ruta

2. 10:32 — 11:13

3. 10:32 — 11:10

4. 10:32 — 11:21

Opciones de viaje

Inicio: (43.46500, -3.79981)

Fin: (43.46395, -3.82479)

Salir a las 10:32am 09/21/2022 Ahora

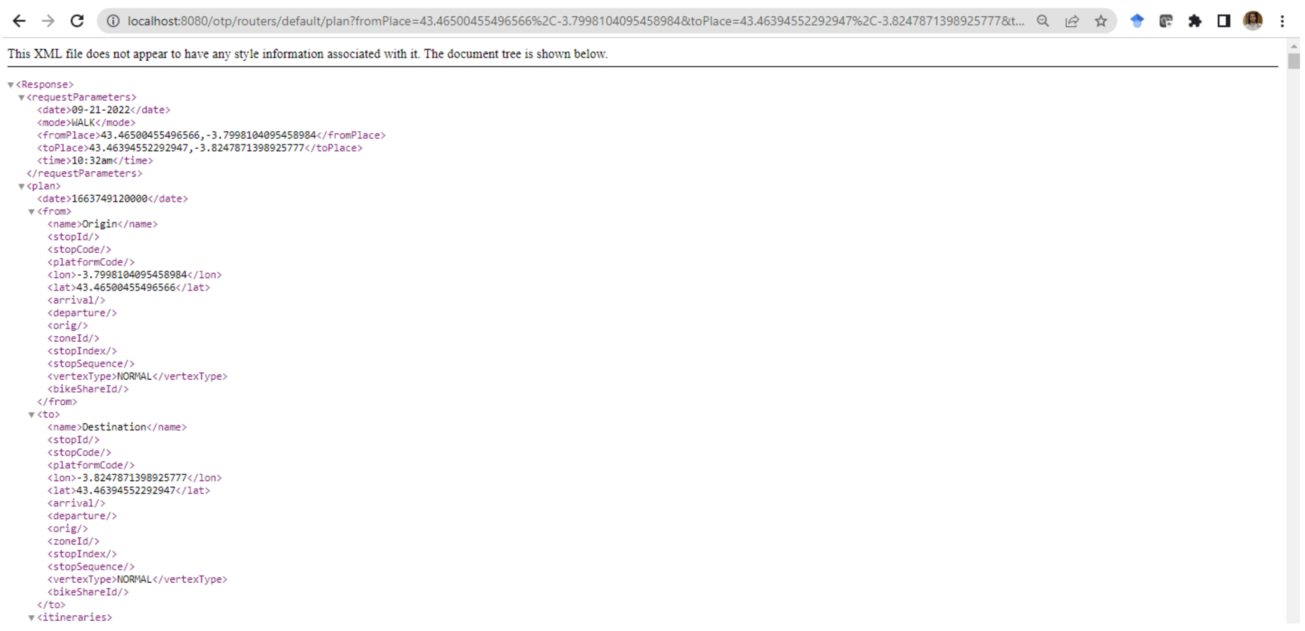
Viajar en: Sólo a pie

🟥	Rápido: 19%
🟦	Sin cuestas: 6%
🟩	Comodidad: 18%
🟪	Amenities encontradas: 58%

Ruta accesible a silla de ruedas: ☐

Calcular Viaje

Figure 89: Accessing the route planner locally though the test-purpose webpage.



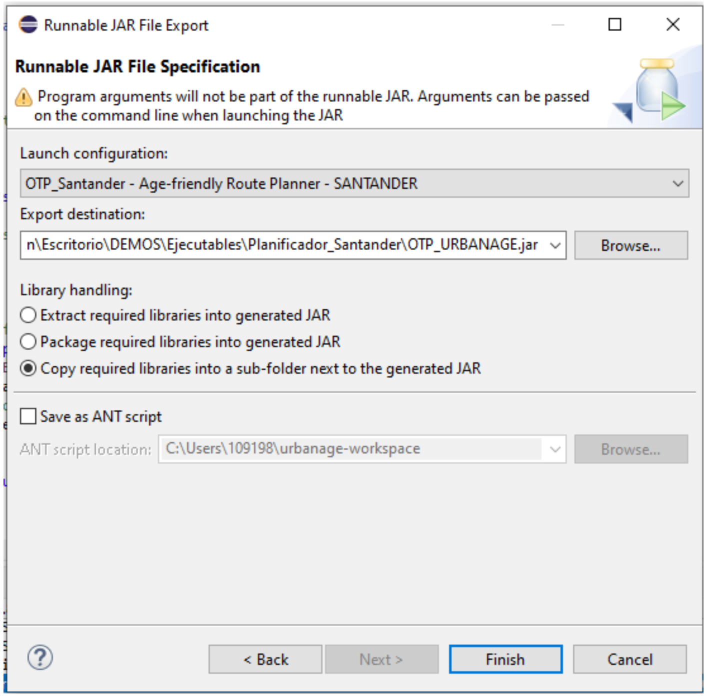
localhost:8080/otp/routers/default/plan?fromPlace=43.46500455496566%2C-3.7998104095458984&toPlace=43.46394552292947%2C-3.8247871398925777&t...

This XML file does not appear to have any style information associated with it. The document tree is shown below.

```
▼<Response>
  ▼<requestParameters>
     <date>09-21-2022</date>
     <mode>WALK</mode>
     <fromPlace>43.46500455496566,-3.7998104095458984</fromPlace>
     <toPlace>43.46394552292947,-3.8247871398925777</toPlace>
     <time>10:32am</time>
   </requestParameters>
  ▼<plan>
     <date>1663749120000</date>
    ▼<from>
       <name>Origin</name>
       <stopId/>
       <stopCode/>
       <platformCode/>
       <lon>-3.7998104095458984</lon>
       <lat>43.46500455496566</lat>
       <arrival/>
       <departure/>
       <orig/>
       <zoneId/>
       <stopIndex/>
       <stopSequence/>
       <vertexType>NORMAL</vertexType>
       <bikeShareId/>
     </from>
    ▼<to>
       <name>Destination</name>
       <stopId/>
       <stopCode/>
       <platformCode/>
       <lon>-3.8247871398925777</lon>
       <lat>43.46394552292947</lat>
       <arrival/>
       <departure/>
       <orig/>
       <zoneId/>
       <stopIndex/>
       <stopSequence/>
       <vertexType>NORMAL</vertexType>
       <bikeShareId/>
     </to>
    ▼<itineraries>
```

Figure 90: Accessing the route planner locally using the API

Finally, the **third alternative is related to locally running a JAVA JAR file**, created though the exportation of the project as a Runnable JAR File. We depict in Figure X how this exportation can be done using the Eclipse IDE.



Runnable JAR File Export

Runnable JAR File Specification

Program arguments will not be part of the runnable JAR. Arguments can be passed on the command line when launching the JAR

Launch configuration: OTP_Santander - Age-friendly Route Planner - SANTANDER

Export destination: n\Escritorio\DEMOS\Ejecutables\Planificador_Santander\OTP_URBANAGE.jar

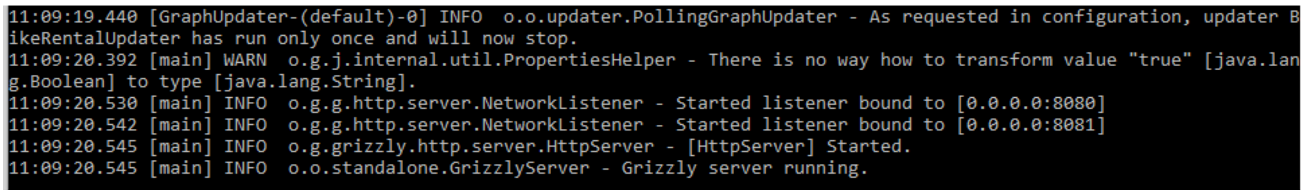
Library handling:
- ○ Extract required libraries into generated JAR
- ○ Package required libraries into generated JAR
- ● Copy required libraries into a sub-folder next to the generated JAR

☐ Save as ANT script

ANT script location: C:\Users\109198\urbanage-workspace

Figure 91: exportation of the project as a Runnable JAVA File for the Santander test case.

Once the JAR is exported and placed in a folder along with the data related to the test pilot (the OSM map, the GTFS, the amenities file…), the service can be deployed using the following command (for the Santander use case): java -jar OTP_URBANAGE.jar --build .\mapaSantander –-inMemory. After that, and when the message INFO (GrizzlyServer.java:130) Grizzly server running is returned (as depicted in Figure 92: Server running using the JAR file.Figure 92), the planner can be accessed using the same protocol as explained for the second alternative.



```
11:09:19.440 [GraphUpdater-(default)-0] INFO  o.o.updater.PollingGraphUpdater - As requested in configuration, updater B
ikeRentalUpdater has run only once and will now stop.
11:09:20.392 [main] WARN  o.g.j.internal.util.PropertiesHelper - There is no way how to transform value "true" [java.lan
g.Boolean] to type [java.lang.String].
11:09:20.530 [main] INFO  o.g.g.http.server.NetworkListener - Started listener bound to [0.0.0.0:8080]
11:09:20.542 [main] INFO  o.g.g.http.server.NetworkListener - Started listener bound to [0.0.0.0:8081]
11:09:20.545 [main] INFO  o.g.grizzly.http.server.HttpServer - [HttpServer] Started.
11:09:20.545 [main] INFO  o.o.standalone.GrizzlyServer - Grizzly server running.
```

Figure 92: Server running using the JAR file.

7.1.2 User Manual – Using the tool though the API

This section aims to describe the REST API design to interact with the Age-Friendly route planner. In a nutshell, and considering a tentative pilot case in Helsinki, the route planning system can be called using the following general API:

```
https://afrp.helsinki.urbanage.digital.tecnalia.dev/otp/routers/default/
plan?fromPlace=X&toPlace=X&time=X&date=X&mode=X
```

As can be seen, the first part of the call is related with the specific request of the service. On the other hand, the last part of the call is dedicated to the set of parameters, each separated by an ampersand. Additionally, this call has been made locally. More concretely, a specific call for a specific route could be the following one:

```
https://afrp.helsinki.urbanage.digital.tecnalia.dev/otp/routers/default/
plan?fromPlace=43.47076747496938%2C-
3.7962913513183594&toPlace=43.46782422361077%2C-
3.8077068328857417&time=12%3A20pm&date=05-04-2022&mode=WALK
&wheelchair=false
```

For this example, the input parameters are the ones represented in Table 16.

Parameter	Values
From Place	43.47077, -3.79629
To Place	43.46782, -3.80771
Time	12:20pm
Date	05-04-2022
Mode	Walk
Wheel Chair	False

Table 16: Parameter values for a specific multimodal route planning service call.

In the following table two different additional examples are displayed, each one of a different nature. The first example corresponds to a walking route, using the four routing preferences and conducted by a wheelchair. The second path corresponds to a transit route, also deeming the four routing preferences: slope, comfortability, amenities and duration.

```
http://afrp.helsinki.urbanage.digital.tecnalia.dev/otp/routers/default/p
lan?fromPlace=43.46161022230138%2C-
3.8229417800903325&toPlace=43.46463164608013%2C-
3.8022994995117183&time=12%3A20pm&date=05-04-
2022&mode=WALK&wheelchair=true&optimize=SQUARE&squareTimeFactor=0.547656
17010892&squareSlopeFactor=0.2206894686384733&squareComfortableFactor=0.
06653734376102902&squareAmenitiesFactor=0.16511701749157773
https://afrp.helsinki.urbanage.digital.tecnalia.dev/otp/routers/default/
plan?fromPlace=43.460115028911055%2C-
3.8236069679260254&toPlace=43.46618902203754%2C-
3.799059391021729&time=12%3A20pm&date=05-04-
2022&mode=TRANSIT%2CWALK&maxWalkDistance=750&wheelchair=false&optimize=S
QUARE&squareTimeFactor=0.54765617010892&squareSlopeFactor=0.220689468638
4733&squareComfortableFactor=0.06653734376102902&squareAmenitiesFactor=0
.16511701749157773
```

Figure 93: Two additional examples of API routing petitions

It should be borne in mind that *From Place* and *To Place* parameters are inserted in the system as geographical coordinates. Additionally, this API call allows the input of the parameters shown in the following **¡Error! No se encuentra el origen de la referencia.**. It should be noted that the parameters and elements colored in blue have been introduced as part of the developments made on this project, meaning that they were not contemplated in the basic version of OTP, and that they are the result of the work conducted in URBANAGE to adapt the Age-Friendly route planner to user's needs. This table is inspired by the one depicted in the official documentation of OTP35, and we have extended it accordingly.

Name	Description	Constrains
date	The date that the trip should depart (or arrive, for requests where arriveBy is true). **This parameter is compulsory.**	DD-MM-YYYY
fromPlace	The start location -- either latitude, longitude pair in degrees or a Vertex label. For example, 40.714476, -74.005966. **This parameter is compulsory.**	Lat,long
optimize	This parameter should have SQUARE value to apply square optimization. If this value is not inserted, the routes are calculated using a 25% of preference for each parameter. **This parameter is optional.** If it is not inserted, each factor has the same importance.	SQUARE

[35] http://dev.opentripplanner.org/apidoc/1.0.0

mode	The set of modes that a user is willing to use, with qualifiers stating whether vehicles should be parked, rented, etc. **This parameter is compulsory.**	"WALK", "TRANSIT"
time	The time that the trip should depart. **This parameter is compulsory.**	long
toPlace	The end location (see fromPlace for format). **This parameter is compulsory.**	Lat,long
squareAmenitiesFactor	For walk square routing, how much amenities matter (range 0-1). **This parameter is optional**	double
squareComfortabilityFactor	For walk square routing, how much comfortability matters (range 0-1). **This parameter is optional**	double
squareSlopeFactor	For walk square routing, how much slope matters (range 0-1).	double
squareTimeFactor	For walk square routing, how much time matters (range 0-1). **This parameter is optional.**	double
wheelchair	Whether the trip must be wheelchair accessible. **This parameter is optional**, and its default value is False.	boolean
enhanceRoute	A boolean parameter. If True: the AI enhanced route is provided. It False: the enhanced route is not calculated. **This parameter is optional**. The default value is True.	boolean
maxWithNoBench	A double which represents the tolerance of the user for the formulas depicted in Section 4.1.3.1.3. **This parameter is optional**. The default value is 500.	double
maxWithNoToilet	A double which represents the tolerance of the user for the formulas depicted in Section 4.1.3.1.3. **This parameter is optional**. The default value is 1000.	double
maxWithNoWater	A double which represents the tolerance of the user for the formulas depicted in Section 4.1.3.1.3. **This parameter is optional**. The default value is 700.	double
useElevators	A boolean parameter. If True, the routes contemplate elevators. It False, the routes do not use elevators. **This parameter is optional**. The default value is True.	boolean
distrElevators	A string parameter which indicated in which part of the route the elevators have more importance. **This parameter is optional**. If the parameter is not introduced, the priority is the same along the whole route. If *useElevators* is False, this parameter is not considered.	"First Half" "Second Half" "Half"
useEscalators	A boolean parameter. If True, the routes contemplate escalators. It False, the routes do not use elevators. **This parameter is optional**. The default value is True.	boolean
distrEscalators	A string parameter which indicated in which part of the route the escalators have more importance. **This parameter is optional**. If the parameter is not introduced, the priority is the same along the whole route. If *useEscalators* is False, this parameter is not considered.	"First Half" "Second Half" "Half"

useBenches	A boolean parameter. If True, the benches are considered for measuring how friendly the route is in terms of the amenities found along the path. It False, the benches have not importance in the route. **This parameter is optional**. The default value is True.	boolean
distrBenches	A string parameter which indicated in which part of the route the benches have more importance. **This parameter is optional**. If the parameter is not introduced, the priority is the same along the whole route. If *useBenches* is False, this parameter is not considered.	"First Half" "Second Half" "Half"
useToilets	A boolean parameter. If True, the toilets are considered for measuring how friendly the route is in terms of the amenities found along the path. It False, the toilets have not importance in the route. **This parameter is optional**. The default value is True.	boolean
distrToilets	A string parameter which indicated in which part of the route the toilets have more importance. **This parameter is optional**. If the parameter is not introduced, the priority is the same along the whole route. If *useToilets* is False, this parameter is not considered.	"First Half" "Second Half" "Half"
useWater	A boolean parameter. If True, the drinking water fountains are considered for measuring how friendly the route is in terms of the amenities found along the path. It False, the drinking water fountains have not importance in the route. **This parameter is optional**. The default value is True.	boolean
distrWater	A string parameter which indicated in which part of the route the drinking water fountains have more importance. **This parameter is optional**. If the parameter is not introduced, the priority is the same along the whole route. If *useWater* is False, this parameter is not considered.	"First Half" "Second Half" "Half"
locale	Language of the user	"es" "en"

Table 17: Possible parameters for the Age-Friendly Route Planning Service Call

7.1.3 User Manual – Understanding the output of the route planner

Once explained the way in which the routing services is called, and after showing some practical examples of the functionality of the deployed system, how the multimodal route planning engine returns all the generated information is described in this section. Briefly explained, the Age-Friendly route planner returns the calculated itineraries using the well-known JSON standard. The structure is based on the main output structure of the basic version of the OTP, extender with new functionalities considered in URBANAGE. In order to further understand the working way of the planner, we depict a generic call and output for a path request, adopting the structure above mentioned. The returning JSON has a significant extension, it is presented below.

API CALL

https://afrp.santander.urbanage.digital.tecnalia.dev/otp/routers/default/plan?fromPlace=X&toPlace=X&time=1X&date=X&mode=X&maxWalkDistance=X&arriveBy=X&wheelchair=X&locale=X

OUTPUT

```
"requestParameters":{ (List of the parameters introduced by the user. The ones show here are some examples)
   "date": "X", (day for the planning)
   "mode": "X", (traverse mode)
   "arriveBy": "X", (it marks if the time is representing the arrival or the departure)
   "wheelchair": "X", (it marks if the itinerary needs to be wheelchair accessible)
   "fromPlace": "X" , (origin of the route in coordinates)
   "toPlace": "X", (destination of the route in coordinates)
   "time": "X", (hour for the route)
   "maxWalkDistance": "X", (maximum walking distance in meters)
   "locale": "X"
},
"plan":{ (It contains the general information of the output)
         "date":1480610100000,
         "from":{ (information about the origin)
            "name":"X",
            "lon":X, (longitude)
            "lat":X, (latitude)
            "orig":"X",
            "vertexType":"X" (type of the vertex)
         },
         "to":{ (information about the destination)
            "name":"X",
            "lon":X, (longitude)
            "lat":X, (latitude)
            "orig":"X",
            "vertexType":"X" (type of the vertex)
         },
         "itineraries": [ (information about all the itineraries)
            { (information about a particular itineraries)
              "duration": X, (duration in seconds)
              "startTime": X, (start time in milliseconds)
              "endTime": X, (end time in milliseconds)
              "walkTime": X, (time spent in other traverse apart from transit, in seconds)
              "transitTime": X, (time spent in public transport)
              "waitingTime": X, (time spent waiting public transport services)
```

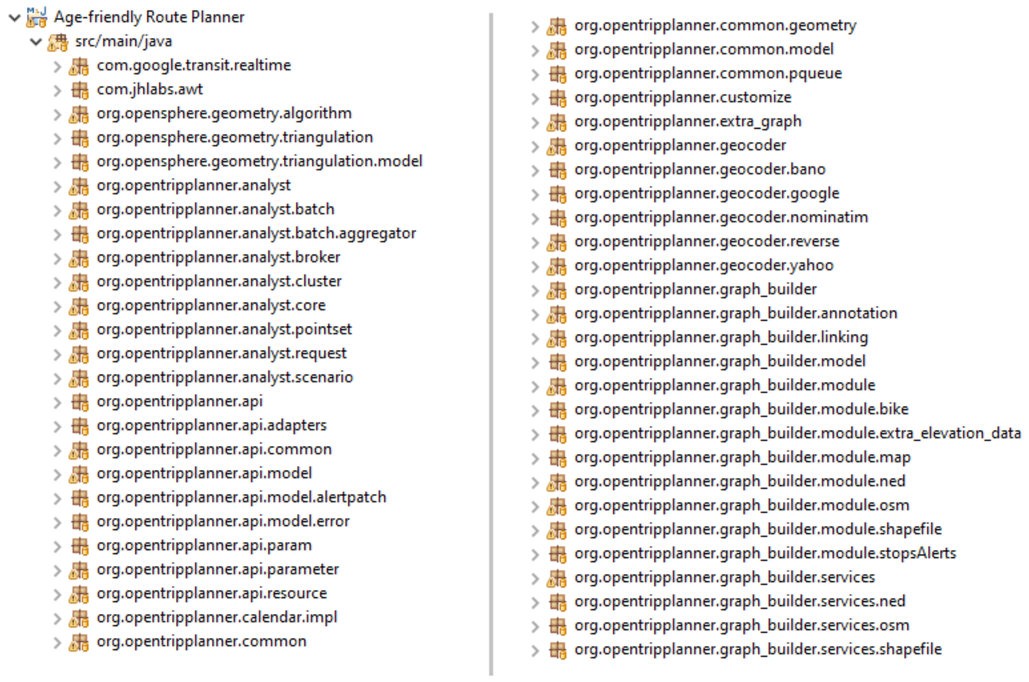
"walkDistance": X, *(distance walked)*
"walkLimitExceeded": X, *(boolean if the walking limit has been exceeded)*
"elevationLost": X, *(from the origin, the different to the less elevated point)*
"elevationGained": X, *(from the origin, the different to the most elevated point)*
"transfers": 0,
"legs": [*(information related to all the legs that compose the whole route. A leg is a part of the route made by the same transit mode)*
{ *(information about a particular leg)*
"startTime": X, *(start time in milliseconds)*
"endTime": X, *(end time in milliseconds)*
"departureDelay": X, *(delay between the fixed starting time and the real departure time)*
"arrivalDelay": X, *(delay between the time set as arrival time and the real arrival time)*
"realTime": X,
"distance": X, *(distance in meters)*
"pathway": X, *(boolean to mark if pathways are traversed)*
"mode": "X", *(traverse mode)*
"route": "",
"agencyTimeZoneOffset": X,
"interlineWithPreviousLeg": X, *(boolean marking if there is an interline with the previous line)*
"from": {
"name": "X",
"lon": X, *(longitude)*
"lat": X, *(latitude)*
"departure": X, *(departure time in milliseconds)*
"orig": "",
"vertexType": "X" *(type of the vertex)*
},
"to": {
"name": "X",
"lon": X, *(longitude)*
"lat": X, *(latitude)*
"arrival": X, *(arrival time in milliseconds)*
"orig": "X",
"vertexType": "X" *(type of the vertex)*
},
"legGeometry": {
"points": X, *(points marking the geometry)*

```
                                "length": X (length of the geometry)
                            },
                            "duration": X, (duration in seconds)
                            "transitLeg": X, (true if this leg has been made in public transport)
                            "steps": [ (information related to all the short parts that compose the whole route.)
                                {(information about a particular part)
                                  "distance": X, (distance in meters)
                                  "relativeDirection": "X", (direction followed by the user)
                                  "streetName": "X", (street name)
                                  "absoluteDirection": "X", (North, south, east or west)
                                  "stayOn": X,
                                  "area": X, (true if it is part of an area)
                                  "bogusName": X, (true if it has a bogus name)
                                  "lon": X, (longitude)
                                  "lat": X, (latitude)
                                  "elevation": [(information about the elevation of the street)
                                      {(information about the elevation at a specific point of the street)
                                        "first": X, (specific point of the segment, in meters)
                                        "second": X (elevation in meters)
                                      }, (end of a specific elevation)
                                  ] (end of the elevations set)
                          }, (end of a specific step)
                       ] (end of steps set)
                        } (end of a particular leg)
                      ], (end of a legs set)
                      "tooSloped": X,
                 }  (end of a particular itinerary)
             ]  (end of a itineraries set)
}, (end of the plan)
"debugOutput": {
   "precalculationTime": X,
   "pathCalculationTime": X, (time in milliseconds calculating all the paths)
   "pathTimes": [ (times in milliseconds for each path)
    X (times in milliseconds for a specific path)
   ],
   "renderingTime": X, (total rendering time in milliseconds)
   "totalTime": X, (total time in milliseconds)
   "timedOut": X (A boolean explaining if the timeout has reached)
 } (end of the debutOutput)
```

} *(end of the message)*

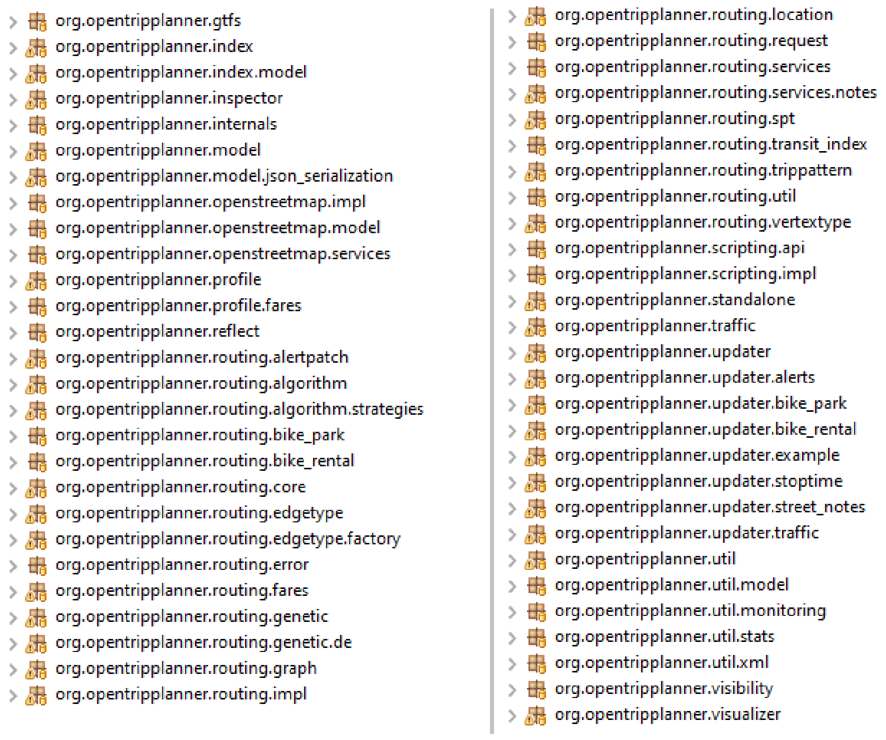
7.1.4 Detailed Package Information

The Age-Friendly route planner is based on the OTP framework, the project package has the same structure as the basic OTP, with some components added. Additionally, a significant number of classes have been modified in order to properly adapt the planner to the URBANAGE needs. Thus, the main structure of the project is composed by the packages shown in the following Figure 94 and Figure 95



```
Age-friendly Route Planner
  src/main/java
    com.google.transit.realtime
    com.jhlabs.awt
    org.opensphere.geometry.algorithm
    org.opensphere.geometry.triangulation
    org.opensphere.geometry.triangulation.model
    org.opentripplanner.analyst
    org.opentripplanner.analyst.batch
    org.opentripplanner.analyst.batch.aggregator
    org.opentripplanner.analyst.broker
    org.opentripplanner.analyst.cluster
    org.opentripplanner.analyst.core
    org.opentripplanner.analyst.pointset
    org.opentripplanner.analyst.request
    org.opentripplanner.analyst.scenario
    org.opentripplanner.api
    org.opentripplanner.api.adapters
    org.opentripplanner.api.common
    org.opentripplanner.api.model
    org.opentripplanner.api.model.alertpatch
    org.opentripplanner.api.model.error
    org.opentripplanner.api.param
    org.opentripplanner.api.parameter
    org.opentripplanner.api.resource
    org.opentripplanner.calendar.impl
    org.opentripplanner.common
    org.opentripplanner.common.geometry
    org.opentripplanner.common.model
    org.opentripplanner.common.pqueue
    org.opentripplanner.customize
    org.opentripplanner.extra_graph
    org.opentripplanner.geocoder
    org.opentripplanner.geocoder.bano
    org.opentripplanner.geocoder.google
    org.opentripplanner.geocoder.nominatim
    org.opentripplanner.geocoder.reverse
    org.opentripplanner.geocoder.yahoo
    org.opentripplanner.graph_builder
    org.opentripplanner.graph_builder.annotation
    org.opentripplanner.graph_builder.linking
    org.opentripplanner.graph_builder.model
    org.opentripplanner.graph_builder.module
    org.opentripplanner.graph_builder.module.bike
    org.opentripplanner.graph_builder.module.extra_elevation_data
    org.opentripplanner.graph_builder.module.map
    org.opentripplanner.graph_builder.module.ned
    org.opentripplanner.graph_builder.module.osm
    org.opentripplanner.graph_builder.module.shapefile
    org.opentripplanner.graph_builder.module.stopsAlerts
    org.opentripplanner.graph_builder.services
    org.opentripplanner.graph_builder.services.ned
    org.opentripplanner.graph_builder.services.osm
    org.opentripplanner.graph_builder.services.shapefile
```

Figure 94: Main structure of the developed prototype (part 1)



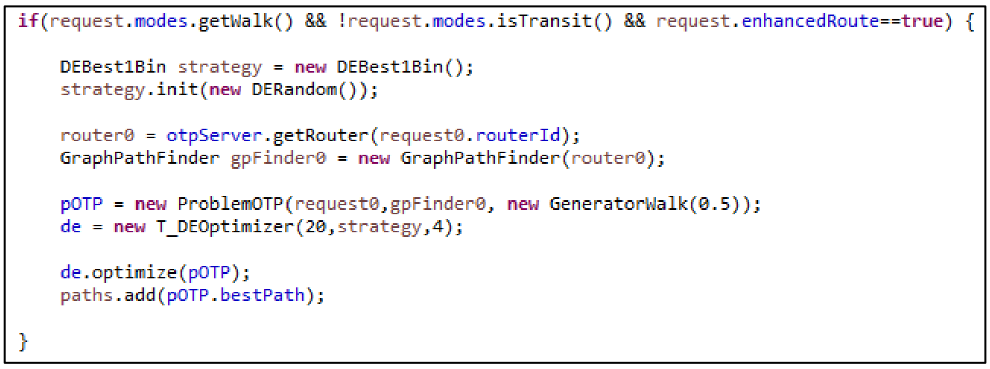
```
org.opentripplanner.gtfs
org.opentripplanner.index
org.opentripplanner.index.model
org.opentripplanner.inspector
org.opentripplanner.internals
org.opentripplanner.model
org.opentripplanner.model.json_serialization
org.opentripplanner.openstreetmap.impl
org.opentripplanner.openstreetmap.model
org.opentripplanner.openstreetmap.services
org.opentripplanner.profile
org.opentripplanner.profile.fares
org.opentripplanner.reflect
org.opentripplanner.routing.alertpatch
org.opentripplanner.routing.algorithm
org.opentripplanner.routing.algorithm.strategies
org.opentripplanner.routing.bike_park
org.opentripplanner.routing.bike_rental
org.opentripplanner.routing.core
org.opentripplanner.routing.edgetype
org.opentripplanner.routing.edgetype.factory
org.opentripplanner.routing.error
org.opentripplanner.routing.fares
org.opentripplanner.routing.genetic
org.opentripplanner.routing.genetic.de
org.opentripplanner.routing.graph
org.opentripplanner.routing.impl
org.opentripplanner.routing.location
org.opentripplanner.routing.request
org.opentripplanner.routing.services
org.opentripplanner.routing.services.notes
org.opentripplanner.routing.spt
org.opentripplanner.routing.transit_index
org.opentripplanner.routing.trippattern
org.opentripplanner.routing.util
org.opentripplanner.routing.vertextype
org.opentripplanner.scripting.api
org.opentripplanner.scripting.impl
org.opentripplanner.standalone
org.opentripplanner.traffic
org.opentripplanner.updater
org.opentripplanner.updater.alerts
org.opentripplanner.updater.bike_park
org.opentripplanner.updater.bike_rental
org.opentripplanner.updater.example
org.opentripplanner.updater.stoptime
org.opentripplanner.updater.street_notes
org.opentripplanner.updater.traffic
org.opentripplanner.util
org.opentripplanner.util.model
org.opentripplanner.util.monitoring
org.opentripplanner.util.stats
org.opentripplanner.util.xml
org.opentripplanner.visibility
org.opentripplanner.visualizer
```

Figure 95: Main structure of the developed prototype (part 2)

Each of these packages has its main objective and its context within the whole project. Furthermore, these packages are also comprised by several JAVA classes. Some of them have functionalities that fall out of the scope of the work made in Task T3.2, such as `org.opentriplanner.visualizer` or `org.opentripplanner.bike_rental`, among many others. For this reason, along this section we will explain the purpose of several crucial packages that have been considered in the work made, also with the description of the main work carried out within these packages.

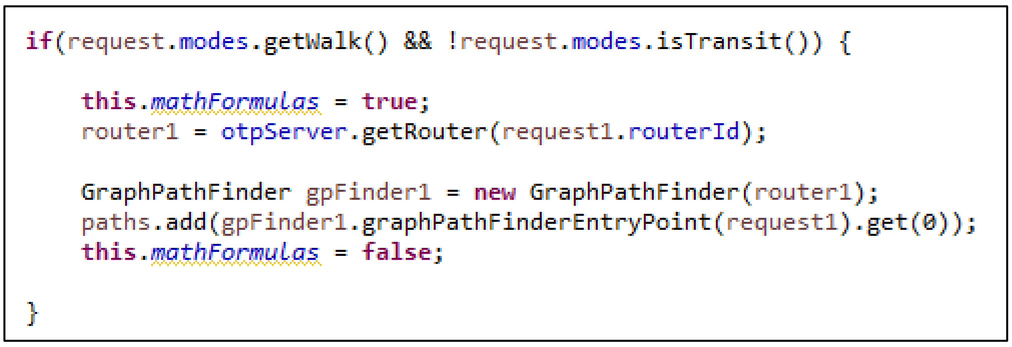
- `org.opentriplanner.routing.edgetype`: this package contains all the edge types that can built the complete graph that represents a city. Thus, it contains classes such as `PathwayEdge`, `StationEdge`, `TranserEdge` or `BikeParkEdge`. Among these classes, there are two specific ones that have been intensively used in this project for contemplate some of the modifications developed. These classes are `StreetEdge` and `StreetWithElevationEdge`. These two classes represent a common street of the city. Many modifications have been made on these two JAVA classes, but there is a specific method, coined as `doTraverse()`, which is the one that calculates the cost of traversing the street at hand. In overall, modifications on these classes have been made for developing functionalities previously detailed.
- `org.opentripplaner.api.resource`: This package contains crucial classes for properly dealing with the API call that make the route planner work. In this package, there is a class coined as PlannerResource, which is in charge of building the routing request and launch that route calculations. After that, it takes the results provided by the routing algorithm and it calls to the output methods in order to return solutions to the user as described in Section 4.1.4.5. This class has been severely modified in order to contemplate the calculation of the routes described in 4.1.1. For example, the

except of code depicted in Figure 96 has been added to permit the calculation of the routes, while the code represented in has been implemented for calculating the path described in Figure 97, and the code shown in Figure 98 for considering the equilibrated route described in 4.1.3.1.2.



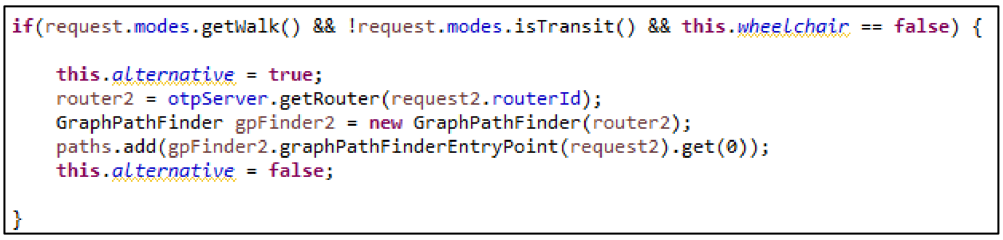
```
if(request.modes.getWalk() && !request.modes.isTransit() && request.enhancedRoute==true) {

    DEBest1Bin strategy = new DEBest1Bin();
    strategy.init(new DERandom());

    router0 = otpServer.getRouter(request0.routerId);
    GraphPathFinder gpFinder0 = new GraphPathFinder(router0);

    pOTP = new ProblemOTP(request0,gpFinder0, new GeneratorWalk(0.5));
    de = new T_DEOptimizer(20,strategy,4);

    de.optimize(pOTP);
    paths.add(pOTP.bestPath);

}
```

Figure 96: Except of the code added to `PlannerResource` to calculate AI Enhanced Routes



```
if(request.modes.getWalk() && !request.modes.isTransit()) {

    this.mathFormulas = true;
    router1 = otpServer.getRouter(request1.routerId);

    GraphPathFinder gpFinder1 = new GraphPathFinder(router1);
    paths.add(gpFinder1.graphPathFinderEntryPoint(request1).get(0));
    this.mathFormulas = false;

}
```

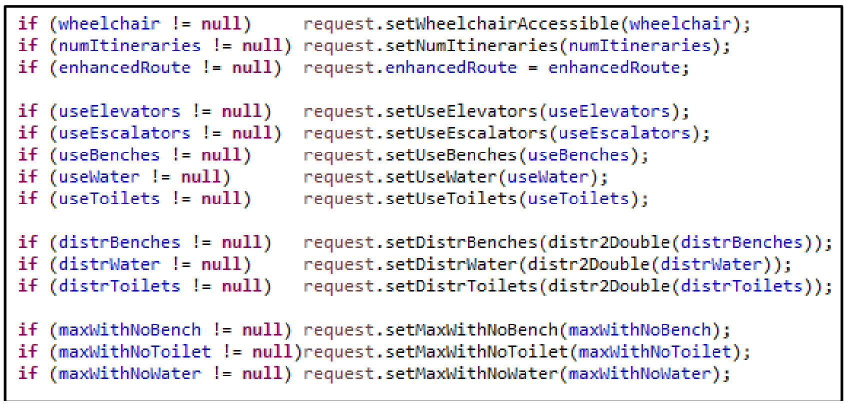
Figure 97: Except of the code added to `PlannerResource` to calculate the ASF-Based route



```
if(request.modes.getWalk() && !request.modes.isTransit() && this.wheelchair == false) {

    this.alternative = true;
    router2 = otpServer.getRouter(request2.routerId);
    GraphPathFinder gpFinder2 = new GraphPathFinder(router2);
    paths.add(gpFinder2.graphPathFinderEntryPoint(request2).get(0));
    this.alternative = false;

}
```

Figure 98: Except of the code added to `PlannerResource` to calculate the equilibrate Balanced route

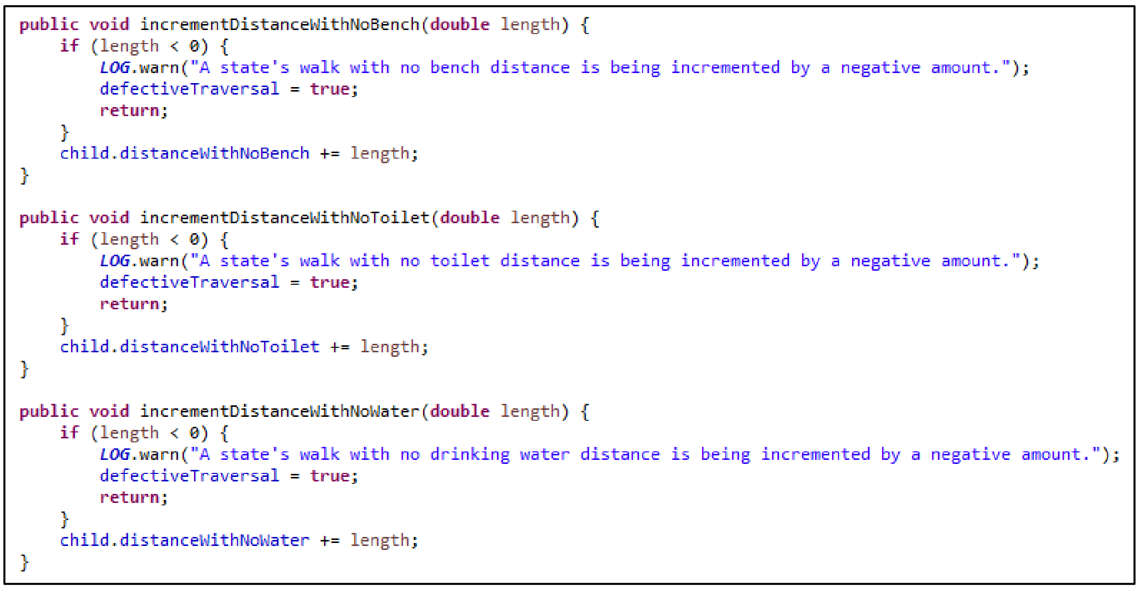
- `org.opentripplaner.api.common`: This package contains few classes, which are in charge of taking the API call introduced by the user (such as the ones described in Section 4.1.4.4) and properly reading it in order to extract and traduce all the information into JAVA variables. Within this package we can fiid the class `RoutingResource`, in which we have made several crucial modifications for contemplating all the newly added API parameters (described in Section 4.1.4.4). We represent in an excerpt Figure 99 of the modifications made in `RoutingResource` for contemplating new API parameters.



```
if (wheelchair != null)      request.setWheelchairAccessible(wheelchair);
if (numItineraries != null) request.setNumItineraries(numItineraries);
if (enhancedRoute != null)   request.enhancedRoute = enhancedRoute;

if (useElevators != null)    request.setUseElevators(useElevators);
if (useEscalators != null)   request.setUseEscalators(useEscalators);
if (useBenches != null)      request.setUseBenches(useBenches);
if (useWater != null)        request.setUseWater(useWater);
if (useToilets != null)      request.setUseToilets(useToilets);

if (distrBenches != null)    request.setDistrBenches(distr2Double(distrBenches));
if (distrWater != null)      request.setDistrWater(distr2Double(distrWater));
if (distrToilets != null)    request.setDistrToilets(distr2Double(distrToilets));

if (maxWithNoBench != null) request.setMaxWithNoBench(maxWithNoBench);
if (maxWithNoToilet != null)request.setMaxWithNoToilet(maxWithNoToilet);
if (maxWithNoWater != null) request.setMaxWithNoWater(maxWithNoWater);
```

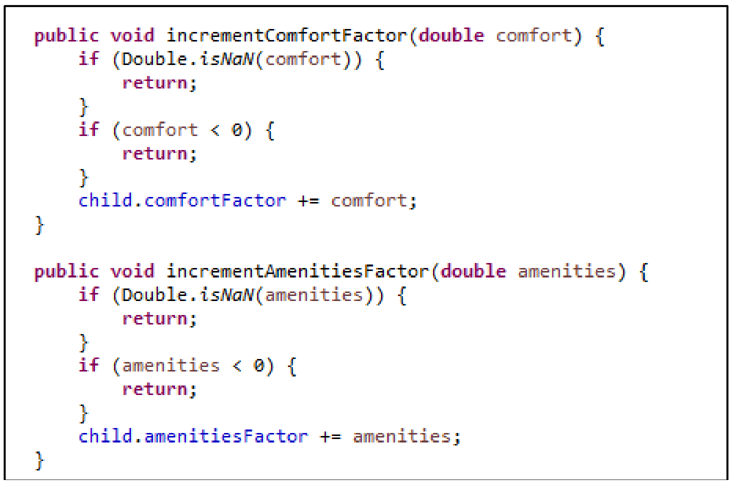
Figure 99: an excerpt of the modifications made in RoutingResource for contemplating new API parameters

- `org.opentripplaner.routing.core`: This package contains several classes used to control the routing process. Within the package, two different classes have been intensively modified. The first one is the coined as `RoutingRequest`, which includes all the requisites introduced by the user for calculating the route, such as the transit mode, origin, destination, date or time, among many others. This class has been modified for contemplating all the newly introduced parameters, such as the ones introduced in the table of Section 4.1.4.4. Additionally, a class named as `StateEditor` has also been modified. This class is the one in charge of evaluating the state of the route as well as it is being calculated by the AI algorithm. In this class several additions have been implemented in order to consider parameters such as the distance without finding a toilet along the route, or the distance without a bench (see Figure 100 for more detail). Also, several methods have been implemented in that class for calculating the overall comfortability and amenity factors, as can be seen in the excerpt of code represented in Figure 101.



```
public void incrementDistanceWithNoBench(double length) {
    if (length < 0) {
        LOG.warn("A state's walk with no bench distance is being incremented by a negative amount.");
        defectiveTraversal = true;
        return;
    }
    child.distanceWithNoBench += length;
}

public void incrementDistanceWithNoToilet(double length) {
    if (length < 0) {
        LOG.warn("A state's walk with no toilet distance is being incremented by a negative amount.");
        defectiveTraversal = true;
        return;
    }
    child.distanceWithNoToilet += length;
}

public void incrementDistanceWithNoWater(double length) {
    if (length < 0) {
        LOG.warn("A state's walk with no drinking water distance is being incremented by a negative amount.");
        defectiveTraversal = true;
        return;
    }
    child.distanceWithNoWater += length;
}
```

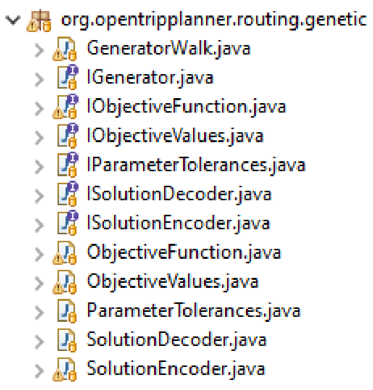
Figure 100: Excerpt of code added in `StateEditor` for considering the amount of distance without an amenity



```
public void incrementComfortFactor(double comfort) {
    if (Double.isNaN(comfort)) {
        return;
    }
    if (comfort < 0) {
        return;
    }
    child.comfortFactor += comfort;
}

public void incrementAmenitiesFactor(double amenities) {
    if (Double.isNaN(amenities)) {
        return;
    }
    if (amenities < 0) {
        return;
    }
    child.amenitiesFactor += amenities;
}
```

Figure 101: Excerpt of code added in `StateEditor` for incrementing the amenity and comfort factors.

- `org.opentripplaner.api.model`: This package contains several classes which are crucial for transforming the solution provided by the AI into the JSON that is then returned to the user. On this package, classes such as `TripPlan`, `Leg` or `Itinerary` have been modified in order to contemplate the new information provided by the Age-Friendly Route Planned (more details in Section 4.1.4.5).
- `org.opentripplaner.routing.genetic`: This is a newly generated package, which has been added to the OTP structure in order to contemplate the calculation of the AI-Enhanced routes described in 4.1.3.1.3. In this case, there are some classes employed for a generic implementation, such as the encoders and decoder, and also classes for defining which are the parameter for being modified by the solving metaheuristic. Also, objective function and values for the optimization metaheuristic are defined as well as some interface classes for being completed by the concrete definition of the algorithm.



```
org.opentripplanner.routing.genetic
    GeneratorWalk.java
    IGenerator.java
    IObjectiveFunction.java
    IObjectiveValues.java
    IParameterTolerances.java
    ISolutionDecoder.java
    ISolutionEncoder.java
    ObjectiveFunction.java
    ObjectiveValues.java
    ParameterTolerances.java
    SolutionDecoder.java
    SolutionEncoder.java
```

Figure 102: structure of the newly created package org.opentripplaner.routing.genetic

In the following table we describe the main functionality of each of the classes implemented in this package.

Name of the class	Description of the class
ParameterTolerances	This class contains all the upper and lower limits of the variables that compose the solution.
GeneratorWalk	This class generated the ranges in which the variables should be found.
ObjectiveFunction	This class has a method named as `evaluate()` which evaluates the solutions of the population. This is strictly related to the class coined as `ProblemOTP`, and described below. This method returns an *ObjectiveValues* instance.
ObjectiveValues	This class, which is the output of the above mentioned `evaluate()` method of class `ObjectiveFunction` contains the specific values that a solution has for each parameter.
SolutionDecoder	This class is in charge of decoding a solution generated by the DE, creating the corresponding `RoutingRequest`.
SolutionEncoder	This last class is the responsible of receiving a routing request and extracting the array of values that compose the solutions used by the DE.

Table 18: Functionalities of main classes in package `org.opentripplaner.routing.genetic`:

- `org.opentripplaner.routing.genetic.de`: This newly generated package is a concretization of a differential evolution algorithm, employed for calculating the routes described in 4.1.3.1.4. Furthermore, this package uses the definitions included in the previously described `org.opentripplaner.routing.genetic`. It should be highlighted that many operators have been considered in the package as part of the process of parameterization of the method.

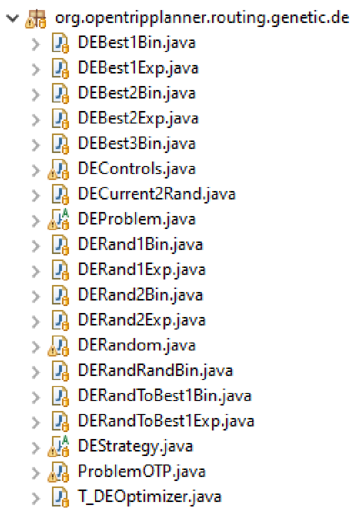

org.opentripplanner.routing.genetic.de
- DEBest1Bin.java
- DEBest1Exp.java
- DEBest2Bin.java
- DEBest2Exp.java
- DEBest3Bin.java
- DEControls.java
- DECurrent2Rand.java
- DEProblem.java
- DERand1Bin.java
- DERand1Exp.java
- DERand2Bin.java
- DERand2Exp.java
- DERandom.java
- DERandRandBin.java
- DERandToBest1Bin.java
- DERandToBest1Exp.java
- DEStrategy.java
- ProblemOTP.java
- T_DEOptimizer.java

Figure 103: structure of the newly created package org.opentripplaner.routing.genetic.de

We detail in the following table the main functionality of each of the classes implemented in this package.

Name of the class	Description of the class
T_DEOptimizer	this is the DE algorithm. In this class, some methods have been developed for properly run the technique, as the iterative phase, the survivor function and the initialization procedure.
ProblemOTP	This class is in charge of evaluating each solution created by the DE. The specific method `evaluate()` gets as input a configuration generated by the DE, then decodes it, and calls to the OTP routing for generating the route associated to this configuration.
DERandom	This class generated random numbers.
DEBest1Bin (and other variants)	These classes depict the operators employed by the DE for generating new candidate solutions. Both the crossover function and the mutation function.

Table 19: Functionalities of main classes of package `org.opentripplaner.routing.genetic.de`

In addition to all these modifications, further work has been made in the client side, in order to adapt the testing-purpose webpage to the URBANAGE needs. All these modifications have been conducted in the packages that can be found in `src/client`.

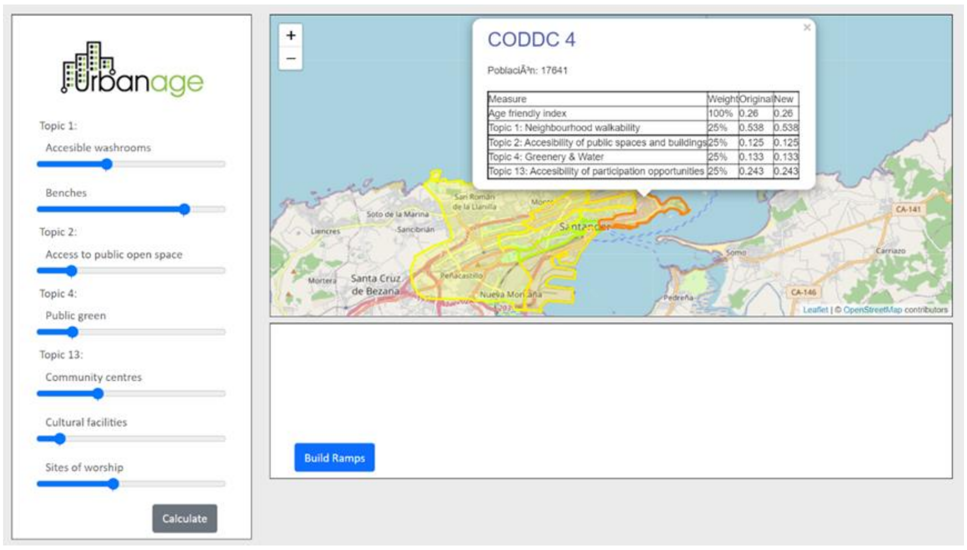
7.2 Simulation tool for long-term urban planning

7.2.1 User Manual

In this section a list of user interactions is provided:

7.2.1.1 How to see the KPI values of a district

Access the simulation tool interface. When the map is loaded, and blue sliders appear you can select one district by clicking into its polygon as can be seen in Figure 104 . This will update the blue sliders to the right to showcase this district's indicator values.



CODDC 4

PoblaciÃ³n: 17641

Measure	Weight	Original	New
Age friendly index	100%	0.26	0.26
Topic 1: Neighbourhood walkability	25%	0.538	0.538
Topic 2: Accesibility of public spaces and buildings	25%	0.125	0.125
Topic 4: Greenery & Water	25%	0.133	0.133
Topic 13: Accesibility of participation opportunities	25%	0.243	0.243

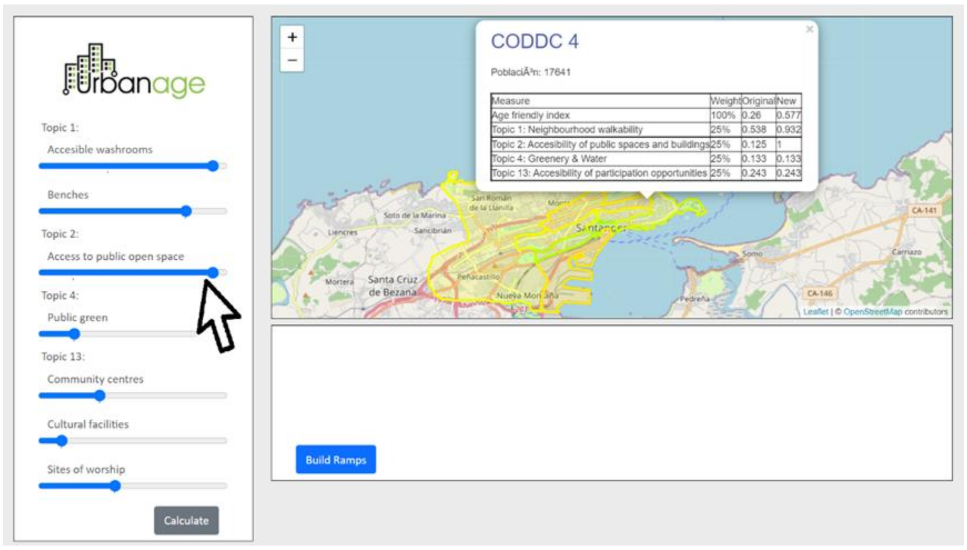
Figure 104: Age Friendliness Index represented in a color gradient and all the initial indicator values.

7.2.1.2 How to see the Age-friendliness index of a district

Follow the steps of "How to see the KPI values of a district". When a district is clicked on a table will pop up in the map. This table contains the Age-friendliness index of the selected district.

7.2.1.3 How to change KPI values

Follow the steps of "How to see the KPI values of a district".
Now to change the value of the KPIs, just like can be seen in the difference between Figure 104 and Figure 105, by clicking and dragging the blue sliders all values will be updated client-side. This will immediately update the map and index values.



CODDC 4

PoblaciÃ³n: 17641

Measure	Weight	Original	New
Age friendly index	100%	0.26	0.577
Topic 1: Neighbourhood walkability	25%	0.538	0.932
Topic 2: Accesibility of public spaces and buildings	25%	0.125	1
Topic 4: Greenery & Water	25%	0.133	0.133
Topic 13: Accesibility of participation opportunities	25%	0.243	0.243

Figure 105: Age Friendliness Index represented in a color gradient and all the updated indicator values.

7.2.1.4 How to see the difference between the original index and the new one after the KPI modification

Access the simulation tool interface. When the map is loaded, and blue sliders appear you can select one district by clicking into its polygon as can be seen in Figure 105. Below the sliders a grey line will appear with the original values. If these values are changed to simulate a scenario the line will be kept still to be able to double check what the original value was. Additionally, the map pops up a table with all the original values and the current state of the simulation.

7.2.1.5 How to add a ramp

Access the simulation tool interface. When the map is loaded, and blue sliders appear you can click on the blue button “Build a ramp” that is shown in Figure 104 which will disable the district layout. Now that the districts are not on the way whenever there is a click in the map a purple line will appear describing the closest road that Nominatim finds next to the location where there was the click. This behaviour can be seen in Figure 106, If clicked again in roughly the same location the ramp will be deleted. If clicked elsewhere a new road will be added to the ramps list.

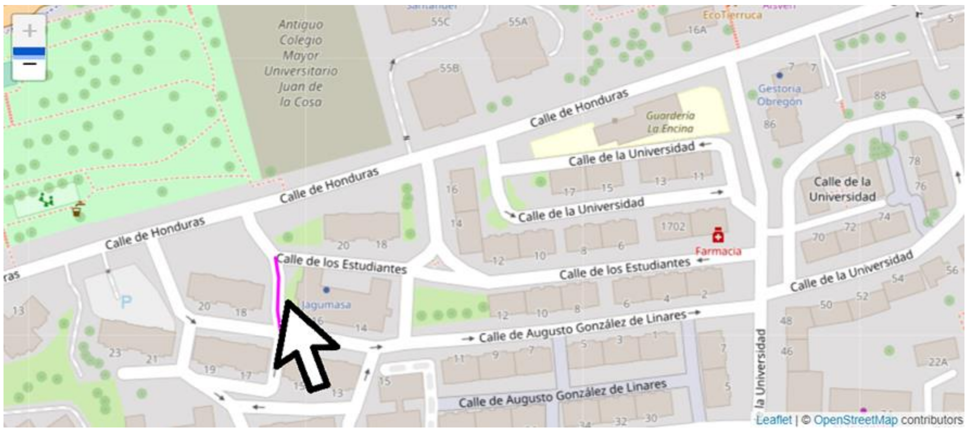


Figure 106: adding a ramp after clicking into Build Ramps and then clicking on a street.