EcoWay

A path to sustainability

Marco Cioci
Person code: 10694772

Francesco Rosnati
Person code: 10562427

Luca Simei
Person code: 10714016

Alberto Taddei
Person code: 10733817

April 2024
v1.0.0

POLITECNICO
MILANO 1863


ecoway
A PATH TO SUSTAINABILITY

Contents

1 EcoWay: A path to sustainability

1.1 The Problem

Two pressing global challenges are environmental sustainability and climate change, worsened by transportation-related air pollution and greenhouse gas emissions.
Urban commuting significantly contributes to these issues, with many urban areas still heavily dependent on personal vehicles. Despite being perceived as the most comfortable and efficient form of commuting, numerous studies suggest that better alternatives often exist.
Enhancing the efficiency of public transportation systems can make them more attractive to daily commuters. This improvement not only promises to reduce environmental impacts but also to enhance the overall quality of life for urban residents.

1.2 The Goal

EcoWay plans to change how people travel in cities by making it easier for everyone to use public transportation and sharing vehicles.
It offers optimized route planning and real-time updates about vehicles, ensuring that users can travel efficiently from one location to another. This makes public transport not only convenient but also a fast and reliable option.
Therefore the goal is to encourage more people to use public and shared transportation, resulting in a better solution for our environment.

1.3 Stakeholders

Citizens: *EcoWay* offers a mobile app that allows citizens to input the origin and the destination of their trips within the urban area, together with any constraint they have.

Urban Area Managers: *EcoWay* offers to managers a dashboard through which they can visualize reports concerning the daily usage of the various available transportation means, their occupation rates and delays.

1.4 Main Idea

The primary goal of the *EcoWay* project is to advocate for a more sustainable environment by making eco-sustainable transportation more appealing to urban commuters. *EcoWay* also aims to directly impact the quality of life of urban citizens. Therefore, the ambitions of the project include reducing not only the most polluting and power-consuming vehicles but also those that contribute to unhealthy conditions and traffic congestion. Due to the exhibited reasons *EcoWay* intends to be a tool tailored for the citizens of metropolitan areas and for their Urban Area Managers.

Regular users The service designed for the first cited group revolves around improving the usability of integrating public transport with a selection of sharing service vehicles.
The system guarantees the following benefits:

- Access to real-time data for the entire urban area's timetables, available to any city user for free.
- Calculation of the fastest route from any location to any other, along with suggestions for more ecological alternatives that take approximately the same time. Public transportation will be the only option available to free users. For those seeking additional features from *EcoWay*, a subscription to a Premium Version option will be available to incorporate all available sharing services into their routes.

Urban Area Managers *EcoWay* provides urban area managers (u.a.m.) the possibility to review reports on different aspects of urban commuting through a dashboard. This includes transit conditions, occupancy rates, feedbacks from regular users and many others. Additionally the system will give u.a.m. a way to directly contact the local Urban Transportation Organization and Sharing Companies (via proper Web Services) in order to suggest changes over the mobilty services, with the possibility to attach the reports in such communications. Least but not last the chance to directly change the subscription fares of the Premium Version.

1.4.1 Choice of Sharing Vehicles

Not all sharing vehicles satisfy the goals of *EcoWay*. First and foremost no sharing service with non-electrical engine are part of the selection. The good choice are Bikes, Electric Bikes, and Electric Scooters, this means of transporation are the ones with the least possbile impact on pollution, air quality and power consumption.
Moreover *EcoWay* has intentionally excluded also electric cars and motorcycles from its system. Although these vehicles are more environmentally friendly compared to their fossil-fuel counterparts, they also are very hard to integrate in a unique route comprehensive of both sharing and traditional vehicles. This would contradict the aim of making public transportation the most functional and appealing way to commute for citizens.

1.4.2 Subscription System & Tickets

To achieve the goal of pushing for metropolitan areas free of pollution, noise and traffic *EcoWay* Premium Version allows users to use for their routes both sharing and public vehicles with a fixed fare for a given period.
The division in daily, weekly, monthly and annual plans welcomes a various range of different users: from tourists lodging for a single day, to citizens living the city all year long. A pricing strategy that is sufficiently affordable, combined with an extremely efficient service, could encourage users to completely forgo their own car. Unsubscribed users are not allowed to use sharing vehicles in their route computation, they are offered at each route the possibility to buy a ticket from the local public transport portal. The limitation for unsubscribed users is designed to encourage subscriptions by highlighting the convenience and additional benefits of access to sharing vehicles.

1.4.3 Green Routes Calculation

The main functionality of *EcoWay* is the computation of the shortest route from a given starting position to a desired destination. However this first route is only one possibility given to the user, used merely as a benchmark. A distinctive feature of *EcoWay* approach is to provide up to nine green alternatives. These alternatives are taught to be still efficient: the travel time never differs from the shortest route by more than 20%. On the other hand these routes are realized to minimize environmental impact, utilizing a classification system for each type of transportation based on its eco-friendliness (table below). EcoWay prioritize the routes where the distance travelled on low environmental imapact classes of vehicles are maximized.

	Traditional Transports	**Sharing Vehicles**
Class 0	Bike, Walking	Bike
Class 1	Electric Bike	Electric Bike, Electric Scooter
Class 2	Eco Bus, Tram, Subway	-
Class 3	Trains, Thermal Buses	-

Table 1: Classification of transportation modes for Traditional and Sharing transports

1.4.4 External Services

The system depends on the integration with pre-existing external services, which facilitate data and permission exchanges with *EcoWay* across various execution stages.
Among these, the Urban Transportation Web Service serves as complete interface for the system with other important actors such as the Ticket Vendor Service, Urban Transportation Portal, and Sensors Service. The Sensors Service enables *EcoWay* to access pre-processed data concerning urban mobility and transportation occupancy rates. EcoWay has no direct access to the sensors nor to the others.
Another critical service on which *EcoWay* relies is the Database Service. On one hand this component allows EcoWay to store and retrieve pertinent information in real-time, both for immediate use in route or timetable computation, but also to visualise report already processed of the gathered data. On the other hand *EcoWay* massively relies on this service in ancillary computation, necessary for the correct functioning of his own components.

Urban Area Managers utilize the services above to analyze and revise existing routes, timetables, occupancy rates and many more on the already computed reports.
Last example given in this section: users directly interact only with an external app, which on its turn uses an API gateway to communicate with the system.

Further characterizations of external services are presented in *Section 2.2.2: Secondary Actors*

2 Requirements Analysis

2.1 Domain Model: a first diagram

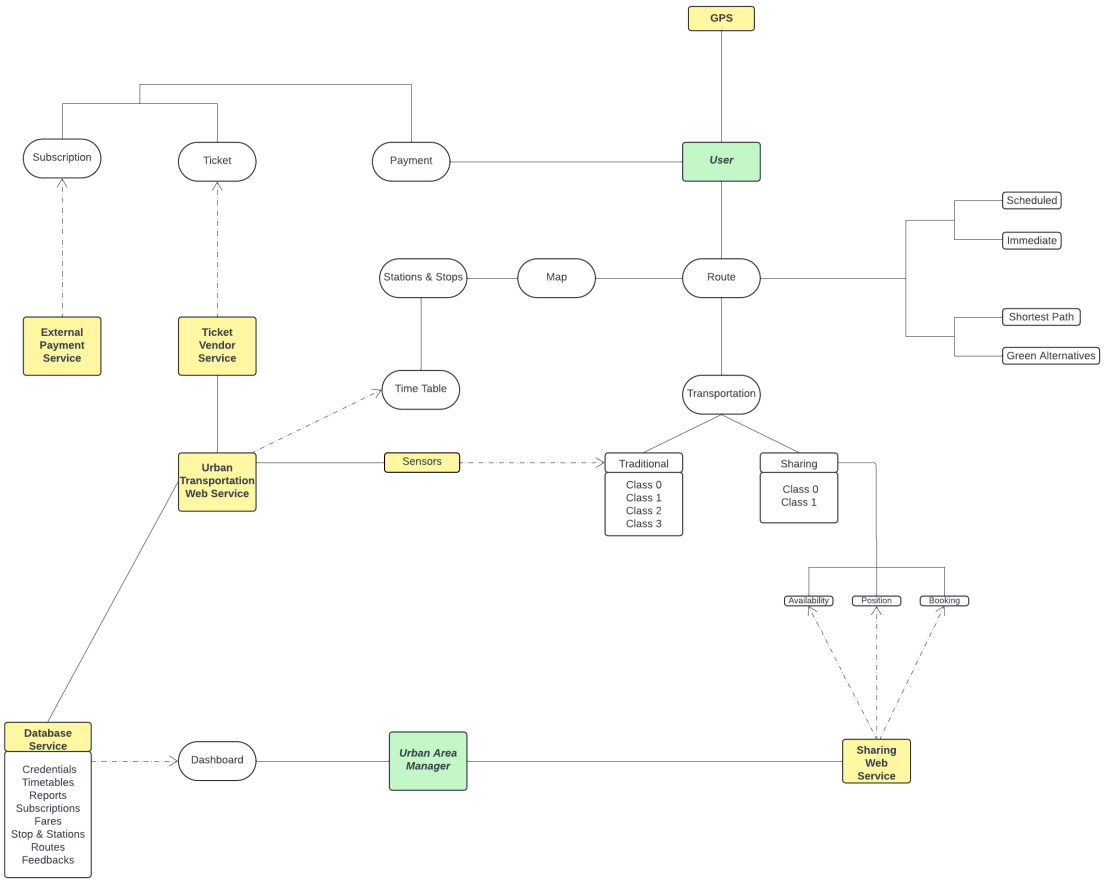


Figure 1: in green boxes, primary actors. In yellow boxes, secondary actors. Dotted lines represents connections from the system with external services.

2.2 Actors & Terminology

2.2.1 Primary Actors

The only actors who can initiate the use of the system, furthermore they can also partecipate.

Regular User: Individuals who interact with the *EcoWay* system to plan their daily commutes using public and shared transportation. Users utilize the system to access real-time transit data, calculate routes and integrate various modes of transport to optimize their travel in a more sustainable way.

Urban Area Manager: City officials who use *EcoWay* to monitor and manage urban transportation networks. They access system-generated data on vehicle occupancy, service usage and commuter behavior to make informed decisions about transit policies, service improvements and infrastructure investments.

Both regular users and urban area managers are *human actors.*

2.2.2 Secondary Actors

They can only partecipate in use cases started by a *primary actor.*

Database Service: A central repository that *EcoWay* relies on for storing, computing and later retrieving all operational data including user profiles, route information and historical transit usage. This service ensures data integrity, security and availability to support real-time application needs.

External Payment Service: A financial transaction service that integrates with *EcoWay* to handle subscriptions and other fee-based interactions. It enables secure processing of user payments for subscribing to enhanced features within the system.

Urban Transportation Web Service: This web service aggregates multiple functionalities including real-time data feeds from public transit systems (*sensors*), ticket purchasing capabilities (*ticket vendor service*) and administrative interactions (*urban transportation portal*). It is crucial for updating *EcoWay* with current transit statuses, facilitating ticket sales and enabling administrative oversight.

Sensors: Embedded within the urban transportation infrastructure, these devices provide vital data regarding vehicle statuses, traffic conditions and environmental parameters. This data is integrated into *EcoWay* to enhance route planning algorithms and improve the accuracy of transit schedules and predictions. They also contribute to gather part of the data later accessible by u.a.m. on dahboard.

Ticket Vendor Service: Specializes in the sale of tickets for various traditional transportation modes like buses, trains and trams. Connected with *EcoWay*, it allows users to purchase transit tickets directly through the interface, enhancing user convenience.

Urban Transportation Portal: Acts as the administrative link for urban area managers, allowing them to manage and oversee the operations of public transit systems. This portal provides essential tools for modifying transit routes, schedules and managing the overall transit service infrastructure.

Sharing Web Service: Connects *EcoWay* with various shared transportation services like bike-sharing and ride-sharing. This integration allows users to access a broader range of transportation options, making their travel more flexible and environmentally friendly.

Urban Area Manager Portal: A secure platform that authenticates and authorizes urban area managers, providing them with access to *EcoWay*'s administrative features. This portal facilitates the management of user permissions, system settings and access to sensitive data analytics.

GPS: A critical component for enabling location-based services in EcoWay. It allows the app to determine real-time user locations, suggest optimal routes and provide navigation assistance. It enhances the system's capability to offer tailored transportation solutions based on the user's current location.

Eventually all *secondary actors* result to be *non-human actors*

2.2.3 Terminology

***EcoWay*:** An integrated software system designed to optimize urban transportation by facilitating user access to public and shared transport options, managing subscriptions, and providing real-time scheduling and route planning through a user-friendly app interface.

App and API Gateway: A mobile application acting as the front-end for users, interfacing with the EcoWay system through an API gateway.

Login Interface: A user portal for entering authentication details to access their EcoWay account.

Registration Interface: A portal for new users to sign up and create a new *EcoWay* account.

Change Default Transportations Interface: A user preference area to update selected transportation modes within *EcoWay*.

User Interface: The primary navigation area of the app where users access different transportation services.

Subscription Plan Interface: An interface for users to select or change their *EcoWay* service subscription tier.

Compute Route Interface: A feature allowing users to input travel details and receive proposed route options.

Route: A dynamically generated pathway for user travel, composed of various transport options and segments.

Ticket: An authorized credential for access to traditional public transport services.

Timetable: A compilation of scheduled times for public transport arrivals and departures at various stops.

Stop/Station: Specific locations where public transport vehicles consistently pick up and drop off passengers.

2.3 Scenarios

In the following scenarios *actors* are in *italics* in order to be more easily distinguishable

Unregistered *regular user* registers

1. *Regular user* uses app to interface with *EcoWay*
2. *EcoWay* requires the regular user to be logged in
3. *EcoWay* presents the login interface
4. Login interface presents boxes to enter credentials, or links to the registration interface
5. *Regular user* selects the registration interface
6. Registration interface asks if *regular user* wants to register as a "regular user" or "urban area manager"
7. *Regular user* chooses the "regular user" option
8. *Regular user* enters registration data (login credentials), if not valid (e.g. regular username already exists) they are asked for new credentials
9. Registration data is sent to the *database service*
10. *EcoWay* generates the account: login credentials (regular user ID and password) are saved in the *database service* in regular user data section; subscriber status set to "non-subscribed", traditional transportation set to "active", sharing transportations set to "inactive"
11. *EcoWay* presents the login interface

Registered *regular user* logs in

1. *Regular user* uses app to interface with *EcoWay*
2. *EcoWay* requires the *regular user* to be logged in
3. *EcoWay* presents the login interface
4. Login interface presents boxes to enter credentials, or links to the registration interface
5. *Regular user* enters login credentials
6. *EcoWay* checks from the *database service* that login credentials are correct. If they are incorrect, *regular user* is asked to enter them again
7. If login credentials are correct, *EcoWay* presents the regular user interface

Logged in *regular user* buys tickets

1. *EcoWay* presents the regular user interface
2. Regular user interface presents 5 options: "Get Route", "Change Default", "Transportations", "Get Timetable", "Manage Subscription", "Buy Ticket"
3. *regular user* selects "Buy Ticket"
4. *EcoWay* shows a pop-up that asks *regular user* if he wants to be redirected to Ticket Vendor Service
5. *regular user* can close the dropdown with an "X" or press "Continue"
6. If *regular user* presses "Continue", they are redirected to *ticket vendor service* via urban transportation Web service interface
7. *EcoWay* presents the regular user interface whether *regular user* presses "X" or "Continue"

***Regular user* changes default transportations**

1. *EcoWay* presents the regular user interface
2. Regular user interface presents 5 options: "Get Route", "Change Default Transportations", "Get Timetable", "Manage Subscription", "Buy Ticket"
3. *Regular user* selects "Change Default Transportations"
4. *EcoWay* retrieves from the *database service*, in regular user data section, the default transportations and subscription status
5. *EcoWay* presents the change default transportations interface, which shows options based on the regular user subscription status
6. *Regular user* changes preferences, selecting vehicle preferences as desired, and confirms
7. *EcoWay* updates regular user data regarding default transportations on the *database service* side
8. *EcoWay* presents the regular user interface

***Regular user* views timetable for a stop/station**

1. *EcoWay* presents regular user interface
2. Regular user interface presents 5 options: "Get Route", "Change Default Transportations", "Get Timetable", "Manage Subscription", "Buy Ticket"
3. *Regular user* selects "Get Timetable"
4. *EcoWay* accesses the *database service*, maps section
5. *EcoWay* displays map to select the desired stop/station
6. *Regular user* selects stop/station
7. *EcoWay* retrieves from the *database service*, timetable section, the information related to that stop/station
8. *EcoWay* connects to urban transportation Web service and retrieves real-time timetable and vehicle status
9. *EcoWay* updates, from the data just retrieved, the timetable
10. *EcoWay* returns, for each available vehicle at that stop/station, the updated timetable
11. *EcoWay* presents the timetable interface for the selected stop/station, which displays, for each vehicle, the minutes until the next arrival and the timetable

Subscribed *Regular user* clicks on "Subscription"

1. *EcoWay* presents the regular user interface
2. Regular user interface presents 5 options: "Get Route", "Change Default Transportations", "Get Timetable", "Manage Subscription", "Buy Ticket"
3. *Regular user* selects "Manage Subscription"
4. *EcoWay* retrieves from the *database service*, in regular user data section, the subscription status, which in this case is set to "ON" in the *database service*
5. *EcoWay* returns to the *regular user* the subscription interface, which in this case shows that the subscription already exists

Non-subscribed *regular user* clicks on "Subscription"

1. *EcoWay* presents the regular user interface
2. Regular user interface presents 5 options: "Get Route", "Change Default Transportations", "Get Timetable", "Manage Subscription", "Buy Ticket"
3. *Regular user* selects "Manage Subscription"
4. *EcoWay* retrieves from the *database service*, regular user data section, the subscription status, which in this case is set "OFF" in the *database service*
5. *EcoWay* retrieves the subscription fares from the *database service*
6. *EcoWay* presents the subscription fares through the subscription plan interface
7. *Regular user* selects the desired rate
8. *EcoWay* redirects to *external payment service*
9. *EcoWay* waits for confirmation of successful or unsuccessful payment from *external payment service*:
    - if the payment fails, *EcoWay* returns the subscription plan interface, displaying an error message
    - if the payment succeeds, *EcoWay* updates the subscription status to "ON" in the regular user data section of the *database service*, and finally redirects to the subscription plan interface

Subscribed/non-subscribed *regular user* requests scheduled route

1. *EcoWay* presents the regular user interface
2. Regular user interface presents 5 options: "Get Route", "Change Default Transportations", "Get Timetable", "Manage Subscription", "Buy Ticket"
3. *Regular user* selects "Get Route"
4. *EcoWay* accesses the *database service*, maps section
5. *EcoWay* presents the compute route interface, where the *regular user* can input departure and arrival locations, with "Immediate departure" being the default option. Alternatively, the *regular user* can choose the "Scheduled Route" option by setting departure time or arrival time
6. *Regular user* inputs departure and arrival locations from those allowed by the *database service* (dropdown menu)
7. *EcoWay* updates the dropdown menu at each letter typed and displays suggested completion to the *regular user*
8. *Regular user* chooses departure time or arrival time
9. *EcoWay* retrieves from the *database service*, regular user data section, default transportations
10. *EcoWay* selects for the route, from the default transportations, only traditional transportations; all sharing transportations are not considered
11. *EcoWay* accesses the timetable section of the *database service*
12. *EcoWay* remains connected to the *database service*, maps section
13. *EcoWay* performs the route calculation, with the selected vehicles, without considering real-time data on vehicles, using the previously indicated locations and times
14. Shortest path is calculated

15. Up to 9 alternative green routes are calculated, using the shortest path as a benchmark; the commute time of the alternatives can never take 20% more than the time of the shortest path. For each transportation (both sharing and traditional), there is a classification, based on how sustainable they are considered to be:

    - Class 0: Bike; Walking
    - Class 1: Electric Bike; Electric Scooter
    - Class 2: Eco Bus; Tram; Subway
    - Class 3: Trains; Thermal Buses

16. The alternative routes are calculated to optimize the distance covered using the most environmentally sustainable transportation options available
17. *EcoWay* displays route. For each alternative it shows: duration, sustainability level, vehicles used
18. *Regular user* selects the preferred route
19. *EcoWay* displays the route on the map
20. *Regular user* selects "Exit From Route"

Non-subscribed *regular user* requests immediate route

1. *EcoWay* presents the regular user interface
2. Regular user interface presents 5 options: "Get Route", "Change Default Transportations", "Get Timetable", "Manage Subscription", "Buy Ticket"
3. *Regular user* selects "Get Route"
4. *EcoWay* accesses the *database service*, maps section
5. *EcoWay* connects to *GPS*
6. *EcoWay* presents the compute route interface, where the *regular user* can input departure and arrival locations, with "Immediate Departure" being the default option
7. *Regular user* inputs departure and arrival locations from those allowed by the *database service* (dropdown menu). By default, the starting position is the current GPS location
8. *EcoWay* has an " Address Suggestion" feature, that updates the dropdown menu with each letter typed and displays it to the *regular user*
9. *EcoWay* retrieves from the *database service*, regular user data section, default transportations, which includes only traditional vehicles.
10. *EcoWay* selects for the route the retrieved transport vehicles.
11. *EcoWay* accesses the timetable section of the *database service*
12. *EcoWay* remains connected to the *database service*, maps section
13. *EcoWay* connects to the *urban transportation Web service* to obtain real-time data for the computation he is about to make
14. *EcoWay* performs the route calculation, with the selected vehicles, without considering real-time data on vehicles, using the previously indicated locations and times
15. Up to 9 alternative green routes are calculated, using the shortest path as a benchmark; the commute time of the alternatives can never take 20% more than the time of the shortest path. For each transportation (both sharing and traditional), there is a classification, based on how sustainable they are considered to be:

    - Class 0: Bike; Walking

- Class 1: Electric Bike; Electric Scooter
- Class 2: Eco Bus; Tram; Subway
- Class 3: Trains; Thermal Buses

16. The alternative routes are calculated to optimize the distance covered using the most environmentally sustainable transportation options available.
17. *EcoWay* displays route, for each alternative it shows: duration, sustainability level, vehicles used
18. *Regular user* selects the preferred route
19. *EcoWay* displays the route on the route interface; the route is shown on the map
20. *EcoWay* marks in the regular user data section of the *database service* if the route is active for that *regular user*
21. *EcoWay* (without removing the route) offers the *regular user* the option to purchase an urban transportation ticket
22. If the *regular user* accepts to purchase the ticket: *regular user* is directed to *transportation ticket vendor service* through the *urban transportation Web service*, an external service to the app, and the route remains displayed on the app
23. *EcoWay* checks the *regular user*'s position through the GPS: there is a timer that periodically checks it in parallel with all regular users currently on a route; there is a table of active regular users
24. *EcoWay* exits the route interface in the following cases:

    - GPS location of the *regular user* matches the arrival destination
    - *Regular user* selects "Exit From The Route"

25. Upon completion of the route according to exit cases mentioned above, *EcoWay* marks in the regular user data: active route to "false", sharing transportations to "ID" to "NaN", and booking time to "NaN"
26. Upon completion of the route with any exit cases, the *regular user* is asked to leave feedback
27. At the same time, *EcoWay* saves the *regular user*'s route in the itinerary section of the *database service* (saving happens even if the route is interrupted without reaching the destination)

Subscribed *regular user* requests immediate route

1. *EcoWay* presents the regular user interface
2. Regular user interface presents 5 options: "Get Route", "Change Default Transportations", "Get Timetable", "Manage Subscription", "Buy Ticket"
3. *Regular user* selects "Get Route"
4. *EcoWay* accesses the *database service*, maps section
5. *EcoWay* connects to *GPS*
6. *EcoWay* presents the compute route interface, where the *regular user* can input departure and arrival locations, with "Immediate Departure" being the default option
7. *Regular user* inputs departure and arrival locations from those allowed by the *database service* (dropdown menu). By default, the starting position is the current GPS location
8. *EcoWay* has an " Address Suggestion" feature, that updates the dropdown menu with each letter typed and displays it to the *regular user*

9. *EcoWay* retrieves from the *database service*, regular user data section, default transportations, which may include both traditional and sharing vehicles
10. *EcoWay* selects for the route the retrieved transport vehicles.
11. *EcoWay* accesses the timetable section of the *database service*
12. *EcoWay* remains connected to the *database service*, maps section
13. *EcoWay* connects to the *urban transportation Web service* to obtain real-time data for the computation he is about to make
14. *EcoWay* performs the route calculation, with the selected vehicles, using the previously indicated locations and times
15. Up to 9 alternative green routes are calculated, using the shortest path as a benchmark; the commute time of the alternatives can never take 20% more than the time of the shortest path. For each transportation (both sharing and traditional), there is a classification, based on how sustainable they are considered to be:

    - Class 0: Bike; Walking
    - Class 1: Electric Bike; Electric Scooter
    - Class 2: Eco Bus; Tram; Subway
    - Class 3: Trains; Thermal Buses

16. The alternative routes are calculated to optimize the distance covered using the most environmentally sustainable transportation options available
17. *EcoWay* displays route, for each alternative it shows: duration, sustainability level, vehicles used
18. *Regular user* selects the preferred route
19. *EcoWay* displays the route on the route interface; the route is shown on the map
20. *EcoWay* marks in the regular user data section of the *database service* if the route is active for that *regular user*
21. *EcoWay* connects to the Sharing Web Service to book sharing transportations
22. Sharing Web Service sends response:

    - In case of error, it sends an error message, and the *user* is redirected to the Sharing Web Service interface
    - In case of success, Sharing Web Service books the sharing vehicle, sends a Vehicle Id to *EcoWay* for unlocking via QR-code scan

23. *EcoWay* saves Vehicle Id and Booking Time in user data section of Database Service
24. *EcoWay* periodically checks *user*'s position through GPS
25. *User* unlocks sharing vehicle via QR-code scan and starts usage
26. *EcoWay* counts usage time and connects to Sharing Web Service for GPS sensor data
27. *EcoWay* periodically checks user and vehicle distance
28. *EcoWay* exits route interface if:

    - User's GPS matches Arrival Destination
    - *User* selects "Exit from the route"

29. *EcoWay* marks route as inactive in Database Service upon completion or exit
30. *User* is prompted for feedback
31. Route is saved in itinerary section of Database Service (saving happens even if the route is interrupted without reaching the destination)

Unregistered *urban area manager* registers

1. *Urban area manager* uses the app to interface with *EcoWay*
2. *EcoWay* requires the *urban area manager* to be logged in
3. *EcoWay* presents the login interface
4. Login interface presents boxes to enter login credentials or a link to the registration interface
5. *Urban area manager* selects the registration interface
6. Registration interface asks if *urban area manager* wants to register as a regular user or *urban area manager*
7. *Urban area manager* selects the urban area manager option
8. *EcoWay* redirects the *urban area manager* to the *urban area manager portal* to authenticate, while *EcoWay* waits for the response
9. This response (token) can be "Positive" or "Negative":
    - If the response is negative, it displays an error message and redirects to the registration interface
    - If the response is positive, the *urban area manager portal* sends the access token
10. *Urban area manager* enters registration data for *EcoWay* (login credentials), if not valid (e.g. regular username already exists) they are asked for new credentials
11. *EcoWay* generates the account and saves the login credentials in the *database service*, *u*rban area manager data section
12. *EcoWay* presents the login interface

Registered *urban area manager* logs in

1. *Urban area manager* uses the app to interface with *EcoWay*
2. *EcoWay* requires the *urban area manager* to be logged in
3. *EcoWay* presents the login interface
4. Login interface provides boxes to enter the credentials or a link to the registration interface
5. *Urban area manager* enters the login credentials
6. *EcoWay* checks from the *database service*, urban area manager data section, if the credentials are correct (regular user ID and password)
7. If they are not correct, they are asked to enter them again
8. If the credentials are correct, *EcoWay* presents the urban area manager interface

***Urban area manager* views the dashboard**

1. *EcoWay* presents the urban area manager interface
2. Urban area manager interface features 3 options: "Get dashboard", "Contact Mobility Operators", "Change Subscription Fares"
3. *Urban area manager* selects "Get Dashboard"
4. *EcoWay* displays the dashboard. Dashboard includes two dropdown menus: one to select the date and another to select the type of report (disruptions, feedback, vehicle occupancy, routes); the central body displays the selected report

5. *EcoWay* retrieves data related to the selected report from the database service, itinerary section, and displays it. *EcoWay* doesn't need to process the data to create report, because the *database service* pre-computes all the data and generate the reports, ready to be send to the front-end of the dashboard.
6. *Urban area manager* can navigate back to the dashboard at any time using a dedicated "Back" button

Urban area manager* suggests modifications to *urban transportation and sharing services

1. *EcoWay* presents the urban area manager interface
2. Urban area manager interface features 3 options: "Get Dashboard", "Contact Mobility Operators", "Change Subscription Fares"
3. *Urban area manager* selects "Contact Mobility Operators"
4. EcoWay displays the contacts interface, which shows 3 dropdown menus: "Company Contact", "Report Date", "Report Type", and a central panel with a text box for the message to be sent and a "Send" button
5. *Urban area manager* fills in the various fields and clicks on the "Send" button
6. EcoWay continuously accesses the *database service* throughout the entire previous operation to suggest entries for the 3 dropdown menus
7. EcoWay redirects the *urban area manager* to the urban area manager interface
8. *Urban area manager* can navigate back to the dashboard at any time using a dedicated "Back" button

***Urban area manager* changes the subscription fares**

1. *EcoWay* presents urban area manager interface
2. Urban area manager interface features 3 options: "Get Dashboard", "Contact Mobility Operators", "Change Subscription Fares"
3. *Urban area manager* selects "Change Subscription Fares"
4. *EcoWay* displays the subscription fares modification interface, where for each type of subscription it's possible to modify the regular user's subscription fares in a textbox
5. *EcoWay* saves the changes within the *database service* in the data regular user section
6. *Urban area manager* can navigate back to the u.a.m. interface at any time using a dedicated "Back" button

2.4 Use Cases

2.4.1 Use Cases Diagram

Conventions

- Primary actors are always shown on the left of the system, while secondary are on the right.
- All the following use cases are part of a unique EcoWay system, for clarity and graphical reason they are divided by functionality: at first the ones relating to authentication, then there are separated diagrams for regular user related use cases and urban area manager ones.

Authentication Use Case

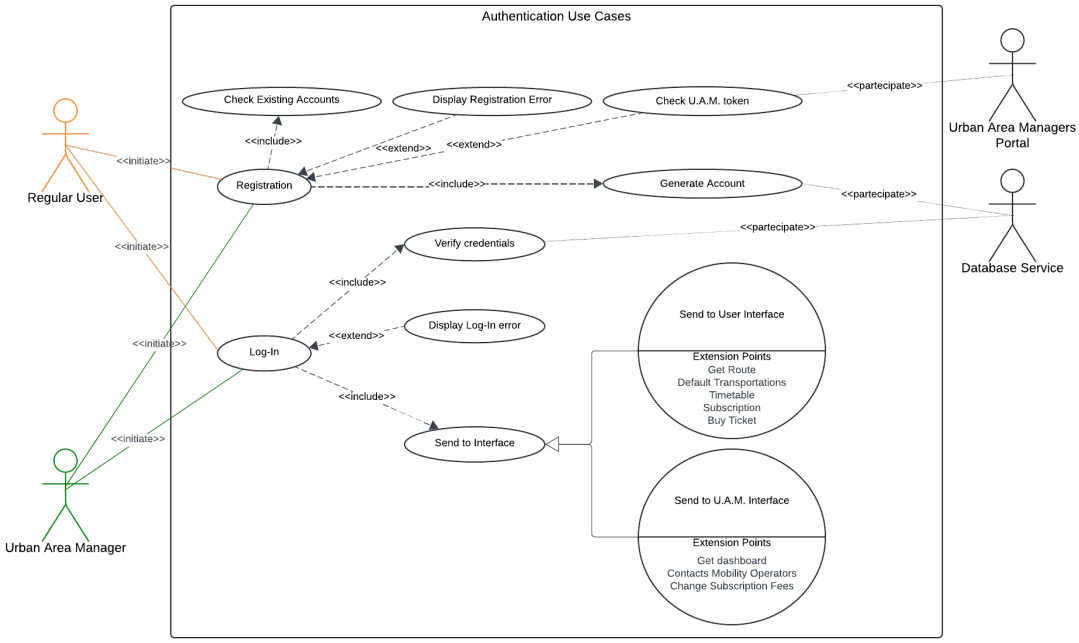


Figure 2: the use cases shown are related to authentication process only. After correct registration and login, the user is brought to the proper interface. Functionalities of such interfaces are shown in the following paragraphs.

Regular User Use Case

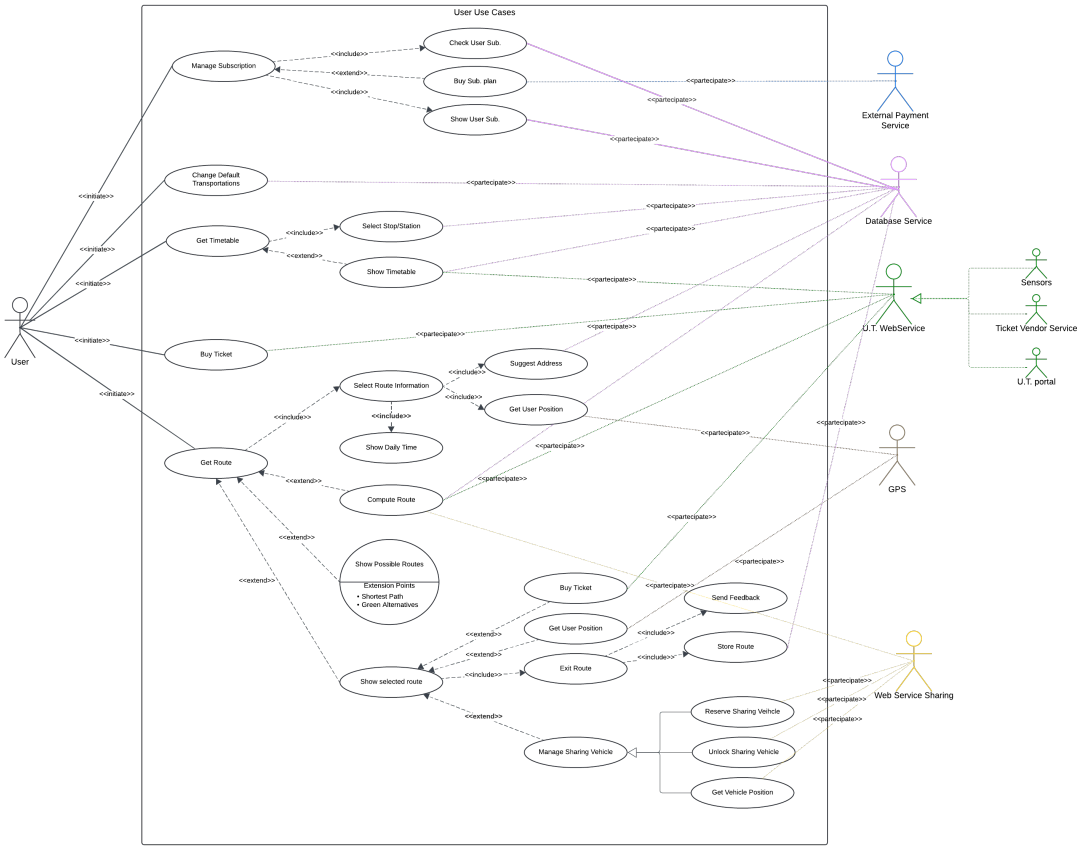


Figure 3: the diagram shows possible interactions for a regular user after authentication and all related background use cases necessary for the system to accomplish the request. The system strongly relies on communication with external services.

Urban Area Manager Use Case

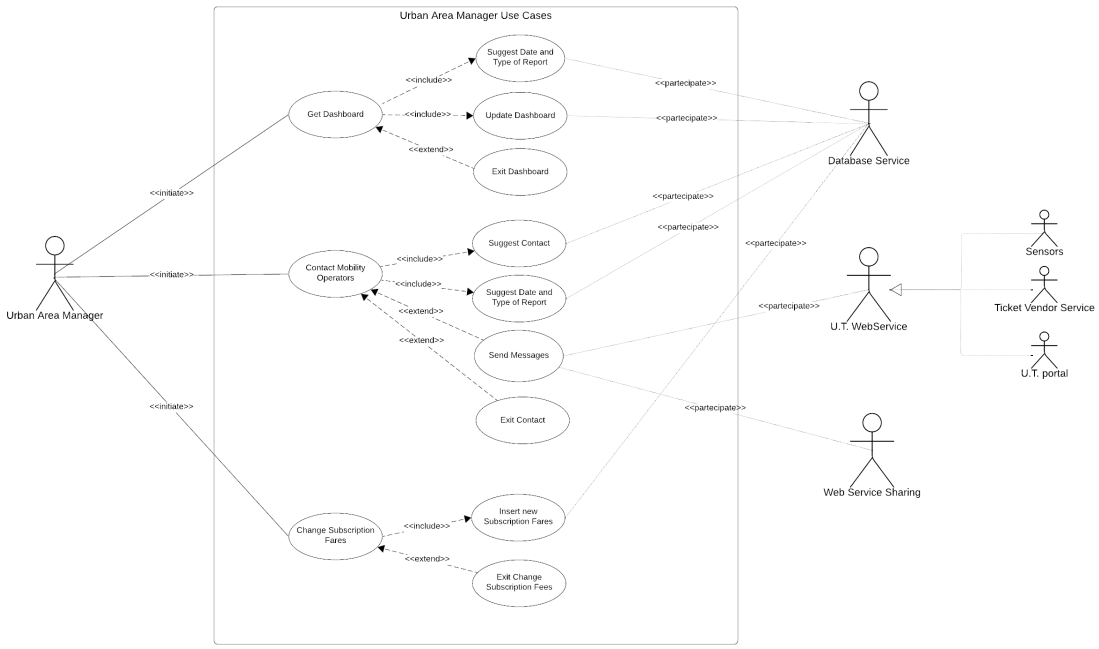


Figure 4: the diagram presents a focused view of the *urban area manager* interaction with the *EcoWay* system, in post authentication phase.

2.4.2 Use Cases Descriptions

A thorough description of all the possible actions available to the users of the *EcoWay* system are already provided in the scenarios section, which was previously presented.
Nonetheless, a more rigorous use case description can be given, in order to explicitly show for each functionality of *EcoWay* the following:

- proper name
- entry condition
- flow of events
- exit condition
- exceptions
- special requirements

The current section provides examples of such description, in particular a complete representation of the authenitication phase is presented, along with a selection of functionalities from the both the user and urban area manager functionalities.

Name	**Any User Login**
Participating Actors	Any user: regular user/u.a.m, Database Service
Entry Condition	True
Flow of Events	1. Any user accesses the EcoWay system through the App that shows the login interface 2. If the user is not registered, he is able to create a new account by clicking the proper option, the system executes the Registration use case 3. The system asks the user to insert his credentials 4. The user inserts his credentials: userID, password 5. Once received such information the system sends it to the database service requiring a validity check 6. The database service checks the credentials and sends an answer to the system
Exit Condition	The user is re-routed to the proper interface, based on the user type (regular or u.a.m.)
Exceptions	• If the credentials inserted are not valid, the database sends an error message to the system which displays an error message asking to insert again • If the connection with the database fails, the system displays an error message asking to insert again
Special Requirements	If the time of connection with the database plus the credentials check lasts more than 3 seconds, the system considers it a failed connection

Name	**Any User Registration**
Participating Actors	Regular user/u.a.m., Database Service
Entry Condition	An user of any type has selected registration; Failed previous attempt to register
Flow of Events	1. The system asks the user if he wants to register as a regular user or as a u.a.m. 2. If the user selects u.a.m. the system executes the check u.a.m token use case 3. The system asks the user to choose and insert userID, password (credentials) 4. Once received such information the system sends it to the database service requiring a validity check 5. The database service checks the credentials 6. The database service creates an account of the proper type (regular or u.a.m. access) based on initial choice 7. The database service sends an answer to the system 8. The system displays a message of correct registration
Exit Condition	The system executes the login use case; the user is now ready to log in
Exceptions	• If the user chooses to register as a u.a.m., this use case is interleaved to retrieve the u.a.m. token as described in the specific use case; once obtained, the token will resume normally. • If the retrieval of the u.a.m. token fails, the user is asked again to select if he wants to register as a regular user or as a u.a.m. • If the UserID chosen by the user already exists, the database sends an error message to the system which displays an error message asking to insert again • If the connection with the database fails, the system displays an error message asking to insert again
Special Requirements	If the time of connection with the database plus the credentials check lasts more than 3 seconds, the system considers it a failed connection

Name	**Check u.a.m. token**
Participating Actors	u.a.m, u.a.m. portal
Entry Condition	A user during registration selected to register as a u.a.m.
Flow of Events	1. The system re-routes the u.a.m. to the u.a.m. portal 2. The u.a.m. logs in the portal 3. The u.a.m. portal sends back the token to the system
Exit Case	The system correctly received the token and is ready to proceed with u.a.m. registration
Exceptions	• If the connection to the u.a.m. portal fails or the u.a.m. fails to log in, this use case quits and the system executes the use case registration
Special Requirement	After 3 minutes from the re-routing, the system will consider the connection to the u.a.m. portal failed anyway

Name	**Change Default Transportation**
Participating Actors	Regular User, Database Service
Entry Condition	The regular user is logged into the EcoWay system. The user has selected "Change Default Transportation" in the user interface.
Flow of Events	1. The system retrieves the current default transportation settings from the Database Service. 2. The system presents the default transportation options available (e.g., bus, tram, bike) to the user along with their current settings. 3. The user selects new default transportation preferences. 4. The system sends these new preferences to the Database Service to update the user's settings. 5. The Database Service confirms the update and notifies the system. 6. The system displays a confirmation message to the user that the default transportation settings have been updated.
Exit Condition	The user's default transportation preferences are successfully updated and confirmed by the system. The user is sent back to the user interface.
Exceptions	• If the Database Service fails to update the preferences due to a server error or connectivity issue, the system displays an error message asking the user to try again later. • If the user attempts to select a transportation mode that is not available for their level of subscription, the system informs the user and asks if they want to subscribe to a premium version. In such a case, the system executes the Manage Subscription use case; otherwise, the user is free to continue setting their choices of transportation.
Special Requirements	The transaction of the system with the Database Service for updating preferences should complete within 2 seconds; if it takes longer, the system will consider it failed due to connectivity issues.

Name	**Get Timetable**
Participating Actors	Regular User, Database Service, Urban Transportation Web Service
Entry Condition	The regular user is logged into the EcoWay system. The user has navigated to the option "get timetable" in the user interface.
Flow of Events	1. The system retrieves information about the map from the Database Service. 2. The system displays a map interface, prompting the user to select a specific stop or station. 3. The user selects the desired stop or station from the map. 4. The system retrieves the standard timetable for the selected stop from the Database Service. 5. Simultaneously, the system requests real-time data for the selected stop from the Urban Transportation Web Service. 6. The system combines the static timetable data with the real-time data to display a comprehensive timetable for the user.
Exit Condition	The user is presented with a detailed timetable, including real-time updates for all transports at the selected stop.
Exceptions	• If the Database Service fails to retrieve the timetable due to a server error or connectivity issue, the system displays an error message asking the user to try again later. • If the Urban Transportation Web Service fails to provide real-time updates, the system displays only the static timetable.
Special Requirements	The real-time integration with the Urban Transportation Web Service should be completed within 5 seconds to ensure timely information delivery. The system must ensure accurate mapping of stops and stations on the user interface to prevent any selection errors.

Name	**Contact Mobility Operators**
Participating Actors	u.a.m., Database Service
Entry Condition	The u.a.m. is logged into the EcoWay system. The u.a.m. has selected "Contact Mobility Operators" in the u.a.m. interface.
Flow of Events	1. EcoWay displays the contacts interface, which includes three dropdown menus for selecting the company contact, report date, and report type, along with a central panel containing a text box for the message and a "Send" button. 2. The u.a.m. has the option to return to the Dashboard at any time during the process using a dedicated "Back" button. 3. The system suggests entries for the dropdown menus by continuously accessing the Database Service to ensure the most up-to-date and relevant options are available. 4. The u.a.m. fills in the required fields in the dropdown menus and types a message in the text box. 5. The u.a.m. clicks the "Send" button to submit the message to the selected mobility operator. 6. After sending the message, EcoWay redirects the u.a.m. back to the main u.a.m. interface.
Exit Condition	The message has been successfully sent to the mobility operator, and the u.a.m. is redirected back to the u.a.m. interface.
Exceptions	• If there is a failure in retrieving data from the Database Service for the dropdown menus, the system displays an error message indicating the issue. • If the message fails to send due to connectivity or service issues, the system informs the u.a.m. of the failure and offers an option to retry.
Special Requirements	The interaction with the Database Service for retrieving dropdown menu options should be quick and efficient, ensuring a response time of no more than 2 seconds to maintain system performance. The system must ensure that all communication links with the mobility operators are secure to protect the integrity and confidentiality of the transmitted data.

2.5 Requirements

2.5.1 Functional Requirements

User registration

1. Any unregistered user must be able to access the app to interface with *EcoWay*.
2. The system shall allow to register as either a "Regular User" or "Urban Area Manager".
3. The systems shall allow unregistred urban area managers to access the interface of u.a.m. Portal to verify their identity.
4. The system shall be able to interleave the registration process to wait for the token or an error from the u.a.m. Portal.
5. The system shall be able to correctly communicate the credentials to the Database Service.
6. The system shall be able to ask the user to insert again if the answer from Database Service was an error.
7. The system shall be able to show a message of correct regitration if Database Service answer was positive.

User login

8. The system shall be able to correctly send information to the Database Service to be verified.
9. *EcoWay* shall be able to display an error and ask to insert credentials again, if the Database Service answer that credentials are wrong.
10. *EcoWay* shall be able to display an error and ask to insert credentials again, if connection with database fails.
11. The sytem must be able to correctly understand the user type (regular or u.a.m.).
12. The system must correctly re-route the user to the proper interface based on the user type (regular or u.a.m.).

Route Computation

13. The system shall provide a user interface that allows regular users to input their origin and destination for travel.
14. The system shall be able to compute the route from origin to destination.
15. The system allows to select origin and destination addresses only from the ones provided by the Database Services.
16. The system shall be able to retrieve user position from the GPS.

17. The system shall use the current position of the regular user as a default starting point.
18. The systems shall allow the user to choose its starting point.
19. The system shall be able to compute a route starting immediately.
20. The system shall be able to compute scheduled routes, with the choice of arrival time or departure time alternatively.
21. The systems shall allow the user to choose the departure time if he wants to schedule a route.
22. Alternatively, the system shall allow the user to choose the arrival time if he wants to schedule a route.
23. Regular users should be able to choose what vehicle they want to use.
24. The system shall use in the computation only vehicles that the specific user has selected.
25. The system shall use real-time data only for immediate routes.
26. The system must be able to acquire real-time data from Urban Transportation Web Service.
27. The system shall be able to ask to the Database Service for the user default vehicles and for the timetable.
28. The system shall compute the route using only vehicles actually available for the specific level of user subscription.
29. The system shall compute multiple routes for a single user.
30. The system shall access the interface of the Sharing Web Service when a user wants to reserve a vehicle.
31. The system shall be able to book the sharing vehicles via the Sharing Web Service.
32. The regular user has the final decision about what of the offered routes he desire to follow.
33. The system shall offer to the unsubscribed regular user an interface to the Ticket Vendor Service if a traditional transportation is included in the route.
34. The system shall be able to tell if the position of the user is near the position of the sharing vehicle.
35. The system shall be able to tell if the position of the user matches the final destination.
36. The system shall be able to ask the user a feedback every time he exits the route.
37. Regular users should be able to provide feedback on routes and services.
38. The system shall send to the Databse Service the feedbacks received in order to be stored.

Timetable Computation

39. The system is able to retrieve the map from the Database Service.
40. The system is able to display a map interface.
41. User must be able to select the stop or station of interest from the map.
42. The system is able to access from U.T. Web Service real-time timetable data for the specific station.
43. The system shall access from Database Service the standard timetable.
44. The system shall integrate real-time timetable with standard timetable.
45. The system provides the user the updated timetable.

Change Default Transportations

46. The system shall be able to correctly retrieve information about regular user's default transportation preferences from the Database Service.
47. *EcoWay* shall provide a user interface that allows regular users to change their default transportations preferences.
48. The system shall allow subscribed users to change their default transportation preferences, selecting from all type of vehicles.
49. The system shall allow unsubscribed users to change their default transportation preferences, selecting from all type of traditional transports.
50. The system must not allow unsubsribed users to make available sharing vehicles.
51. The system shall communicate back to the Database Services the modifications.
52. *EcoWay* shall show a confirmation message to the user, when Database Service communicates to the system the change accomplishment.
53. The system shall display an error message, whether the Database Service fails to update the preferences due to a server error or connectivity issue.
54. If the user attempts to select a unavailable transportation mode, the system shall informs the user and asks if they want to subscribe to a premium version.
55. In such a case, the system executes the Manage Subscription phase; otherwise, the user is free to continue setting their choices of transportation.

Subscription Management

56. *EcoWay* shall provide a user interface that allows regular users to see their current subscription status.
57. The system shall be able to correctly retrieve information about regular user's subscription status from the Database Service.
58. The system shall allow regular users that were subscribed in the past or are currently subscribed to update their plan.
59. The system shall allow regular users that have never subscribed to buy a new plan.
60. The system shall allow regular users that want to buy or to update a subscription for the first time to see the list of all available plans.
61. The system shall redirect the user securely to the External Payment Service to make the financial transaction.
62. The system is able to receive confirmation from the External payment Service.
63. The system is able to communicate result of transaction to Database Service

Ticket Purchasing

64. *EcoWay* shall provide a user interface that allows regular users to buy a ticket.
65. The system shall redirect the user securely to the Ticket Vendor Service via urban transportation Web Service interface.

Dashboard

66. *EcoWay* shall provide an interface that allows urban area managers to see the Dashboard.
67. The system shall be able to correctly ask the Database Service the reports about urban mobility data.
68. The system shall allow urban area managers to select the type and date of report they look for.
69. The system shall allow urban area managers to visualize the report they selected in the Dashboard.
70. The system shall allow urban area managers to update the dashbord, when the u.a.m. selects a new report.

Contact Mobility Operator

71. *EcoWay* shall provide an interface that allows urban area managers to contact mobility operators (both U.T. WebService and Sharing Web Service).

72. The system shall be able to correctly retrieve contacts from the Database Service.

73. The system shall allow urban area managers to select the desired contact.

74. The system shall allow urban area managers to type and send a message.

75. The system shall allow urban area managers to include a report as an attachment to the message, by selecting type and date of the report.

76. The system shall forward the message to U.T. WebService or Sharing Web Service, based on the input contact selected.

Change Subscription Fares

77. *EcoWay* shall provide an interface that allows urban area managers to change subscription fares.

78. The system shall communicate and save the modification in the regular user section of database service.

79. The system shall permit to urban area managers to navigate back to the u.a.m. interface.

2.5.2 Non-Functional Requirements

User Registration

1. If the time of connection with the database plus the credentials check lasts more than 3 seconds, the system considers it a failed connection.

User Login

2. After 3 minutes from the re-routing, the system will consider the connection to the u.a.m. portal failed anyway.

Route Computation

3. The system shall offer up to 9 alternative routes to a single user.
4. It shall be ensured that the longest route from the routes offered to does not exceed 120% of the duration of the shortest path.

Timetable Computation

5. The systems must process the requested timetable within 2 seconds of user request.
6. The system must ensure accurate mapping of stops or stations on the user interface to prevent any selection errors.

Change Default Transportations

7. The transaction of the system with the Database Service for up- dating preferences should complete within 2 seconds; if it takes longer, the system will consider it failed due to connectivity is- sues.

Subscription Management

8. The purchasing of a subscription must be done in a completely secure way thorugh appropriate protocols.

Ticket Purchasing

9. All interactions with the Ticket Vendor Service shall be encrypted to ensure the security and privacy of user data.
10. The ticketing feature must ensure HTTPS communication, strict CSP, and sandboxing to safeguard user transaction integrity and confidentiality.
11. The system should be able to handle 150 users accessing the ticketing feature simultaneously without performance degradation.

Change Subscription Fares

12. The system must let modify subscription fares by at most one u.a.m per time in order to avoid conflicts.

2.6 Domain Assumptions

User Registration

1. The external app already exists.
2. The Database Service is able to check validity of the credentials.
3. The Database Service is able to create accounts.
4. The Database Service is able to send correct or error answer to *EcoWay*.
5. The u.a.m. portal works correctly and sends back either the token or an error to EcoWay.

User Login

6. The Database Service correctly checks the the login credentials and sends answer to the EcoWay system (either confirmation or error)

Route Computation

7. The Database Service correctly and continuosly provides addresses to the system during user address selection.
8. GPS systems used to track user are operational and provide precise location data at all times.
9. User must grant the system access to their GPS location to enable real-time location tracking.
10. The app reliably communicate the GPS location to the sytem when asked.
11. Web Sharing Service must fully manage the booking of its own vehicles.
12. Web Sharing Service shall provide to each of its own vehicle a QR code needed for identification.
13. The Database Service associates the sharing vehicle ID to the correct user.
14. The Database Service correctly returns the sharing vehicle ID when needed.
15. The Database Service correctly stores and communicate when needed user preferences, timetable, user subrsciption status.

Timetable computation

16. The system succeeds in integrating real-time and standard timetables.
17. The map contains all stations and stops of the metropolitan area.
18. The Urban Transportation Service knows at each moment real-time timetable status.
19. The Urban Transportation Service communicates reliably real-time timetable data to *EcoWay* as it receives the request.

Change Default Transportations

20. The Database Server correctly communicates to EcoWay the default transportation related to the correct user.

21. The Database Service correctly updates the regular user's default transportation preferences when requested.

Subscription Management

22. The Database Service correctly update the regular user's subscription status when required.

23. The Database Service correctly initializes regular users's plan if it is initialized for the first time.

24. The Database Service continuously checks the regular user's subscription status to see if expired.

25. The Database Service marks the status to unubscribed if the subscription expires.

26. External Payment Services used for managing subscriptions and ticket purchases are available and comply with the latest security standards for financial transactions.

Ticket Purchasing

27. Ticket Vendor Service is operational and gets accessible as it receives the request.

28. Ticket Vendor Service handles payment processing securely and efficiently.

29. Users maintains stable connection with Ticket Vendor Service during all the transaction.

30. Users are assumed to consent to being redirected to the Ticket Vendor Service when they choose to purchase tickets through *EcoWay*.

Dashboard

31. The Database Service correctly and continuosly provides title of reports to EcoWay while the u.a.m. is choosing which report to visualize.

32. The Database Service organizes daily collected data in reports, organized by type and date.

33. The data form sensors is correctly managed by the U.T. Web Service.

34. The Database Service independently updates data from U.T. Web Service.

Contact Mobility Operator

35. The Database Service contains updated contacts of U.T. WebService and Sharing Web Service mobility operators.

36. The message is correctly sent and managed by U.T. WebService or Sharing Web Service.

Change Subscription Fares

37. The Database Service mantains information about modifications, such as the involved urban area manager and the time of change, for security reasons.

Additional

38. The real-time data provided by Urban Transportation and Sharing Web Services regarding vehicle locations, availability, and timetables are accurate and timely.

39. The communication infrastructure (Internet connectivity, API gateways, etc.) supporting interactions between EcoWay, its users, and external services is reliable and sustains continuous operation without disruptions.

40. All transport service providers (buses, trams, shared bikes, etc.) collaborate with *EcoWay* by consistently updating their operational status and vehicle data to the shared databases and services.

41. All relevant entities involved in providing data and services to *EcoWay* comply with applicable local, national, and international regulations concerning data privacy, transport safety, and user security.

3 Design

3.1 Architecture

3.1.1 Components Diagram

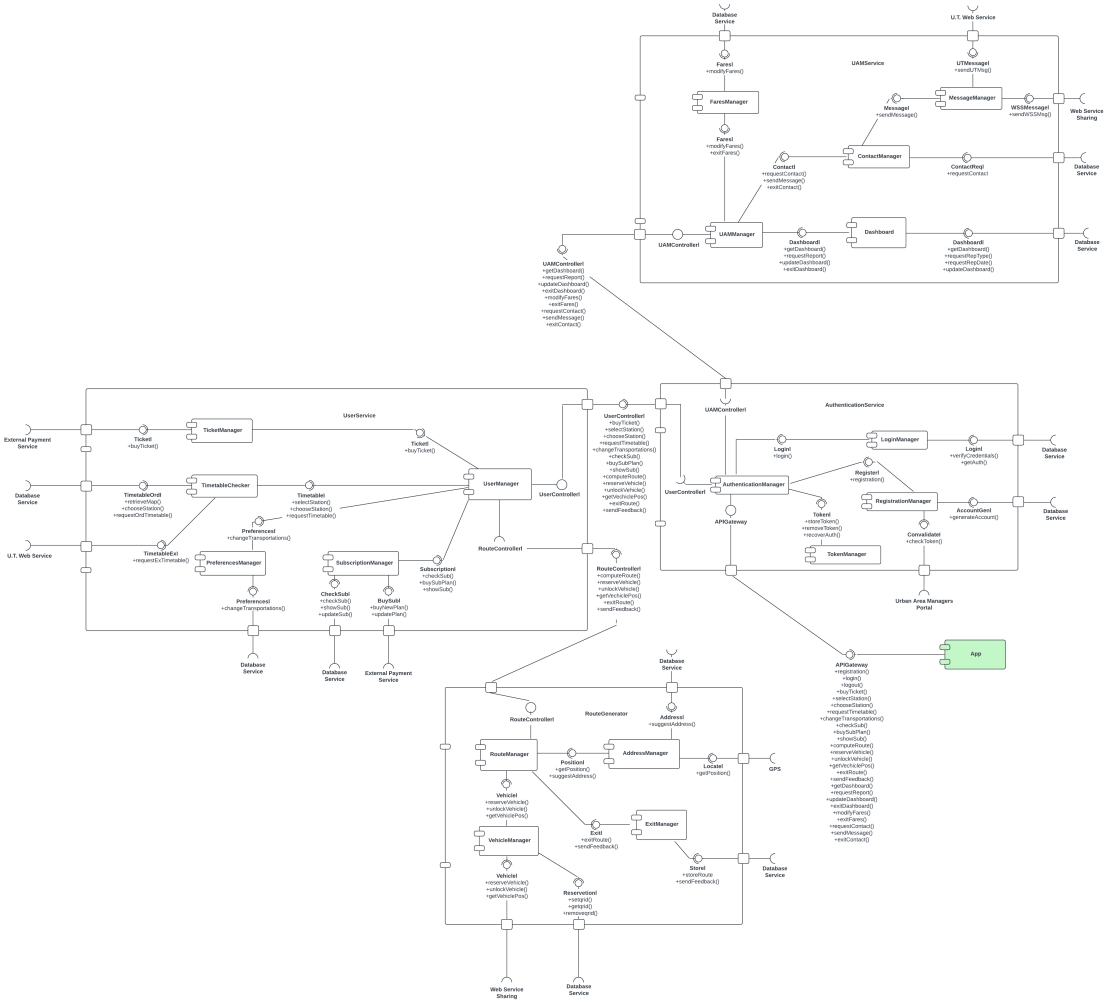


Figure 5: The general structure of the components of the system is here presented. A critical aspect that it's possible to evince is the strong dependance on external service, it explains the design based mainly on required interfaces.

3.1.2 Subsystems and Components

The **AuthenticationService** subsystem includes the following components:

- **AuthenticationManager**: Allows users to register and log in through appropriate interface and methods, and coordinates the LoginManager and RegistrationManager components. It contains a database with active user IDs, which are saved when a user logs in and deleted when the user logs out or their session's unique token expires. This is used to know whom to return response messages to after each operation.
- **LoginManager**: This component receives login requests from AuthenticationManager. It contacts the Database Service to verify the entered login data and requests the user's authorization level.
- **RegistrationManager**: This component receives user registration requests from AuthenticationManager to create a new profile in the *EcoWay* system. If the user wishes to register as an Urban Area Manager, it provides access to the Urban Area Managers Portal, which verifies their identity through a token. Subsequently, in this case, it allows the user to register by providing credentials. User data will be saved in the Database Service. A regular citizen can register directly in the system without needing a token.
- **TokenManager**: This component provides a database that saves the authorization level of each user who has already logged in, along with a unique token identifying the user and their current session, and their userID. This token is also stored by the user, on the App side. This token resolves security-related issues. Thus, when a user makes a request (except for login, logout, and registration requests), the user sends their token. If the token comparison on the TokenManager database is positive, then the user's identity is effectively verified as correct, and access to the desired method is provided. This verification is done at a predetermined time interval between function calls, so for sequential function calls, the check is only performed the first time. The session token is removed when the user logs out or after a specified time during which the user makes no requests.

The **UserService** subsystem includes the following components:

- **UserManager**: Provides users with appropriate methods and interfaces. It coordinates all functions that a regular user can perform.
- **TicketManager**: This component receives from UserManager the user's request to purchase a ticket, directing it to an External Payment Service.
- **TimetableChecker**: This component receives from UserManager the user's request to view the timetable of a selected stop, as well as the request to select that stop. In particular, it first checks with the U.T Service for any real-time schedule changes, and then compares them with the standard timetable in the Database Service. This ensures that the user always receives the updated timetable.
- **PreferencesManager**: This component receives from UserManager the request to modify the user's preferences regarding transportation modes. These preferences will be updated in the Database Service.

- **SubscriptionManager**: This component receives requests from UserManager regarding *EcoWay* subscription plans. It can retrieve the list of subscriptions or the subscription status of the user from the Database Service. It also allows the user to purchase a subscription by directing them to an External Payment Service.

The **RouteGenerator** subsystem includes the following components:

- **RouteManager**: This component receives from UserManager all requests regarding the actual calculation of a route and the methods necessary before and for starting the execution of a single trip. It also manages this execution.
- **AddressManager**: This component receives from RouteManager the request for route suggestions when the user enters inputs about the departure and arrival stations. These addresses are suggested via autocomplete from the Database Service. Additionally, it collects user location data from GPS.
- **VehicleManager**: This component receives from RouteManager all requests related to the management of shared vehicles: reservation, unlocking, and position request, which are provided by a Sharing Web Service.
- **ExitManager**: This component receives from RouteManager the request from the user to terminate the execution of the trip. Therefore, it saves the executed route on the Database Service and asks the user if they would like to leave feedback, collecting this data if provided and saving it in the Database Service.

The **UAMService** subsystem includes the following components:

- **UAMManager**: This component offers Urban Area Managers the methods and interfaces required for collecting and visualizing data reports, contacting partners, and making changes to the services offered.
- **Dashboard**: This component receives requests from UAMManager for Urban Area Managers to access the dashboard. These users can view, update, and request data through the dashboard by selecting the report type and date. They can also disconnect from the dashboard. All report data is saved in the Database Service.
- **ContactManager**: This component receives requests from UAMManager to contact external partners. It allows gathering partner data by accessing the Database Service.
- **MessageManager**: This component executes the request from ContactManager to contact external partners. Once the partner's data is collected, it allows contacting either the U.T Web Service or the Sharing Web Service based on the selection. Users can send messages to these actors, along with any previously collected reports.
- **FaresManager**: This component receives requests from UAMManager for users to modify fares, accessing the Database Service.

3.1.3 Methods

- **registration()**: When the user wants to register, the request is forwarded to the RegistrationManager. Then, the user is asked to select whether they want to register as an Urban Area Manager. If so, they are provided with the Urban Area Manager Portal interface, a digital identity confirmation service where they can confirm their identity with ‘checkToken()‘. If the outcome is positive, or if the user wants to create a profile as a regular citizen, the provided user input data is used to register them in the Database Service with ‘generateAccount()‘.
- **generateAccount()**: Called during and at the end of the execution of ‘registration()‘. With the data provided by the user through ‘registration()‘, including userID and password, the user is registered and saved in the Database Service.
- **checkToken()**: Called during the execution of ‘registration()‘, offers the user the Urban Area Manager Portal interface to confirm their identity to ensure their rights as an Urban Area Manager.
- **login()**: When the user wants to log in, the request is forwarded to the LoginManager. Their input data must first be verified with ‘verifyCredentials()‘, and if successful, their authorization level (citizen or Urban Area Manager) is retrieved from the Database Service with ‘getAuth()‘. With this method, a token is generated and forwarded to TokenManager via AuthenticationManager, where it is saved with ‘storeToken()‘. This token uniquely represents an active session of a specific user. This allows checking if the user is already logged in with each user action and quickly retrieving the user’s authorization to direct them to the correct component (citizen or Urban Area Manager).
- **getAuth()**: Called during the execution of ‘login()‘. Once the credentials entered by the user with ‘login()‘ are verified to be correct with ‘verifyCredentials()‘, it requests the user’s authorization level from the Database Service and generates a unique token to uniquely identify the user.
- **verifyCredentials()**: Called during the execution of ‘login()‘, allows access to the Database Service interface to compare the data provided as input for login by the user with the data saved in the Database Service. It can return a positive outcome and continue the execution of ‘login()‘, or return an error message to the user in case of a negative outcome.
- **‘logout()‘**: When the user wants to log out, the message is forwarded to AuthenticationManager. Here, the ‘removeToken()‘ method is called, which removes the token on the user’s session from the TokenManager database component for the user who wants to log out.
- **storeToken()**: Called during the execution of ‘login()‘. After retrieving the user’s login authorization level from the Database Service with ‘getAuth()‘ and generating their unique session token, AuthenticationManager calls the ‘storeToken()‘ method to save the user’s authorization level, their userID, and their respective token in the TokenManager database.
- **removeToken()**: Called during the execution of ‘logout()‘. Removes the token on the user’s session from the TokenManager database component for the user who wants to log out.

- **recoverAuth()**: Called during the execution of a user request, except for 'login()', 'logout()', and 'registration()'. Specifically, this method is used to verify if the user is already logged in and retrieve their authorization level from the TokenManager database. AuthenticationManager, based on the token sent by the user, retrieves their authorization level. Now AuthenticationManager can verify that the user truly has access to the method they called and forward the request to the correct component.

- **buyTicket()**: When the user requests to purchase a ticket, the request is forwarded to the TicketManager component, which provides the user with the interface of the External Payment Service.

- **requestStation()**: When the user wants to select a station on the map, the request is forwarded to the TimetableChecker, which allows the user to interface with the Database Service containing all saved stations. The list containing all these stations is retrieved with 'retrieveMap()'. The user's choice is finalized with '+chooseStation()'.

- **retrieveMap()**: Called during the execution of 'requestStation()', retrieves from the Database Service a list containing all stations and provides it to the user.

- **chooseStation()**: Called during the execution of 'requestStation()' after retrieving the list of stations with 'retrieveMap()'. It finalizes the user's request by allowing them to choose and input a station from the provided list. This is used to request the timetable of a specific station. 'chooseStation()' allows saving the station selected by the user in the Database Service so that when 'requestOrdTimetable()' is called, the timetable of a specific station is retrieved. This data on the database expires after a predetermined time or upon calling 'requestOrdTimetable()'.

- **requestTimetable()**: When the user requests to view the timetable, the request is forwarded to the TimetableChecker. Then, 'requestExTimetable()' is called, followed by 'requestOrdTimetable()'. Finally, the updated real-time timetable is provided to the user.

- **requestExTimetable()**: Called during the execution of 'requestTimetable()', contacts the U.T Web Service to receive any changes to the standard timetable schedule first. Information regarding all timetables is retrieved in the form of a list of timetables.

- **requestOrdTimetable()**: Called during the execution of 'requestTimetable()' after 'requestExTimetable()'. It compares the standard timetable with any schedule changes to offer the user the updated real-time timetable. Specifically, starting from the list of timetables obtained with 'requestExTimetable()', the individual timetable saved in the user's Database Service is compared. After this comparison, the timetable saved in the user's Database Service level is removed, and the updated timetable is provided to the user.

- **changeTransportations()**: When the user wants to modify their preferences regarding the use of certain means of transportation, the request is forwarded to PreferencesManager, which modifies the new choices entered by the user as input on the Database Service.

- **checkSub()**: When the user wants to see their current subscription status, the request is forwarded to Subscription Manager. Then, the user's status is retrieved from their profile on the Database Service.

- **buySubPlan()**: When the user wants to purchase a new subscription or upgrade their current one, the request is forwarded to Subscription Manager. First, the user's subscription status is checked with 'checkSub()'. Then, if a subscription is active or was previously active and the user wants to extend or re-subscribe, 'updatePlan()' is called. If the user has never subscribed before, 'buyNewPlan()' is called. In any case, payment is processed by providing the user with the External Payment Service interface. If the transaction is successful, the user's subscription status is updated on the Database Service with 'updateSub()'.

- **updatePlan()**: Called during the execution of 'buySubPlan()'. If it is found that the user has already purchased a plan in the past and wants to renew it, or is already subscribed and wants to extend or modify the type of subscription, their plan is updated on the Database Service.

- **buyNewPlan()**: Called during the execution of 'buySubPlan()'. If the user subscribes for the first time, their plan is initialized on the Database Service.

- **updateSub()**: Called at the end and during the execution of 'buySubPlan()'. If the subscription purchase is successful, the user's subscription is updated on the Database Service.

- **showSub()**: When the user wants to view all available subscription plans, the request is forwarded to Subscription Manager. Then, the list of all available plans is retrieved and shown to the user.

- **computeRoute()**: When the user wants to calculate the route, the request is forwarded to the Route Manager. Initially, the user is asked if they want to use their current position or a new starting position. In the former case, the position is obtained with 'getPosition()', which retrieves the user's position via GPS. Otherwise, based on the user's real-time input, a list containing address autocompletions is provided based on compatible positions retrieved and processed by the Database Service. In any case, this latter algorithm is also used for destination selection. The user is offered a list of suggested routes. Once one is confirmed, if traditional means of transportation are available on the route, the user is offered to purchase a ticket with 'buyTicket()'. Then, if a shared vehicle is selected, it is identified with 'getVehiclePos()' and reserved for use with 'reserveVehicle()'. Upon the user's arrival, it is unlocked with 'unlockVehicle()'. All this is managed by an external interface, which is the Web Service Sharing. The user then completes their route with 'exitRoute()'.

- **reserveVehicle()**: When the user has decided to use a shared vehicle for their route, after identifying the shared vehicle with 'getVehiclePos()' from the Web Service Sharing, the user can reserve it. This is always done through the external interface of the Web Service Sharing. A key is set for the user on the Database Service with 'setqrid()', allowing them to unlock the reserved shared vehicle with a QR code.

- **setqrid()**: Called during the execution of 'reserveVehicle()'. A unique key corresponding to the ID of the shared vehicle reserved by the user is set on the Database Service.

- **removeqrid()**: Called during the execution of 'exitRoute()'. The unique ID of the shared vehicle reserved during the user's route is removed from the Database Service.

- **unlockVehicle()**: When the user wants to unlock the shared vehicle, the ID of the shared vehicle is retrieved from the Database Service with 'getqrid()', and compared with the vehicle's

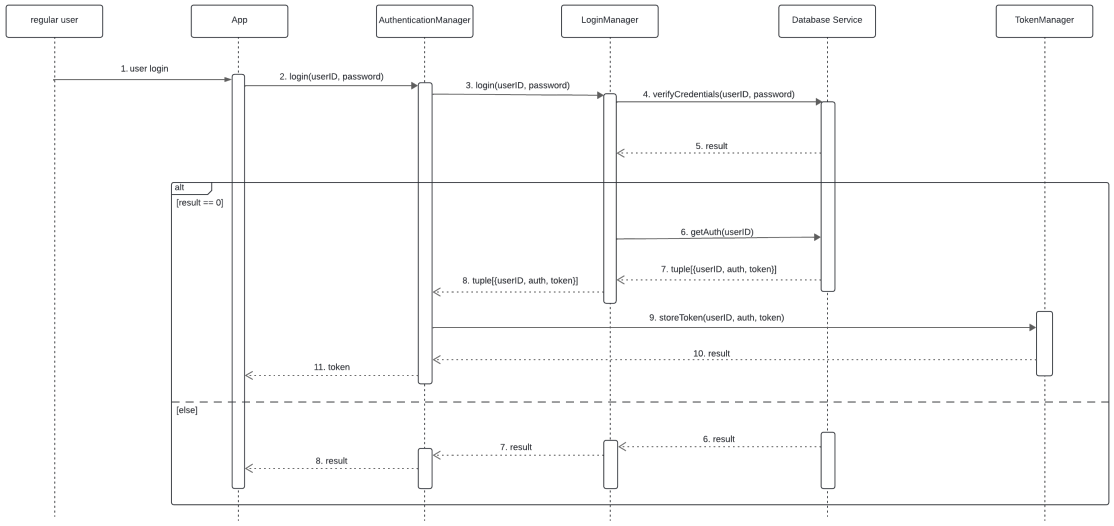
QR code through the Web Service Sharing. If the comparison result is positive, the vehicle is unlocked and made available for use by the user.

- **getVehiclePos()**: Called during the execution of 'reserveVehicle()'. If it is necessary to know the position of the shared vehicle, it is located by the Web Service Sharing using its position sensor.
- **exitRoute()**: When the user wants to end their route, the request is forwarded to ExitManager. Then, the user is asked if they want to leave feedback, which they can send with 'sendFeedback()'. In any case, the route is saved on the Database Service.
- **sendFeedback()**: Called during the execution of 'exitRoute()'. The user can leave feedback at their discretion, which is collected and sent to the Database Service. The request to leave feedback is sent back to the user, and in any case forwarded again to ExitManager: if the response is negative, the process ends, otherwise the feedback is sent with the response message.
- **getDashboard()**: When the Urban Area Manager wants to view the dashboard, the request is forwarded to the Dashboard. Then, the dashboard data is retrieved from the Database Service.
- **requestReport()**: When the Urban Area Manager wants to view a specific report, the request is forwarded to the Dashboard, with the report type and date as input. Then, the component retrieves the requested report from the Database Service by first calling '+equestRepType()' and then 'requestRepDate()'.
- **requestRepType()**: Called during the execution of 'requestReport()', it selects the correct report type parameter from the Database Service to search for.
- **requestRepDate()**: Called during the execution of 'requestReport()' after 'requestRepType()', it selects the correct report date parameter from the Database Service to search for, and returns the requested report if found in the Database Service, or an error message otherwise.
- **updateDashboard()**: When the Urban Area Manager wants to update the dashboard with the latest data, the request is forwarded to the Dashboard, and the data is requested from the Database Service.
- **exitDashboard()**: When the Urban Area Manager wants to exit the dashboard, the request is forwarded to the Dashboard, and if successful, the user is returned to the main menu.
- **modifyFares()**: When the Urban Area Manager wants to modify *EcoWay* fares, the request is forwarded to the FaresManager, where, with the inputs provided by the user, the fares are modified on the Database Service. A message with a positive outcome is returned to the user in case of success, an error otherwise.
- **exitFares()**: When the Urban Area Manager wants to finish and conclude their modifications to *EcoWay* fares, the request is forwarded to the FaresManager, and if successful, the user is returned to the main menu.

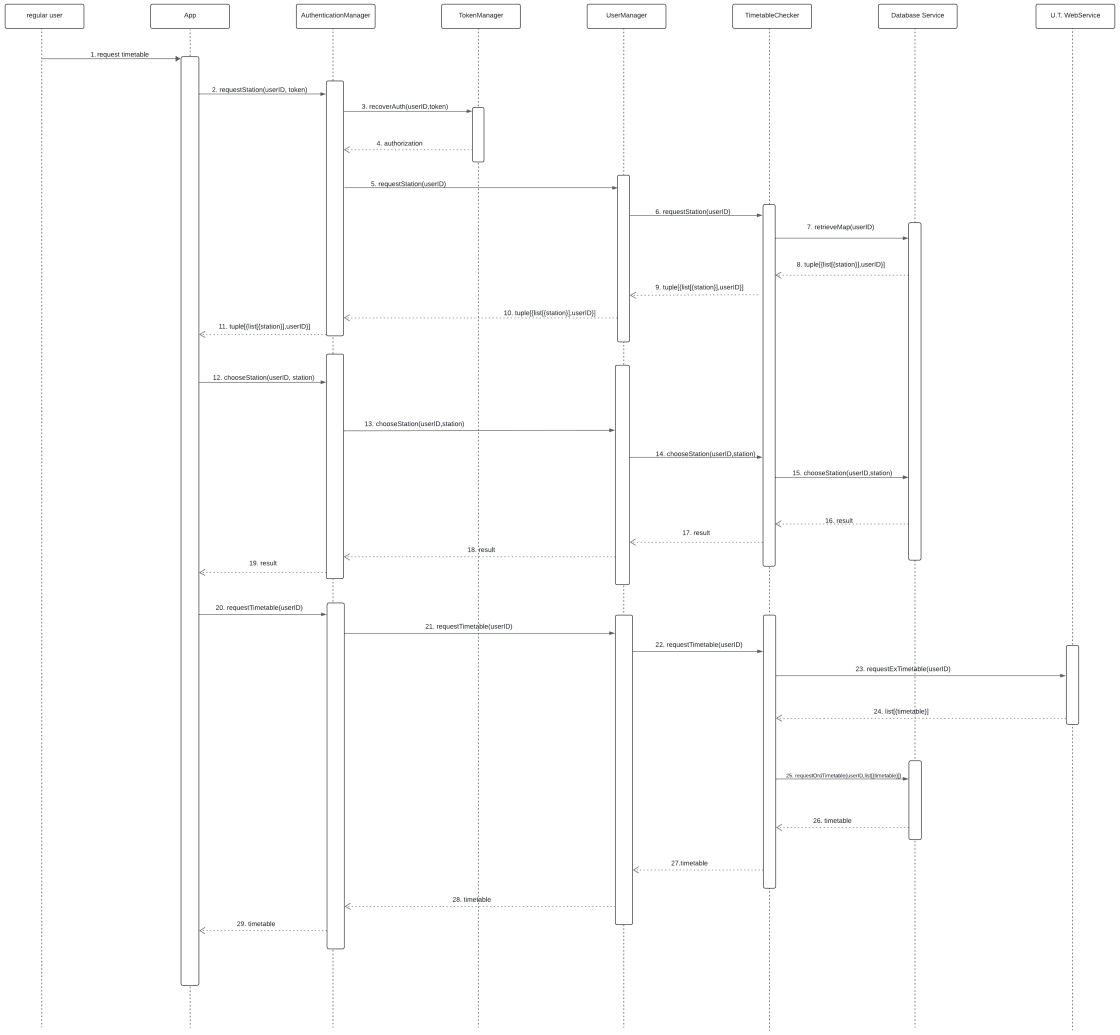
- **requestContact()**: When the Urban Area Manager wants to retrieve information about an external partner, the request is forwarded to the ContactManager. A list containing partner information is retrieved from the Database Service and returned to the user.
- **sendMessage()**: When the Urban Area Manager wants to contact an external partner, the request is first forwarded to the UAMManager. If desired, the report to be sent to the partner can be requested via 'requestReport()'. This report is included in the arguments of the 'sendMessage()' call. Then the request is forwarded to the MessageManager. Based on the input provided, it will be possible to contact the U.T. Web Service or the Web Service Sharing. Consequently, 'sendUTMsg()' or 'sendWSSMsg()' will be called respectively. The execution ends upon user request via '+exitContact()'.
- **sendUTMsg()**: Called during the execution of 'sendMessage()', the Urban Area Manager is put in communication with the interface of the U.T. Web Service.
- **sendWSSMsg()**: Called during the execution of '+sendMessage()', the Urban Area Manager is put in communication with the interface of the Web Service Sharing.
- **exitContact()**: When the Urban Area Manager wants to finish contacting external partners, the request is forwarded to the ContactManager, and if successful, the user is returned to the main menu.

3.2 Sequence diagrams

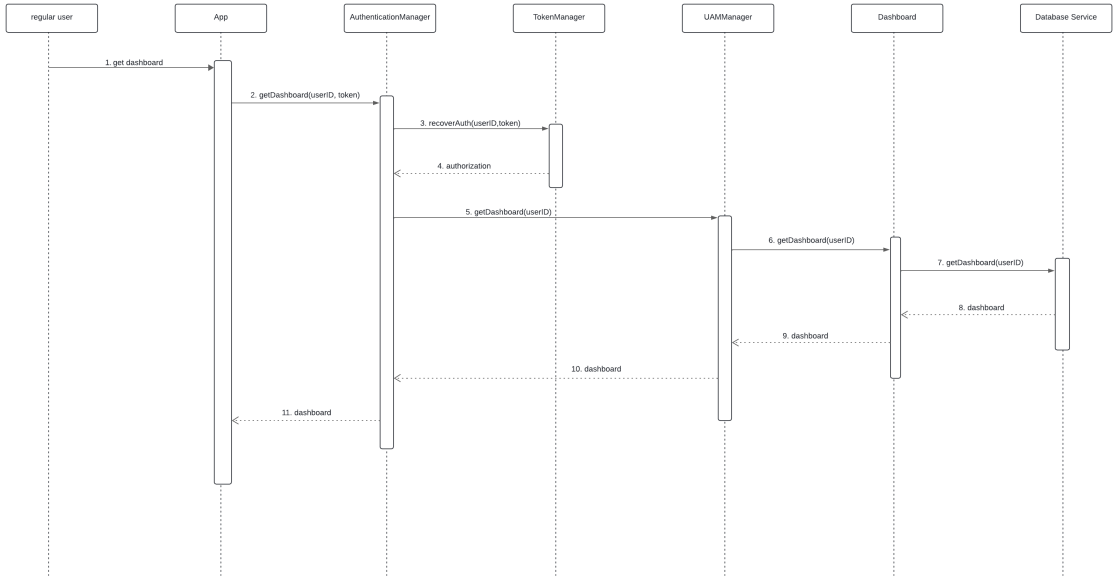
Login Sequence Diagram



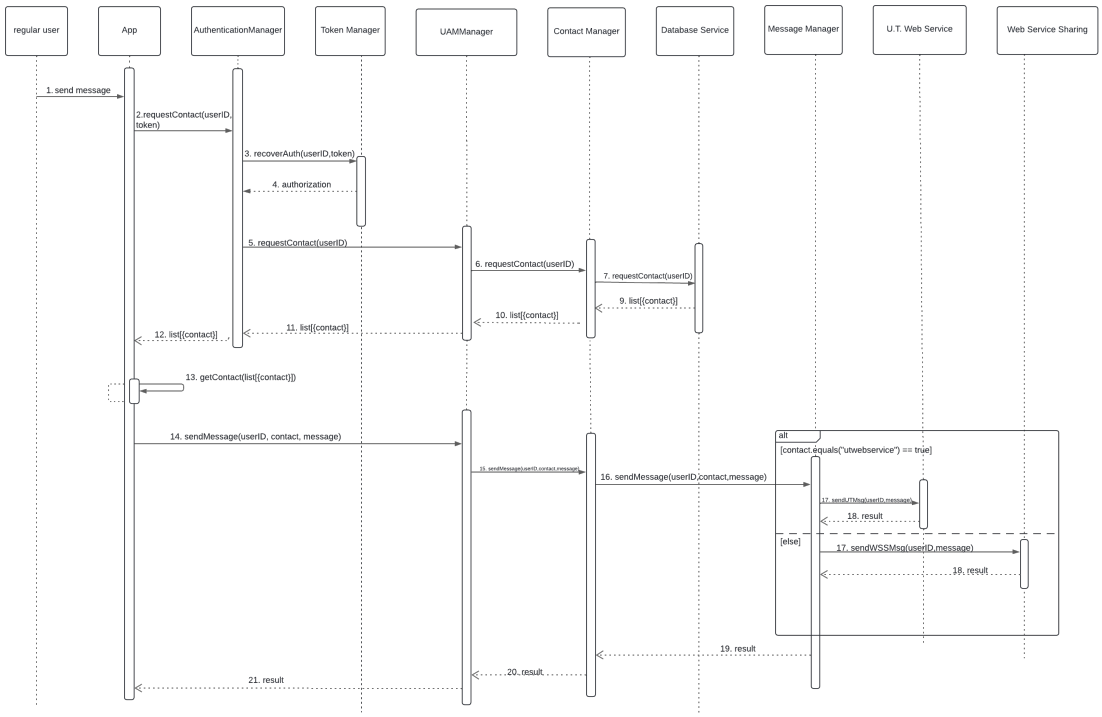
Get Timetable Sequence Diagram



Get Dashboard Sequence Diagram



Send Message Sequence Diagram



3.3 Design decisions

3.3.1 Subsystems

EcoWay is composed of four subsystems, as clearly visible in the component diagram. Partitioning the system helps in general to achieve modularity: the best case is not only being able to develop different parts separately, but is also desirable that different subsytems are indipendent in terms of both functionalities and different need for scalability, number of interactions, dependence on different external actors or services and so on.
Thus division criteria are mainly to split different processes, if possible to split different actors and if necessary to separate functionalities that have critical necessities in terms of computation.
The final choice fell on the following four subsystems:

- AuthenticationService
- UserService
- RouteGenerator
- UAMService

Clearly, the choice to seve the connection between User and UAMService subsystems arises because they focus on different actors. Additionally, *EcoWay*'s functionalities related to the different user types (regular and U.A.M.) are completely independent. The only common ground is some of the data, which, however, is managed by the Database Service. Therefore, there's no actual dependency.
Both primary actors need to register and login. In this case, the paths for the two actors differ only slightly, so it makes sense to have a unique subsystem.
The RouteGenerator is split from the rest because the most complex and data-intensive tasks happen to achieve the correct computation of the route, which is the main feature around which EcoWay is built.

3.3.2 Reliance on external services

Referring back to the component diagram once again, the system design heavily relies on external services not only for data but also for ancillary computations necessary for *EcoWay*'s correct functioning. This dependency on external operations explains why only ball and socket components are depicted in the diagram.
The components constituting the subsystems of *EcoWay* enhance modularity but do not contribute significantly to computation. Instead, they provide access to interfaces for external services, allowing users to complete specific requests such as payments, which are not managed by the system and require access to and functionality of the Database Service.
The Database Service is a critical aspect of our system, as it handles almost all computational tasks and stores most of the data necessary for the system's correct behavior. With this element and assumption, *EcoWay* can guarantee all the promised requirements. However, due to the heavy reliance on external services and this particular device, we must ensure complete reliability and strong efficiency of the Database Service

3.3.3 Remote Presentation

EcoWay adopts a **Remote Presentation Architecture**, in terms of client-server architecture: the users only interact with the application which is basically only a GUI, while the actual computation happens in part in the back-end of *EcoWay* and a significant part in the Database Service.
On the client side is provided an interface for communicating, through an exchange of actions with the Database Service. For example the Urban Area Manager, by retrieving the necessary information, can access the interface to contact secondary actors.
This setup allows us to quickly respond to incoming requests, saving computation on the client side to increase performance: by minimizing computation on the client side, we improve overall system responsiveness.

3.4 The critical aspect: High-Performance Data Processing

The next section will present the main critical aspect which is indeed the HPC necessities of the system, as previously noted in section 3.3.1.
EcoWay system include critical and complex data processing tasks that involve extensive data retrieval and dynamic data interactions. To effectively manage these challenges, employing high-performance computing (HPC) techniques is crucial.

- **Parallel Processing**: Use parallel computing to handle the computation of multiple routes simultaneously. This approach is especially effective when processing routes that involve varying transportation modes and complex criteria such as sustainability and travel time.
- **Distributed Computing**: We enhance efficiency by clustering route calculations on the same server for users with similar start and end points. This approach optimizes resource use by reducing redundant computations and improves response times by leveraging shared data and pre-computed route segments.
- **Load Balancing**: Use load balancers to distribute user requests evenly across servers, ensuring no single server is overwhelmed, which is critical for real-time data processing.
- **Clustering**: We implement server clusters that specialize in handling real-time data feeds and integration, where each cluster is dedicated to processing data from a specific urban zone. By localizing the data handling, each cluster optimizes real-time updates relevant to its specific area, ensuring swift and precise navigation and transportation information to users.