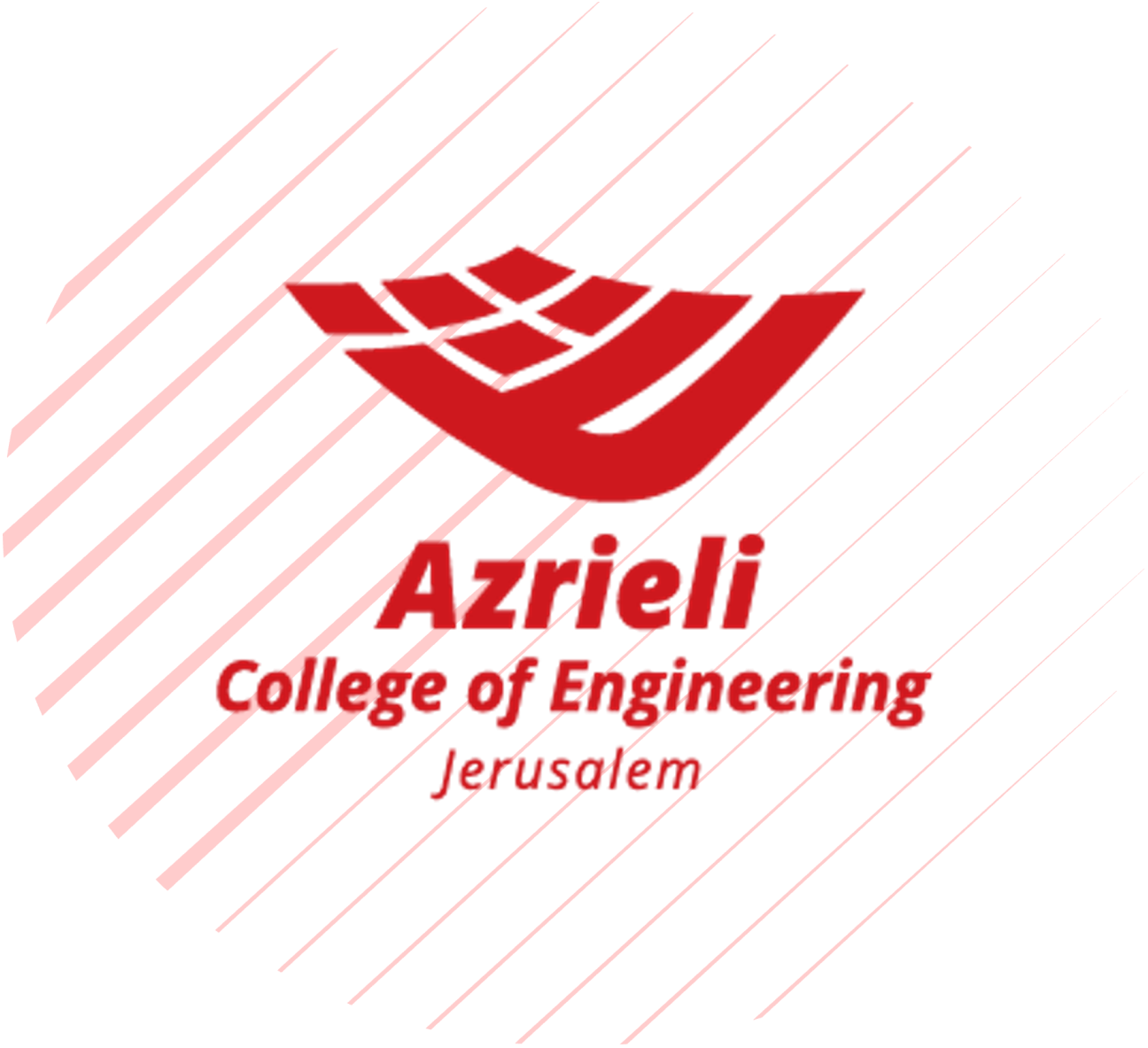
Open Trip Planner



GIS Technologies & Internet Infrastructure Mapping

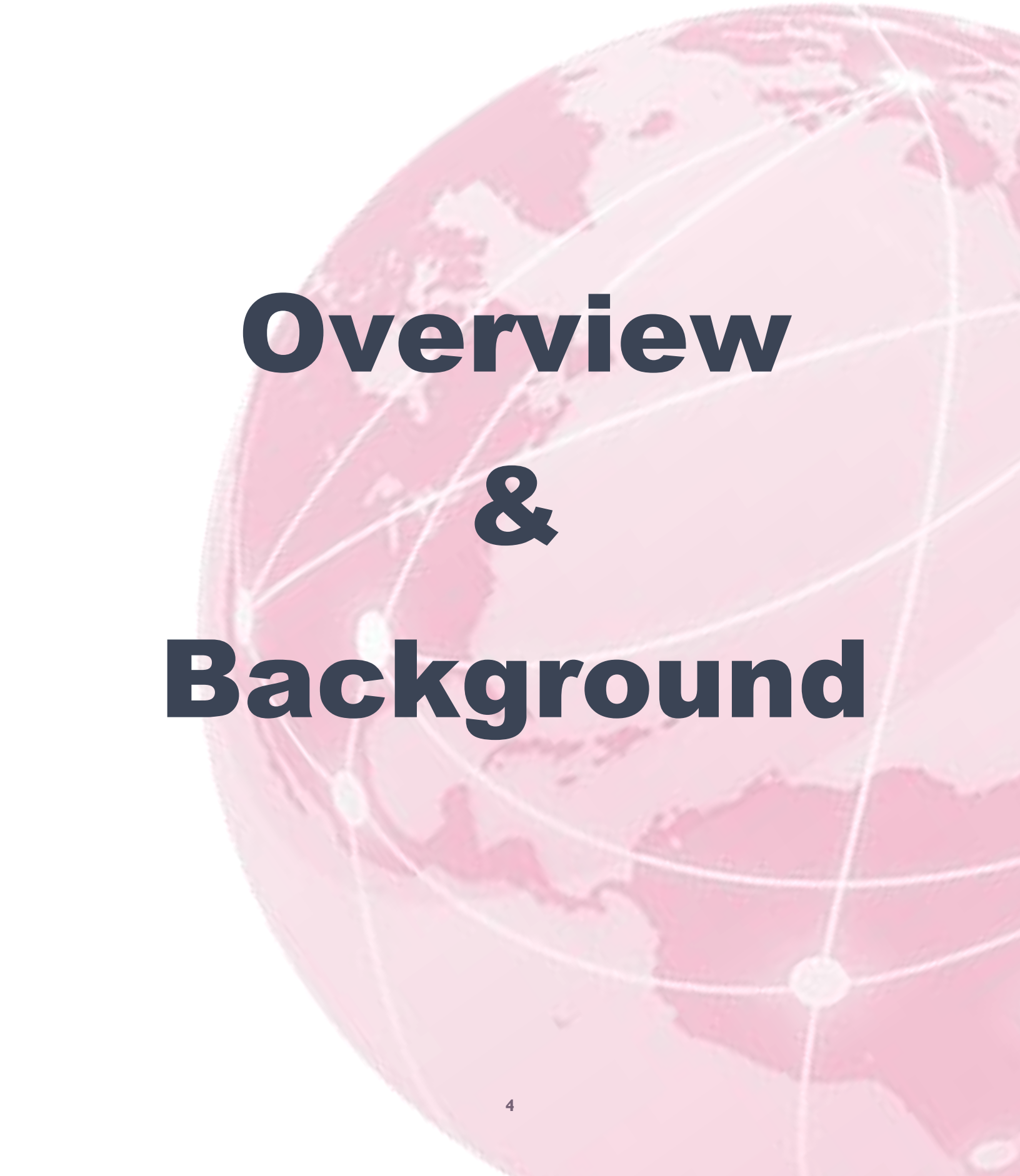
12/05/2023

Itzhak Eli

Hakam Nabulssi & Marah Mani

Instructor Evaluation

TABLE OF CONTENTS

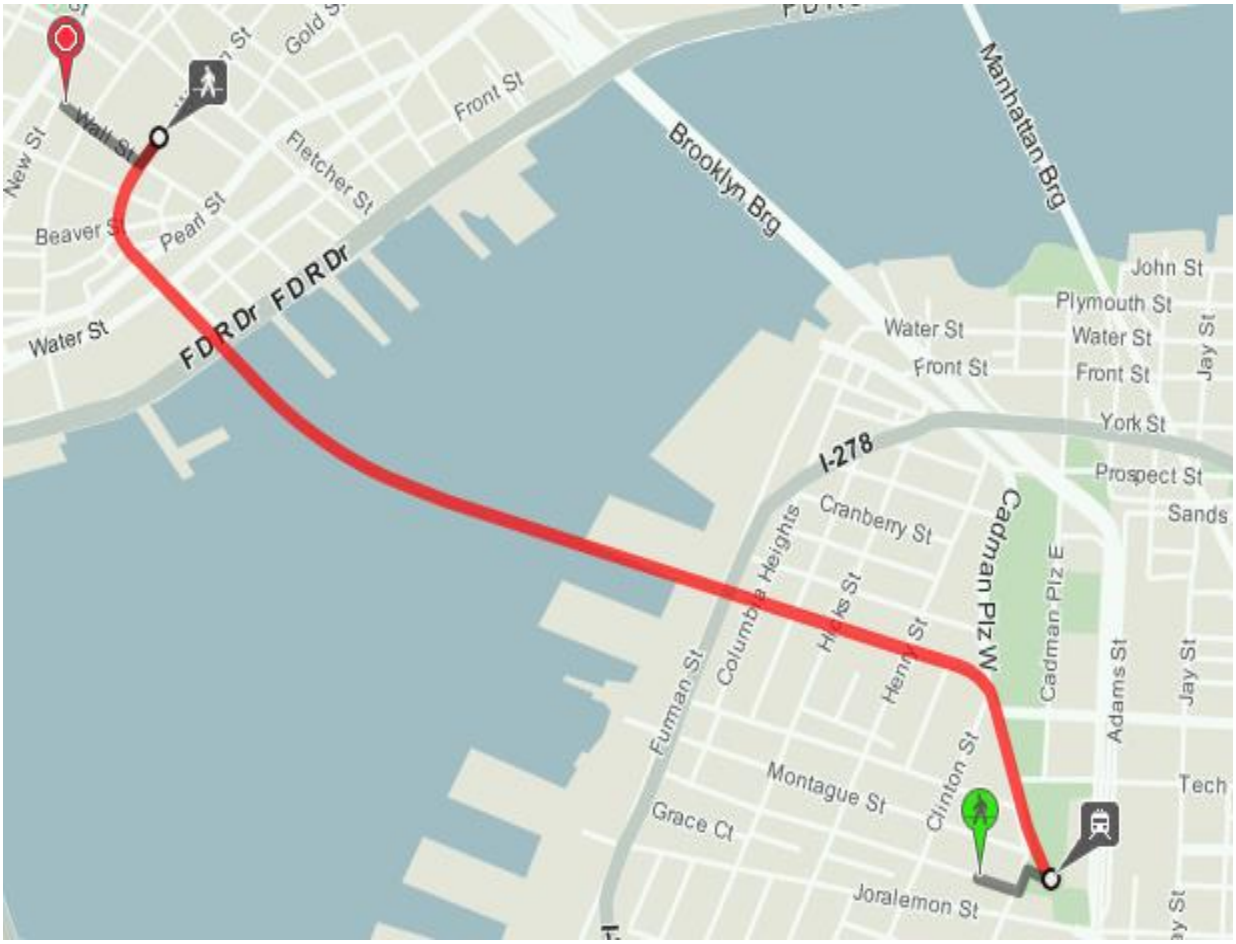


Overview & Background

Introduction

The OpenTripPlannerPlugin is a QGIS plug-in that provides access to OpenTripPlanner (OTP) functionalities within QGIS. OTP is a well-known open-source platform for multimodal trip itinerary planning and transportation system analysis, established in 2009 and deployed in 14 countries.

OTP has a close association with the OpenStreetMap project, with most OTP deployments relying on OSM data for street network routing, and OTP deployment driving local OSM improvement efforts in many communities.



The OTP Plugin is a useful tool for transportation researchers, urban planners, and smart city developers. It allows for easy configuration of different parameters, such as departure time and travel modes, and provides sophisticated spatial analysis capabilities to evaluate accessibility and mobility among different population groups within transportation systems and networks. It can also be used to assess the impact of new transit lines or bike lanes on travel patterns, identify areas with low accessibility, and optimize emergency vehicle routing during disasters.

Plugin Selection

During our research, we compared the OpenTripPlannerPlugin with other transportation-related plugins available in the QGIS Plugin Repository, such as the GRASS GIS plugin and the NetworkX plugin. We found that while the GRASS GIS plugin had similar functionality to the OpenTripPlannerPlugin, it lacked the same level of user-friendliness and required more technical expertise to use effectively. Additionally, the GRASS GIS plugin did not have the same level of support and documentation available online, which made it more difficult to troubleshoot any issues we encountered.

The NetworkX plugin, on the other hand, was limited in its routing capabilities and was not well-suited for our project's specific needs. It did not offer the same range of routing options for multiple modes of transportation or the ability to compute isochrones as the OpenTripPlannerPlugin did.

Based on our research and evaluation of available options, we ultimately chose the OpenTripPlannerPlugin for our transportation planning project. Our specific needs and goals included analysing the accessibility and mobility of different population groups within transportation systems and networks.

We selected the OpenTripPlannerPlugin for several reasons:

- It offers a comprehensive set of routing options for multiple modes of transportation, including walking, cycling, and driving, which is essential for our project.
- The plugin has sophisticated spatial analysis capabilities and can compute areas reachable within specified distances known as isochrones.
- It integrates seamlessly with OpenTripPlanner, a leading open-source platform for multimodal trip itinerary planning and transportation system analysis.

Learning The Plugin

To fully explore the capabilities of the OpenTripPlanner (OTP) plugin, you must start by conducting thorough research. Since your initial knowledge of the plugin may be limited, you should begin by searching for resources online. This can include visiting the OTP website, which provides a detailed overview of the plugin's features and functionality. The website also offers tutorials, user guides, and documentation that can help you gain a better understanding of the plugin.

Another valuable resource to utilize is the OTP GitHub page. This page provides access to the plugin's source code and documentation, which can help you gain a deeper understanding of how the plugin works. Additionally, you can find several blog posts and forum discussions from other users who have experience with the plugin. These resources can offer valuable insights, tips, and best practices for using OTP effectively.

Once you have a solid understanding of OTP's capabilities, you can gain hands-on experience by installing it on your local machine. Follow the installation instructions provided on the OTP website, which are clear and easy to follow. Once you have OTP up and running, you can start by testing simple scenarios, such as finding the fastest route between two points in your local area. As you become more comfortable with the plugin, you can experiment with more complex scenarios, such as multi-modal trip planning and real-time updates.

OTP Plugin Key Applications

Public Transportation Planning: Generates multiple route options when selecting the starting point and destination for planning public transportation routes in a city or region.

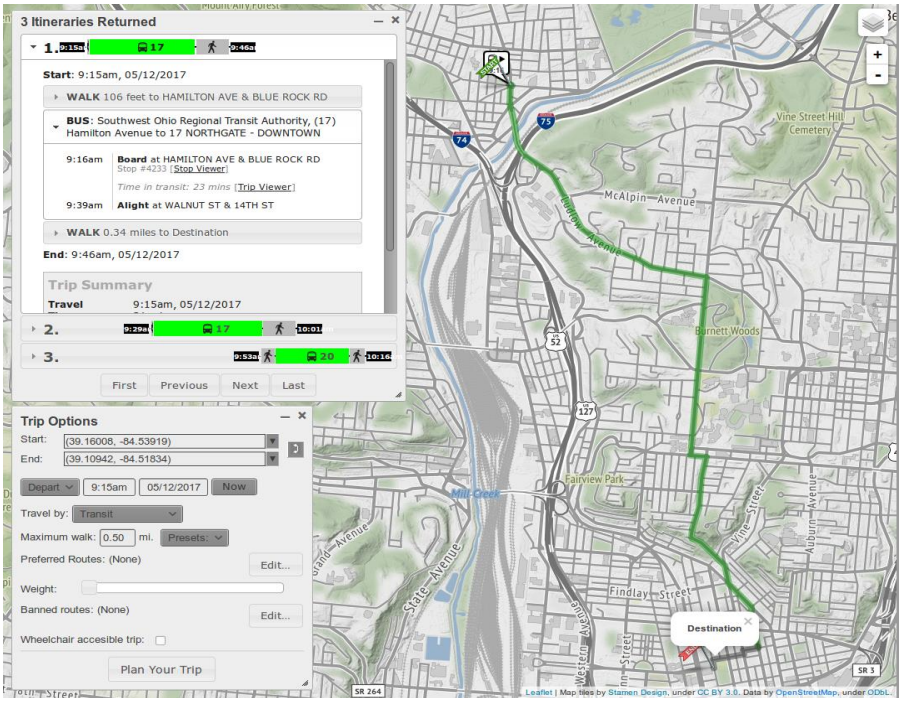
Bicycle Routing: Plans optimal bike routes by taking into account bike lanes, trails, traffic volume, and elevation changes.

Pedestrian Routing: Finds the best pedestrian routes by considering sidewalks, crosswalks, and pedestrian-friendly areas.

Accessibility Planning: Plans routes that cater to the needs of individuals with disabilities by taking into account accessibility features such as curb cuts, wheelchair ramps, and accessible transportation options.

Urban Planning: Analyzes transportation accessibility in different areas of a city, considering various transportation modes to inform urban planning decisions and guide future development initiatives.

Tourism Planning: Plans tourist routes and assesses the accessibility of various tourist destinations to create comprehensive itineraries and provide relevant information to visitors regarding transportation options and accessibility considerations.

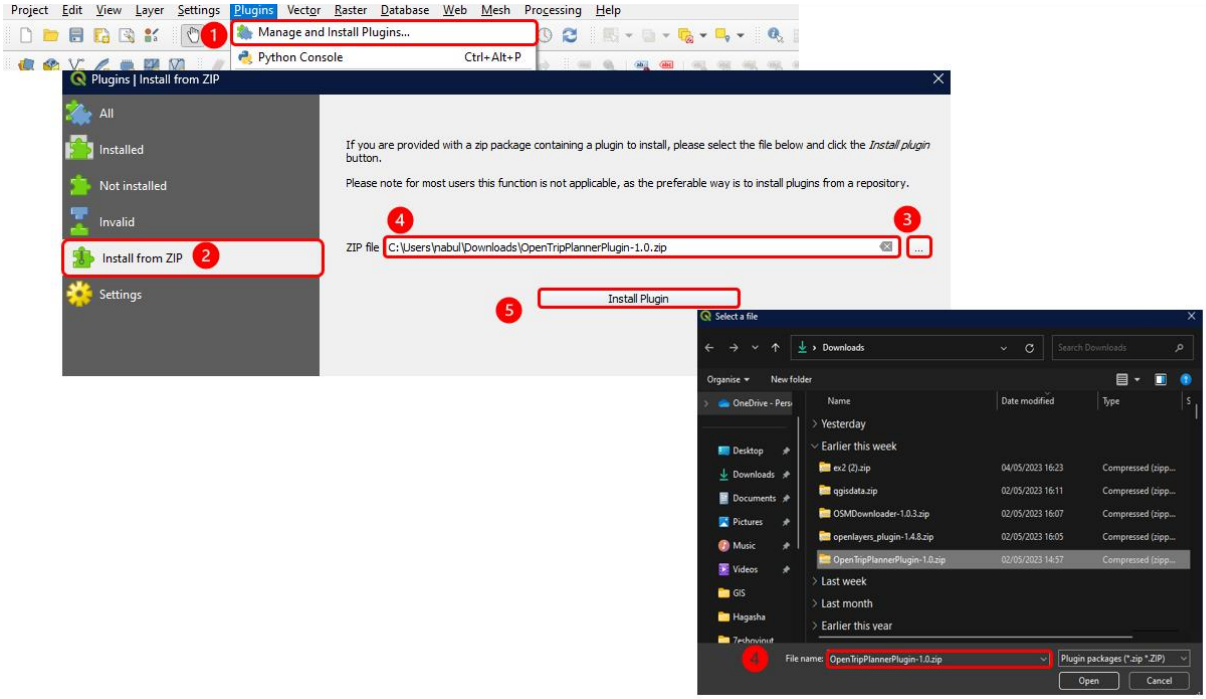


** For detailed instructions on the steps to use the plugin and get started, please refer to the User Guide

USER GUIDE

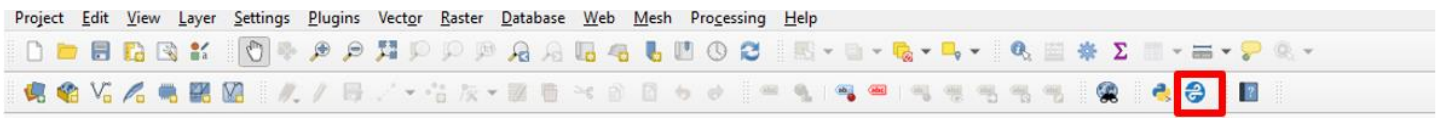
✓ Install the Plug-In

Download the OpenTripPlannerPlugin zip file from the OpenTripPlannerPlugin website, Open QGIS, and click on "Plugins" from the top menu. Choose "Manage and Install Plugins" and click on the "Install from ZIP" button. Navigate to the downloaded OpenTripPlannerPlugin zip file and select it. Install the plugin by clicking on the "Install Plugin" button.

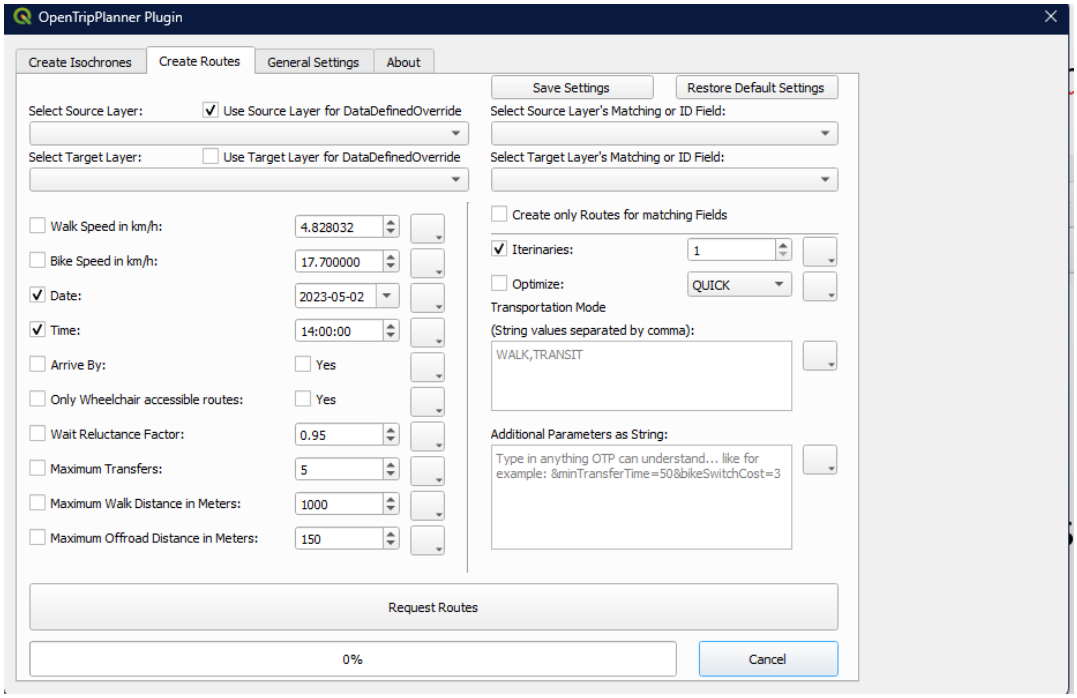


✓ Access OTP functionalities

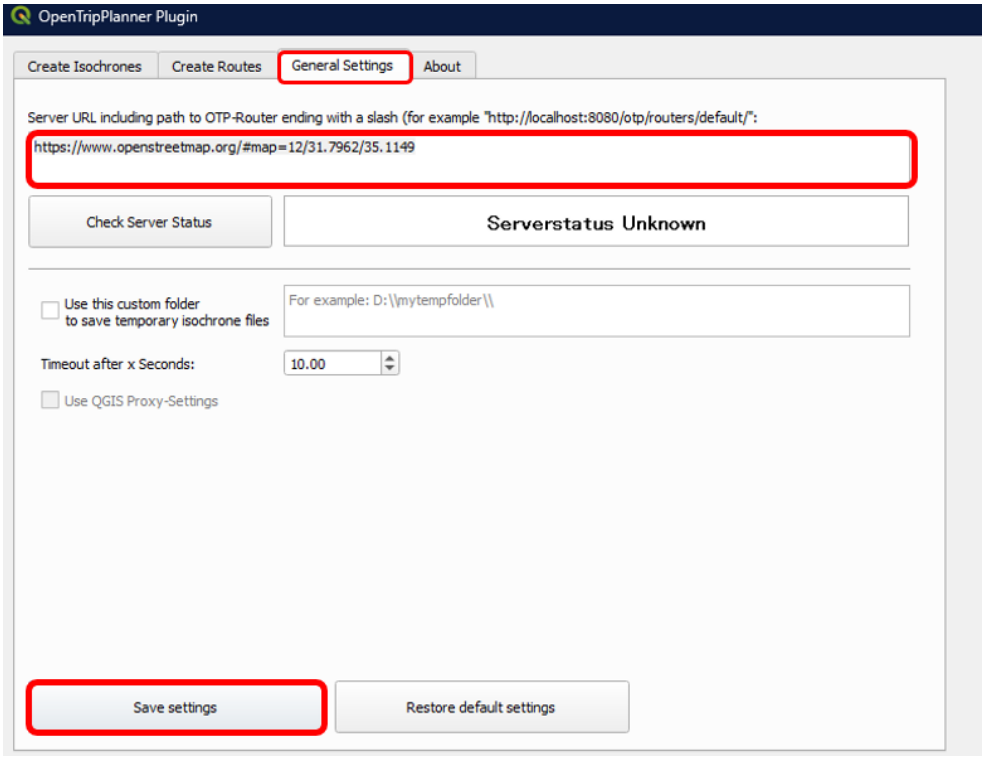
After installing the plugin, a new button "OpenTripPlanner" will show up in the toolbar.



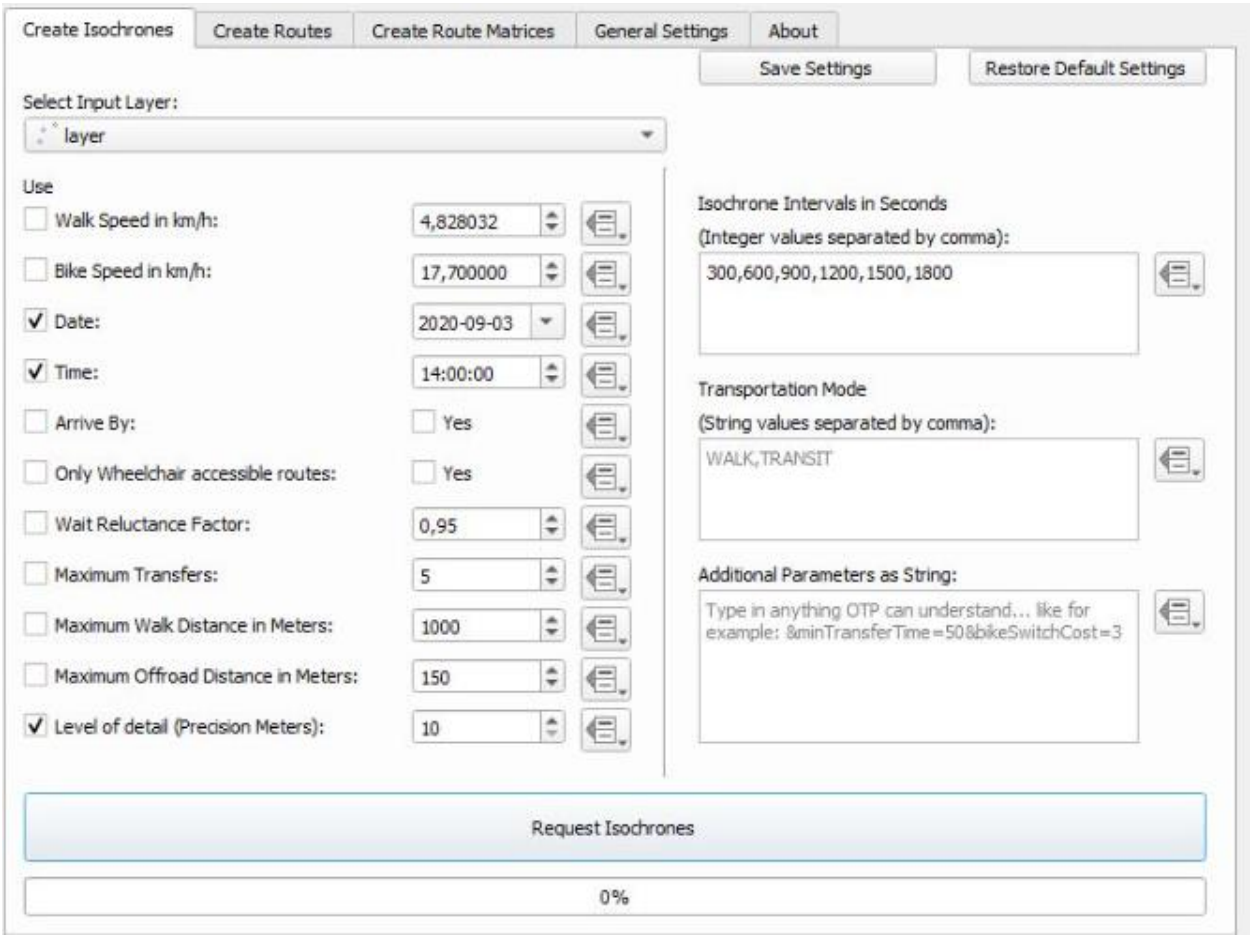
✓ **Click on this button to open the OTP window.**



✓ **General Settings**:
Go to the "General Settings" tab and enter a Server URL including the path to the OTP router, which can be accessed without Proxy or Authentication.



✓ **Choose your Settings**:
Choose your desired settings, such as the departure or arrival time, travel mode, and maximum distance.



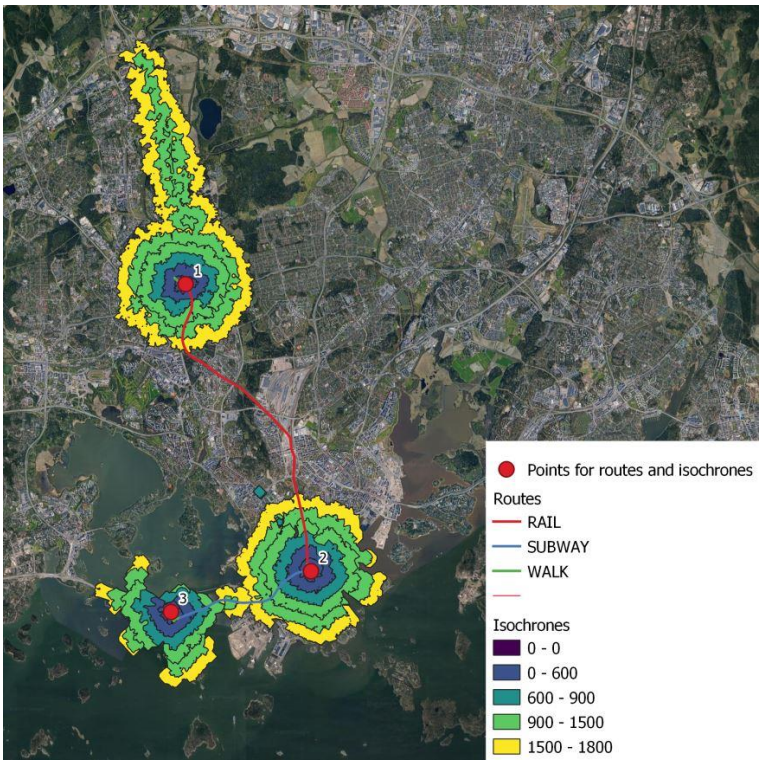
You can find more details about each setting by reading tooltips on mouse hover or checking OTP-API-Docs.

✓ **Run the plugin**:
Click on the "Run" button to execute the OTP request.

Example (isochrones)

Regarding the "**Aggregated Isochrones**" function, it is an experimental feature that the plugin provides to aggregate isochrones based on a given set of parameters. If you choose "Raw", the plugin will return the raw results for each requested datetime for each feature, which can be useful for debugging. If you choose "Maximum only (via Dissolve)", the plugin will create a temporary layer for each feature, store the isochrones for each datetime iteration in that temporary layer, and dissolve the layer by time field. The result will be the most optimistic service area, reachable at some point during the given time range.

On the other hand, if you choose "All possible Aggregations (via Union)", the plugin will create a temporary layer for each feature, process the response isochrones with several algorithms, including fixing geometries, union, joining attributes by location, and deleting duplicate geometries. Then, the plugin will add the processed temporary layer's features to the output layer. The result will be an aggregated isochrone that provides various statistical summaries, including count, unique, min, max, range, sum, mean, and median. These summaries provide useful information about the reachability of the area within the given time range. For example, count tells you how many times the area is reachable during the given time range, unique tells you if there are differences in the time needed to reach the area during the given time range, min tells you the most optimistic service area, and max tells you the most pessimistic service area.

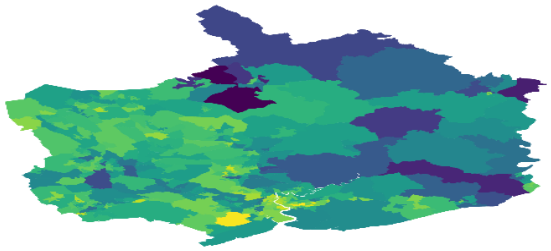


** If you are interested in exploring more OTP deployments, you can find a list on the official OTP documentation website.

Technical Section

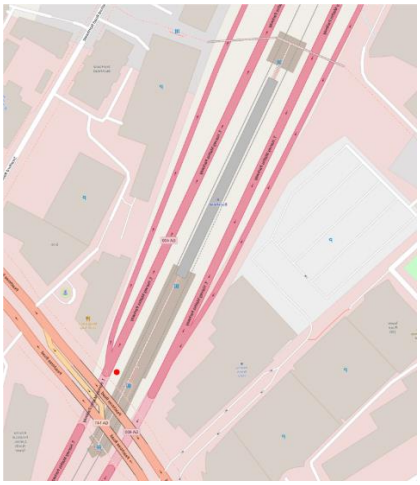
Code Description

The Open Trip Planner Plugin is a Python-based tool for transportation route planning and visualization. It seamlessly integrates with other applications and uses external libraries such as PyQt5, requests, and GeoPandas to ensure robust functionality.



The plugin is implemented as a QGIS Processing algorithm and is accessible through the QGIS Processing Toolbox. It requires users to have an OpenTripPlanner instance running locally or remotely, which the plugin communicates with via the OTP REST API. To import and process geospatial data, the plugin leverages the GeoPandas library, which provides spatial operation capabilities such as buffering and intersection. It uses the NetworkX library to create and manage network graphs, and the Dijkstra algorithm to calculate the shortest paths between two network nodes and find the optimal path. The plugin can import data from diverse sources like shapefiles, CSV files, and OSM files.

The plugin provides various tools for displaying the outcomes of the trip planning algorithm, including visualizing the shortest path on a map and providing information about the total distance and time required to complete the journey. The plugin also includes a graphical user interface created using PyQt5, which offers a comprehensive set of GUI tools and widgets. The GUI features a dialog window with input fields and configuration options for the routing operation.

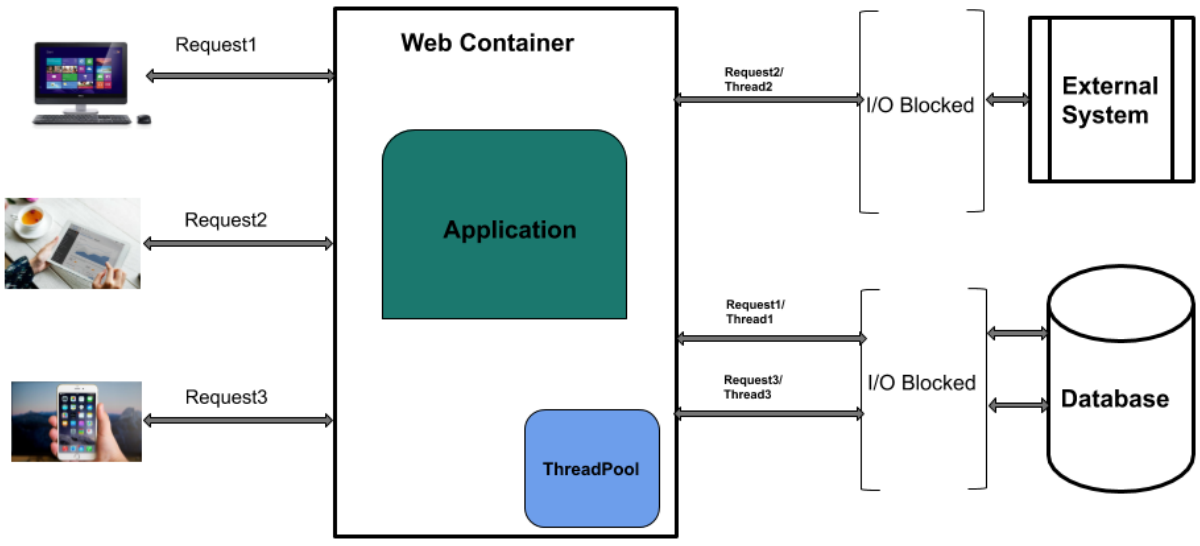


Code Files

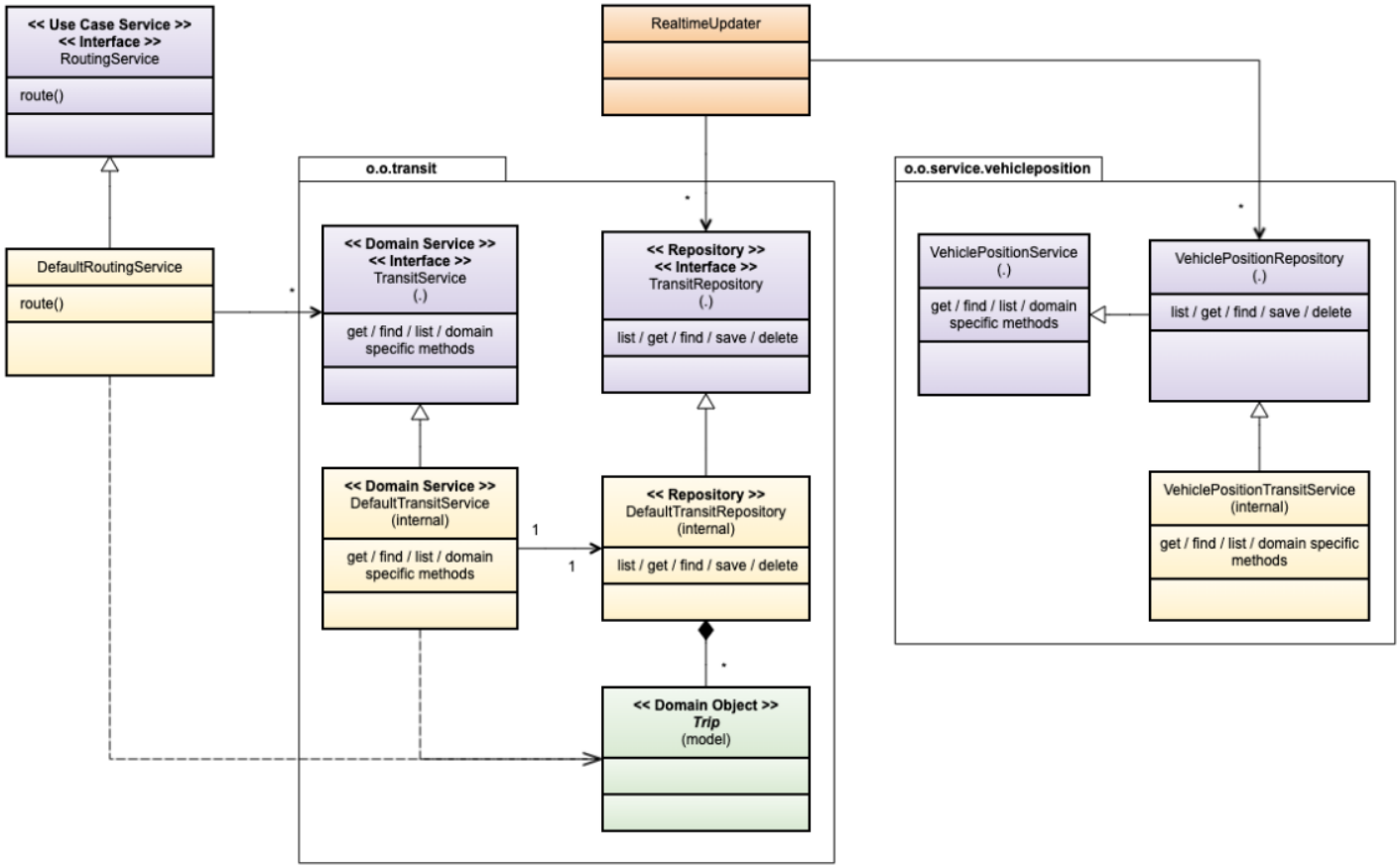
The plugin comprises several organized Python modules within packages, each serving a specific purpose.

- **The otp_plugin.py** file is the main file of the plugin, defining its user interface and functionality.
- **The otp_plugin_dialog.py** file defines the dialog window displayed when the user clicks on the plugin button.
- **The otp_plugin_general_functions.py** file contains general utility functions used throughout the plugin.
- **The otp_plugin_worker_aggregated_isochrones.py, otp_plugin_worker_isochrones.py, and otp_plugin_worker_routes.py** files contain the code responsible for calculating aggregated isochrones, isochrones, and routes, respectively.
- **The metadata.txt** file contains essential metadata about the plugin, and the
- **resources.py** file contains the necessary code for loading the plugin icon and other associated resources.
- **The __init__.py** file serves as an indicator to Python that the OpenTripPlannerPlugin directory should be treated as a Python package

Traditional Rest API Model



OTP Platform Use Case Diagram



This use case diagram shows a transportation service system that provides routing services for trips. The system includes services, repositories, and domain objects.

- The **RoutingServices** use case provides the main routing functionality and depends on the **DefaultRoutingService** implementation.
- The **TransitService** domain service provides transit-related data and depends on the **DefaultRoutingService**, **DefaultTransitService**, and **VehiclePositionService** and Repository use cases.
- The **TransitRepository** stores transit-related data and depends on the **TransitService** and **DefaultTransitService** use cases.
- The **VehiclePositionService** and Repository use cases provide real-time vehicle location data and depend on the **TransitService** and **DefaultTransitService** use cases.
- The Trip domain object stores data related to a trip and is stored in the **DefaultTransitRepository**.

The "run" Method of The OTP Plugin

```
def run(self):
    """Run method that performs all the real work"""
    # Create the dialog with elements (after translation) and keep reference
    # Only create GUI ONCE in callback, so that it will only load when the plugin is started
    if self.first_start == True:
        self.first_start = False
        self.dlg = OpenTripPlannerPluginDialog()
        self.gf = OpenTripPlannerPluginGeneralFunctions(self.dlg, self.iface)
        # Calling maplayer selection on first startup to load layers into QgsMapLayerComboBox and initialize
        # QgsOverrideButton stuff so selections can be done without actually using the QgsMapLayerComboBox
        # (related to currentIndexChanged.connect(self.isochrones_maplayerselection) below)
        self.gf.routes_maplayerselection()
        self.gf.isochrones_maplayerselection()
        self.gf.aggregated_isochrones_maplayerselection()
        # Execute Main-Functions on Click: Placing them here prevents them from beeing executed multiple times,
        # see https://gis.stackexchange.com/a/137161/107424
        self.dlg.Isochrones_RequestIsochrones.clicked.connect(lambda: self.isochronesStartWorker())  # Call the start worke
            method
        self.dlg.Isochrones_Cancel.clicked.connect(lambda: self.isochronesKillWorker())
        self.dlg.AggregatedIsochrones_RequestIsochrones.clicked.connect(lambda: self.aggregated_isochronesStartWorker())  #
            Call the start worker method
        self.dlg.AggregatedIsochrones_Cancel.clicked.connect(lambda: self.aggregated_isochronesKillWorker())
        self.dlg.Routes_RequestRoutes.clicked.connect(lambda: self.routesStartWorker())
        self.dlg.Routes_Cancel.clicked.connect(lambda: self.routesKillWorker())\
        # Calling Functions on button click
        self.dlg.GeneralSettings_CheckServerStatus.clicked.connect(self.gf.check_server_status)
        self.dlg.GeneralSettings_Save.clicked.connect(self.gf.store_general_variables)  # Call store_general_variables
            function when clicking on save button
        self.dlg.GeneralSettings_Restore.clicked.connect(self.gf.restore_general_variables)
        self.dlg.Isochrones_SaveSettings.clicked.connect(self.gf.store_isochrone_variables)
        self.dlg.Isochrones_RestoreDefaultSettings.clicked.connect(self.gf.restore_isochrone_variables)
        self.dlg.Isochrones_Now.clicked.connect(self.gf.set_datetime_now_isochrone)
        self.dlg.AggregatedIsochrones_SaveSettings.clicked.connect(self.gf.store_aggregated_isochrone_variables)
        self.dlg.AggregatedIsochrones_RestoreDefaultSettings.clicked.connect(self.gf.restore_aggregated_isochrone_variables
        self.dlg.AggregatedIsochrones_Now.clicked.connect(self.gf.set_datetime_now_aggregated_isochrone)
        self.dlg.Routes_SaveSettings.clicked.connect(self.gf.store_route_variables)
        self.dlg.Routes_RestoreDefaultSettings.clicked.connect(self.gf.restore_route_variables)
        self.dlg.Routes_Now.clicked.connect(self.gf.set_datetime_now_route)
        # Calling Functions to update layer stuff when layerselection has changed
        self.dlg.Isochrones_SelectInputLayer.currentIndexChanged.connect(self.gf.isochrones_maplayerselection)  # Call
            function isochrones_maplayerselection to update all selection related stuff
          # when selection has been changed
        self.dlg.AggregatedIsochrones_SelectInputLayer.currentIndexChanged.connect(self.gf
            .aggregated_isochrones_maplayerselection)
        self.dlg.Routes_SelectInputLayer_Source.currentIndexChanged.connect(self.gf.routes_maplayerselection)
        self.dlg.Routes_SelectInputLayer_Target.currentIndexChanged.connect(self.gf.routes_maplayerselection)
        self.dlg.Routes_SelectInputField_Source.currentIndexChanged.connect(self.gf.routes_maplayerselection)  # or
            "fieldChanged"?
        self.dlg.Routes_SelectInputField_Target.currentIndexChanged.connect(self.gf.routes_maplayerselection)
        self.dlg.Routes_DataDefinedLayer_Source.stateChanged.connect(self.gf.routes_maplayerselection)
        self.dlg.Routes_DataDefinedLayer_Target.stateChanged.connect(self.gf.routes_maplayerselection)

    # Setting GUI stuff for startup every time the plugin is opened
    self.dlg.Isochrones_Date.setDateTime(QtCore.QDateTime.currentDateTime())  # Set Dateselection to today on restart or
        firststart, only functional if never used save settings,
      # otherwise overwritten by read_route_variables()
    self.dlg.AggregatedIsochrones_FromDateTime.setDateTime(QtCore.QDateTime.currentDateTime())
    self.dlg.AggregatedIsochrones_ToDateTime.setDateTime(QtCore.QDateTime.currentDateTime()
    self.dlg.Routes_Date.setDateTime(QtCore.QDateTime.currentDateTime())
    self.dlg.Isochrones_ProgressBar.setValue(0)  # Set Progressbar to 0 on restart or first start
    self.dlg.AggregatedIsochrones_ProgressBar.setValue(0)
    self.dlg.Routes_ProgressBar.setValue(0)
    self.dlg.GeneralSettings_ServerStatusResult.setText("Serverstatus Unknown")
    self.dlg.GeneralSettings_ServerStatusResult.setStyleSheet("background-color: white; color: black ")
    # Functions to execute every time the plugin is opened
    self.gf.read_general_variables()  # Run Read-Stored-Variables-Function on every start
    self.gf.read_isochrone_variables()
    self.gf.read_aggregated_isochrone_variables()
    self.gf.read_route_variables()
    # show the dialog
    self.dlg.show()
    # Run the dialog event loop
    result = self.dlg.exec_()
    # See if OK was pressed
    if result:
        # Do something useful here - delete the line containing pass and
        # substitute with your code.
        QgsMessageLog.logMessage(
            "OpenTripPlanner Plugin is already running! Close it before, if you wish to restart it.",
            MESSAGE_CATEGORY,
            Qgis.Warning,
        )
        self.iface.messageBar().pushMessage(
            "Error",
            "OpenTripPlanner Plugin is already running! Close it before, if you wish to restart it.",
            MESSAGE_CATEGORY,
            level=Qgis.Warning,
            duration=6,
        )
```

At the beginning of the method, there is a check to see if the plugin is being started for the first time. If it is, the method creates a dialog box using the "QDialog" class, and sets its title and size. It also adds various GUI elements to the dialog box, such as buttons, combo boxes, and labels, using the "QComboBox", "QPushButton", and "QLabel" classes respectively.

Next, the method connects each GUI element to a method that will be executed when the element is clicked or when its selection changes. For example, the "QComboBox" elements are connected to the "onModeChanged" method, which is executed when the user selects a different mode of transportation. The "QPushButton" elements are connected to the "onButtonClick" method, which is executed when the user clicks a button.

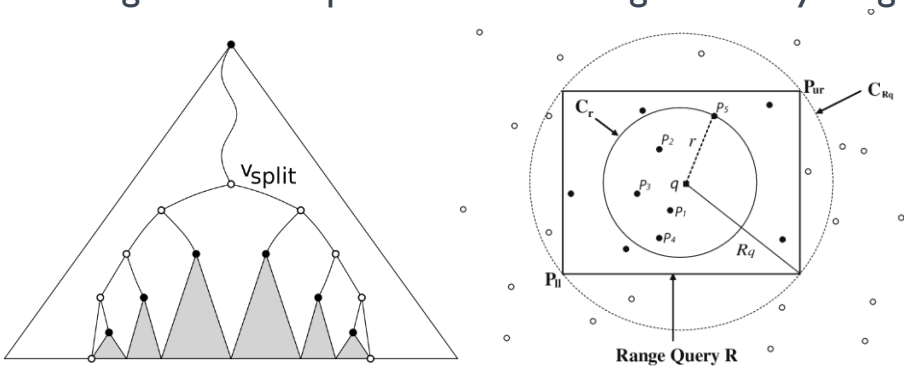
After connecting all the GUI elements, the method sets the initial values of some components, such as the default mode of transportation, and reads some settings that were previously saved if the plugin is not being started for the first time.

Finally, the dialog box is shown using the "exec_" method, which starts the event loop. The event loop listens for user interactions with the GUI elements and executes the connected methods when necessary. If the user clicks the "OK" button while the plugin is already running, the method displays a warning message using the "QMessageBox" class.

Discussion and Conclusions

Recommendations and Improvements

Our research focuses on improving the performance of our plugin, which currently utilizes OTP1 (OpenTripPlanner version 1) and employs an algorithm known as "range queries" for pagination of results. Range queries involve loading the entire data set into memory and determining the appropriate range of results based on the index of the first and last item. While effective for smaller data sets, this approach becomes inefficient for larger data sets, resulting in slow response times and high memory usage.



To address these limitations, we propose implementing API paging, which is the pagination method employed in OTP2. API paging offers a more efficient solution for handling larger data sets. Instead of loading the entire data set, the server retrieves only a portion or "page" of the data in each request, allowing the client to request additional pages as needed. By minimizing the amount of data loaded into memory, API paging significantly improves response times and reduces memory usage.

By incorporating API paging into our plugin, we anticipate substantial improvements in its performance, particularly when processing larger data sets. The plugin will no longer need to load the entire data set into memory but can instead request only the required data for each page. As a result, users can expect faster response times and lower memory usage, enhancing their overall experience. This enhancement is particularly advantageous for applications that rely on our plugin, such as transportation planning and scheduling tools, which frequently handle substantial data sets.

Conclusion

The OpenTripPlannerPlugin is a QGIS plugin that offers valuable functionalities for transportation researchers, urban planners, and smart city developers. It provides users with access to the OpenTripPlanner platform, a well-established open-source tool for multimodal trip itinerary planning and transportation system analysis.

The plugin offers a comprehensive set of routing options for multiple modes of transportation, including walking, cycling, and driving, along with the ability to compute isochrones. It also has sophisticated spatial analysis capabilities that allow users to evaluate accessibility and mobility among different population groups within transportation systems and networks.

The OpenTripPlannerPlugin integrates seamlessly with OpenTripPlanner and has a user-friendly interface and built-in tools for processing data further in QGIS. Key applications of the plugin include public transportation planning, bicycle routing, pedestrian routing, and emergency vehicle routing. Users can learn how to use the plugin effectively through online research and hands-on experience with simple and complex scenarios. Overall, the OpenTripPlannerPlugin is an essential tool for transportation planning and analysis, enhancing accessibility and mobility in different areas and populations.

References

- mkoenigb/OpenTripPlannerPlugin: Repository for OpenTripPlanner Plug-In for QGIS, which lets you access OTP functionalities from within QGIS (github.com)
- OpenTripPlanner Plugin — QGIS Python Plugins Repository
- OpenTripPlanner
- opentripplanner/OpenTripPlanner: An open source multi-modal trip planner (github.com)
- opentripplanner: Setup and connect to 'OpenTripPlanner' (r-project.org)
- (PDF) OpenTripPlanner - creating and querying your own multi-modal route planner (researchgate.net)