Digital Twin - Aveiro Tech City Living Lab

PI - Projeto em Informática
Technical Report - Project Specifications

Bernardo Pinto - 105926
Filipe Obrist - 107471
José Mendes - 107188
Mariana Perna - 108067
Rafaela Dias - 108782

2023/2024

Abstract

This project aims to create a platform for urban Digital Twin of Aveiro, allowing visualization of physical data and simulations of scenarios for autonomous vehicles and multi-modal transportation.

Contents

1 Introduction

Smart cities have been emerging as a viable solution to tackle environmental and social challenges. However, tackling the challenges of understanding and predicting the dynamics of these cities can be problematic. To retify this, the concept of Digital Twin offers a novel approach by virtually representing physical environments, allowing real-time monitoring, analysis, and action based on existing data. With 3D modeling, Digital Twins mirror physical objects in the digital environment, making it more realistic and easier to visualize several scenarios. Developed as a risk-free system, Digital Twins have found new applications in Smart Cities and autonomous mobility, continuously gathering data from sensors and other devices.

Leveraging initiatives like the Aveiro Tech City Living Labs (ATCLL) and Altice Labs' Live! Ecosystem, this project aims to develop a platform supporting Digital Twin concepts for urban environments. By allowing visualization of physical twin data and simulation of scenarios, the platform targets use cases like road blockage scenario emulation and large-scale multi-modal transportation simulation. Validation will occur in a laboratory environment, utilizing real data from the sensors around the city of Aveiro and Altice Labs' urban platform, thereby addressing the challenges of modelling and predicting Smart City dynamics.

2 Our Team

This project is composed of 5 members and 4 advisors. The following table (Table 1) shows these elements and their roles.

Role	Name
Member	Bernardo Pinto
Member	Filipe Obrist
Member	José Mendes
Member	Mariana Perna
Member	Rafaela Dias
Advisor	Duarte Raposo
Advisor	Filipe Cabral Pinto
Advisor	Pedro Rito
Advisor	Susana Sargento

Table 1 - Team Roles

3 Inception Phase

3.1 Product Concept

3.1.1 Problem

Urban traffic management is a complex and challenging task. City planners often struggle with the unpredictability of traffic patterns and the difficulty in managing unexpected events. Without advanced tools, their ability to foresee and mitigate these issues is severely limited. This project aims to address these challenges head-on.

Our project tackles several critical challenges in traffic management and simulation, particularly focusing on the limitations of current methodologies. One of the main issues is the difficulty in simulating extreme or unlikely scenarios, such as road closures or construction sites, without the aid of a Digital Twin. This absence of a digital representation hinders our ability to predict and manage potential traffic crises effectively.

Proactive traffic management and planning are significantly more challenging without a Digital Twin. The lack of detailed data and precise simulations makes it difficult to predict potential problems and implement preventive measures, leading to inefficient traffic management. Additionally, testing specific scenarios often requires physical presence, which involves inefficient resource allocation and substantial time consumption. On-site testing is not only costly but also time-intensive, further complicating efforts to optimize traffic systems.

By developing a Digital Twin for the city of Aveiro, our project aims to overcome these limitations, providing a powerful tool for accurate traffic simulation, real-time data integration, and efficient urban planning.

3.1.2 Goals

To address these challenges, our project aims to develop a comprehensive solution through the creation of a Digital Twin platform. Our primary goal is to build a system that can visualize real city data and generate simulated scenarios, providing a detailed and dynamic digital representation of Aveiro. This Digital Twin will enable advanced simulations and more effective traffic management, revolutionizing how we approach urban planning.

One key objective is to develop a robust platform that integrates real-time data from the city's infrastructure. By visualizing current traffic conditions and simulating various scenarios, city planners can gain a deeper understanding of actual traffic flows and identify potential issues before they escalate.

Additionally, our project explores the complexities of multi-modal transportation scenarios. By conducting large-scale simulations, we aim to understand how different modes of transportation, such as cars, bicycles, and public transit, interact within the city. For example, simulating an increase in the number of bicycles will help us assess its impact on road traffic and optimize overall mobility.

Furthermore, we are committed to ensuring that our simulated scenarios are grounded in reality. To achieve this, we will test and validate our simulations using real data obtained through the city's infrastructure and the urban platform provided by Altice Labs. This step is crucial to ensure that our solutions are effective, reliable, and applicable to real-world urban planning and traffic management.

Ultimately, by developing this Digital Twin for the city of Aveiro, our project aims to provide city planners with a powerful tool to optimize traffic management and urban planning, paving the way for a smarter and more efficient city.

3.1.3 State-of-the-Art

To understand a little bit more about Digital Twin and our project, we analyzed several IEEE papers, of which we present here the most essential ones for gaining deeper insights into the subject.

1. **Building a Motorway Digital Twin in SUMO** [1]

   - Introduces the Digital Twin concept.
   - Develops a Digital Twin specific to the Geneva motorway using SUMO.
   - Incorporates real-time data, aligning with our objectives.

   Click here to visit the document

2. **Smart Mobility Digital Twin for Automated Driving** [2]

   - Provides an overview of the Digital Twin concept.
   - Focuses on real-time monitoring and planned route development considering traffic conditions.

   Click here to visit the document

3. **Efficient Procedure of Building University Campus Models for Digital Twin Simulation** [3]

   - Constructs a 3D model of a university campus (Digital Twin) using the CARLA project in Unreal Engine.
   - Incorporates CARLA with SUMO, demonstrating integration possibilities for our project.

   Click here to visit the document

These works collectively contribute to our understanding of Digital Twin applications, platform development, and integration with real-time data sources, guiding our project's direction and goals.

3.2 Workflow

3.2.1 Tasks and Project Calendar

The team developed a task list in the first moments of this project:

1. **Study of Various Sensor Types and Data Generated**

   Investigate different sensor types and the data they generate.

2. **Study of SUMO Software**

   Explore the functionalities and applications of SUMO for urban traffic simulation, establishing a solid foundation for the project.

3. **Study of CARLA Platform**

   Gain an in-depth understanding of the CARLA platform, dedicated to autonomous vehicle simulation, essential for successful integration in the Aveiro context.

4. **Design of Architecture for Simulated and Real Data Integration**

   Develop the architecture design that will enable the connection between simulators (SUMO and CARLA), visualizers, and data from sensors, systems, and connected vehicles. The conceptual phase is where the overall system structure will be outlined.

5. **Development of Connectors and Frameworks**

   Implement the connectors and frameworks necessary to achieve the designed architecture. Using technologies like REST, ROS2, and MQTT, practical components will be developed to enable efficient communication between various system elements.

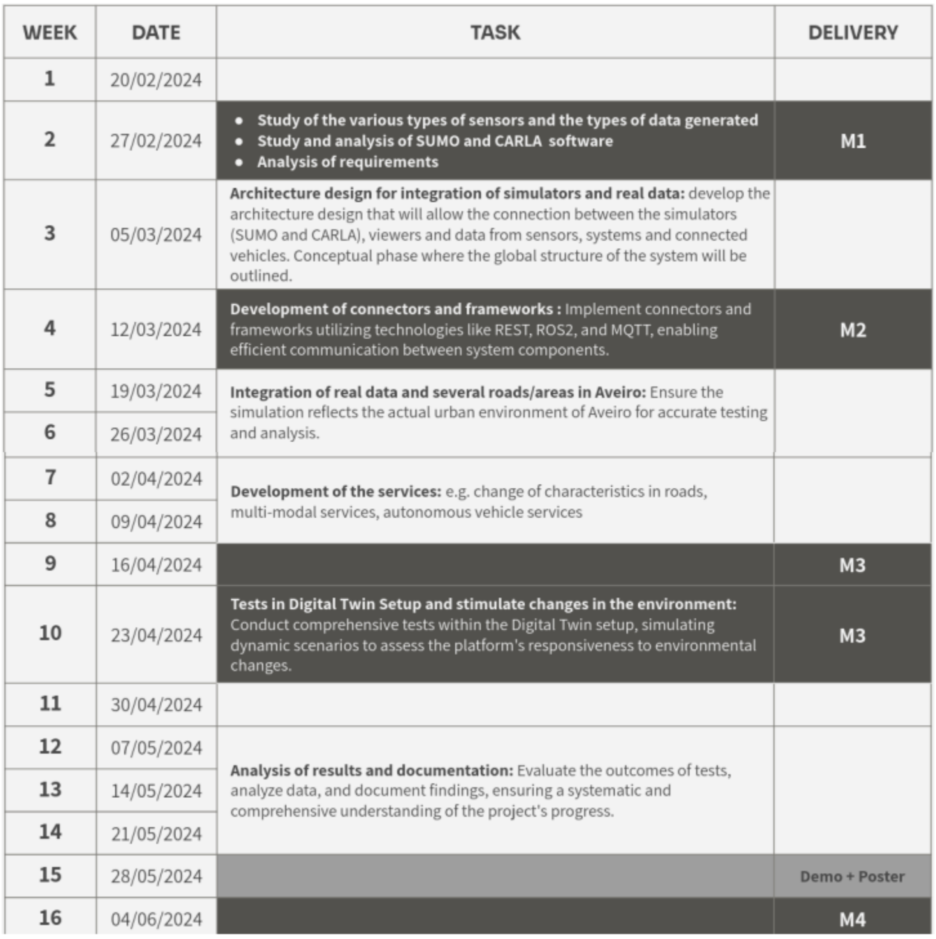
6. **Testing on Digital Twin Setup**

   Conduct tests on the Digital Twin setup after the implementation of connectors and frameworks. Tests will be conducted using simulated and real data from ATCLL infrastructure and Altice Labs urban platform. This phase is crucial to ensure that the architecture functions as expected and that integration between components is effective.

7. **Stimulate Environment Changes**

   Test the robustness of the Digital Twin by modelling dynamic scenarios and evaluating how the system responds to simulated changes in traffic, autonomous vehicle behaviour, and environmental conditions. This will provide valuable insights to optimize the model and anticipate potential challenges in the Aveiro Living Lab.

In establishing our project timeline, we encountered challenges in quantifying the time needed for each task due to their varying complexities and dependencies. Despite these challenges, we revised our timeline various times to ensure it aligned with our project objectives, and we built our project calendar (Fig 1).



WEEK	DATE	TASK	DELIVERY
1	20/02/2024		
2	27/02/2024	- **Study of the various types of sensors and the types of data generated** - **Study and analysis of SUMO and CARLA software** - **Analysis of requirements**	M1
3	05/03/2024	**Architecture design for integration of simulators and real data:** develop the architecture design that will allow the connection between the simulators (SUMO and CARLA), viewers and data from sensors, systems and connected vehicles. Conceptual phase where the global structure of the system will be outlined.	
4	12/03/2024	**Development of connectors and frameworks :** Implement connectors and frameworks utilizing technologies like REST, ROS2, and MQTT, enabling efficient communication between system components.	M2
5	19/03/2024	**Integration of real data and several roads/areas in Aveiro:** Ensure the simulation reflects the actual urban environment of Aveiro for accurate testing and analysis.	
6	26/03/2024		
7	02/04/2024	**Development of the services:** e.g. change of characteristics in roads, multi-modal services, autonomous vehicle services	
8	09/04/2024		
9	16/04/2024		M3
10	23/04/2024	**Tests in Digital Twin Setup and stimulate changes in the environment:** Conduct comprehensive tests within the Digital Twin setup, simulating dynamic scenarios to assess the platform's responsiveness to environmental changes.	M3
11	30/04/2024		
12	07/05/2024	**Analysis of results and documentation:** Evaluate the outcomes of tests, analyze data, and document findings, ensuring a systematic and comprehensive understanding of the project's progress.	
13	14/05/2024		
14	21/05/2024		
15	28/05/2024		Demo + Poster
16	04/06/2024		M4

Fig. 1 - Project Calendar

3.2.2 Communication Plan and workflow

To store our project we created a repository (GitHub Repository) in the **GitHub** platform and we developed a website (Website - Digital Twin) to display and store all our information related to our work, including reports, presentations and the project calendar.

To track our progress, we used the backlog of the GitHub platform, where we defined tasks and distributed responsibilities for each iteration.

A weekly meeting was scheduled to keep in touch with our advisors and update them on our progress and showcase our work. These meetings serve as valuable ways to receive feedback and guidance, ensuring alignment with project objectives and facilitating informed decision-making. **Slack** was also used to keep in touch with our advisors. The team itself maintained contact, through frequent small meetings, ensuring alignment of the objectives of the week. Additionally, a **Discord** group has been established to facilitate communication and information sharing among team members.

4 Elaboration Phase

4.1 Requirements

4.1.1 Requirements Gathering

During the requirements-gathering phase, we explored various approaches:

1. Discussion with our advisors
2. Team brainstorming sessions
3. Information gathering from documentation
4. Understanding the problem
5. Related work review

4.1.2 Functional Requirements

- **Real-time traffic monitoring:** Essential for understanding and managing real-time flow.
- **Traffic simulation:** Enables testing of multiple scenarios and optimization of management strategies.
- **Integration of real and simulated environment:** Facilitates the creation of a comprehensive model of urban mobility.
- **Visualisation of integration in 3D:** Provides a clear and immersive perspective on the system's performance and its potential impact on urban mobility.

4.1.3 Non-Functional Requirements

- **Scalability:** Ensuring that the system efficiently scales with more users and data.
- **Reliability:** Ensuring that simulation and analysis occur seamlessly to promote user confidence.
- **Low Latency:** For a smooth user experience, it's crucial that the system responds quickly to commands and requests, essential for providing timely information.
- **Maintainability:** The code should be easy to understand and modify, allowing for efficient updates.
- **Usability:** Ensuring an intuitive experience for all users, facilitating effective utilization.

4.2 Actors

We have three types of actors who can interact with our project system:

- **Companies:** Focused on utilizing the data to develop applications that improve urban mobility;
- **Researchers:** Inclined on the learning factor and contributing to the field of urban mobility and autonomous vehicles;
- **Developers:** Their objective is to build commercially viable solutions.

4.3 Use Cases

4.3.1 Personas

Dr. Josefa Rodriguez

- Age: 35
- Job: Researcher at a Transportation Research Institute
- Motivation: Josefa is driven by a passion for understanding urban mobility and finding innovative solutions to improve transportation systems. With a background in data science and urban planning, she is dedicated to conducting research that contributes to safer, more efficient, and sustainable urban environments. Josefa thrives on data analysis and simulation modeling, constantly seeking ways to integrate real vehicle data into her research to generate actionable insights for urban planners and policymakers.

Fig. 2 - Dr. Josefa Rodriguez

Artur Cabral

- Age: 40
- Job: Urban Manager at a City Planning Department
- Motivation: Artur is dedicated to ensuring the smooth operation of urban transportation systems and improving the quality of life for city residents. With years of experience in urban planning and management, he understands the importance of data-driven decision-making in addressing traffic congestion and improving traffic flow. Artur is motivated to implement innovative solutions that optimize road network behaviour and enhance the overall efficiency of urban mobility.

Fig. 3 - Artur Cabral

4.3.2 Main Scenarios

Josefa Rodriguez, a seasoned researcher at a prestigious Transportation Research Institute, is tasked with addressing the increasing traffic congestion issues in her city. Armed with a wealth of data science and urban planning knowledge, Josefa turns to the project's system to tackle this challenge.

Scenario 1:

- Josefa starts her day by accessing the project's system, where she collects real-time vehicle data from various traffic sensors.
- With meticulous attention to detail, Josefa integrates this diverse dataset within the system, ensuring it encompasses crucial parameters like traffic flow, vehicle speed, and road conditions.

Scenario 2:

- Josefa delves deep into the integrated dataset, mining it for insights into the city's traffic patterns and behaviours.
- Leveraging her expertise, Josefa seamlessly integrates this real vehicle data into her research models, laying the groundwork for her investigation into urban mobility dynamics.

Scenario 3:

- With the dataset at her fingertips, Josefa employs the project's system to create intricate traffic simulations tailored to the city's unique landscape.
- These simulations encompass diverse scenarios, from routine rush hour congestion to unforeseen events like road closures and accidents, allowing Josefa to envisage a 3D representation of potential traffic scenarios.

Scenario 4:

- With her research progressing, Josefa ensures the preservation of her simulations within the system.
- By exporting and storing her previous simulations, Josefa establishes a comprehensive repository of urban mobility data, laying the foundation for future analyses and informed decision-making.

Artur Cabral, a seasoned urban manager at the helm of a bustling city's Planning Department, faces the daunting task of alleviating traffic congestion and enhancing urban mobility. With an eye for innovation, Artur turns to the project's system to tackle these pressing challenges head-on.

Scenario 5:

- Artur kicks off his day by diving into the project's system, where he crafts detailed traffic simulations tailored to the city's intricate road network.
- Armed with these simulations, Artur embarks on a virtual journey through the city's bustling streets, exploring a plethora of traffic scenarios and testing various management strategies to optimize traffic flow.

Scenario 6:

- Drawing upon insights gleaned from his simulations, Artur orchestrates real-world changes to the city's traffic infrastructure.
- From fine-tuning traffic signal timings to implementing strategic lane configurations, Artur works tirelessly to sculpt a road network that seamlessly accommodates the city's burgeoning traffic demands.

Scenario 7:

- Artur bridges the gap between virtual and real-world traffic environments by integrating real-time traffic data with his simulated scenarios.
- This integration provides Artur with a holistic understanding of the city's urban mobility landscape, empowering him to make informed decisions that resonate with real-world traffic dynamics.

4.3.3 Use Cases Diagram

From the scenarios, we can extract common needs such as data analysis, simulation, and monitoring, which are illustrated in the use case diagram (Figure 4).

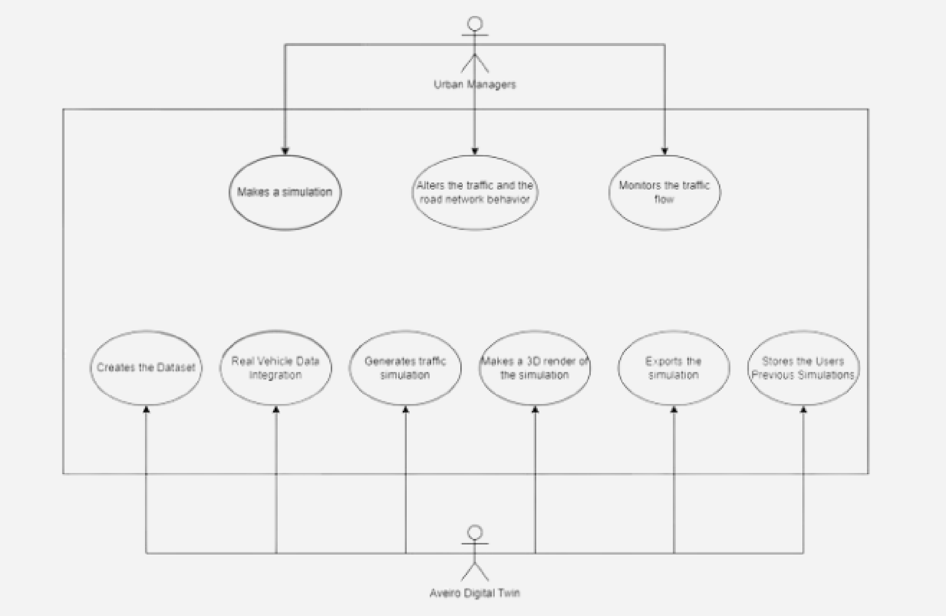

Urban Managers
Makes a simulation
Alters the traffic and the road network behavior
Monitors the traffic flow
Creates the Dataset
Real Vehicle Data integration
Generates traffic simulation
Makes a 3D render of the simulation
Exports the simulation
Stores the Users Previous Simulations
Aveiro Digital Twin

Fig. 4 - Use Case Diagram

4.4 Architecture

Figure 5 describes our project's architecture

The architecture begins by representing the components of ATCLL being reused in this project, such as the sensors for data collection, the MQTT broker for real-time data reception, and the STH Comet API for accessing persisted historical data in a database.

Within the scope of this project, we highlight the first Python module, considered the system's core. This module has various functionalities, including the consumption of real-time vehicle data and simulated vehicle data, explained later. Additionally, this module is responsible for instantiating the 2D (SUMO) and 3D (CARLA) visualization interfaces.

Regarding the user interface, a desktop environment was developed using React with Electron. In this environment, users can, for example, insert simulated vehicles into the simulation during runtime. The request for creating simulated data is sent to a second Python module, which has a Flask API to receive the requests, and a sub-module responsible for creating the simulated data. The simulated data is then published to a local MQTT broker, and, as mentioned earlier, the main Python module consumes this data to insert it into the simulation via the TraCI API.

Additionally, we have two graphical interfaces in the architecture: the SUMO interface, representing the 2D environment, and the CARLA interface, representing the 3D environment. In a 3D simulation, the 2D environment is also present, and both interfaces communicate via sockets. Therefore, the 3D environment functions as an extension for a more realistic visualization.

Lastly, it's worth noting that the final architecture is the result of a series of evolutions that accompanied the project's development, being adapted to ensure proper functionality at every stage of this project's lifecycle.

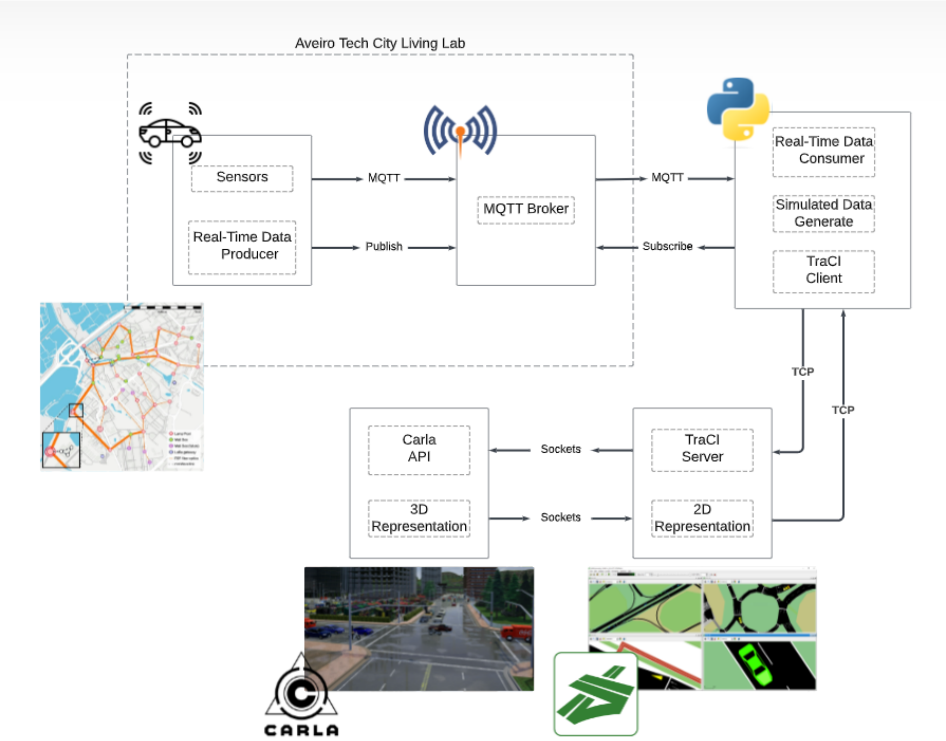


Fig. 5 - Architecture v1

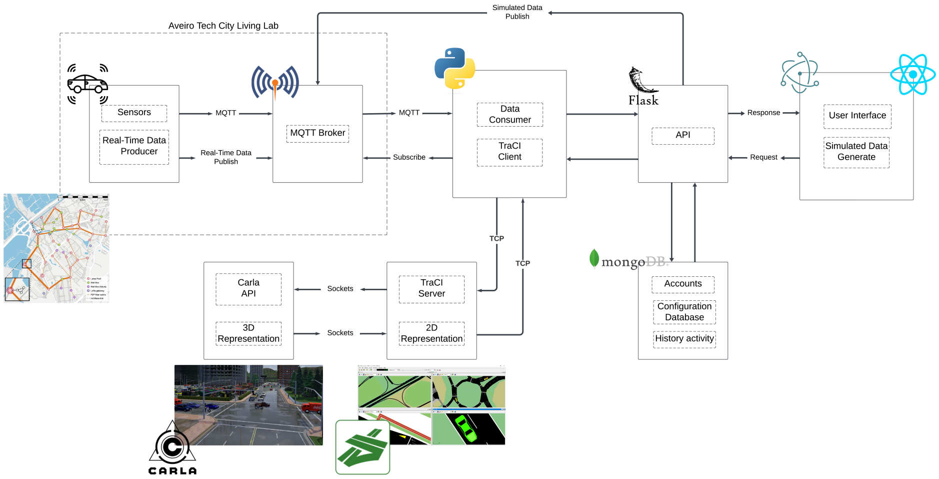


Fig. 6 - Architecture v2

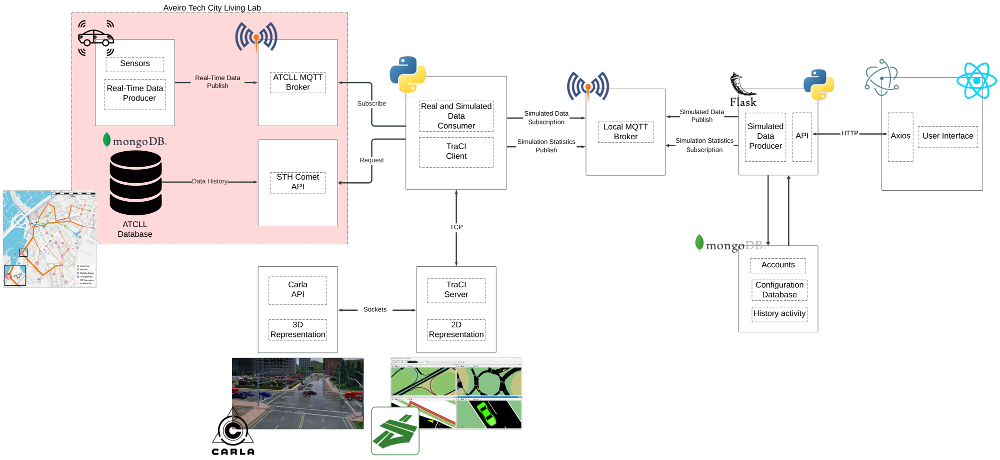


Fig. 7 - Architecture v3

4.5 Technology Stack

4.5.1 React & Electron

We chose React because it was the framework most familiar to every member of our team. Initially, we brainstormed the best options for our project, especially considering the need to launch SUMO and CARLA through the interface. We concluded that a web app wouldn't be the ideal choice since launching desktop applications like SUMO and CARLA through a web interface didn't make sense to us.

After thorough discussions, we determined that a desktop app would be the best and most logical solution. We explored various options, including Electron and Tauri. Electron emerged as the superior and most simple choice, allowing us to use HTML frameworks like React. Moreover, with Node.js, it provided a seamless development experience similar to developing a web app, which we were already accustomed to.

By leveraging React and Electron together, we could build a responsive, performant, and platform-independent desktop application. This combination enabled us to utilize our existing skills and streamline the development process, ensuring that we could efficiently implement the features required for our project.

4.5.2 Flask

We chose Flask because it was a framework our team was already familiar with. Flask provides a very simple and efficient way to build REST APIs, which was crucial for our project's back-end services. Its lightweight nature and flexibility allowed us to quickly set up and scale our application's server-side components.

Given the tight timeline for developing the entire solution, Flask allowed us to focus on more important tasks by minimizing the boilerplate code typically required for setting up the environment. This efficiency enabled us to concentrate on implementing core features related to the Digital Twin (system features) rather than spending time with the set-up.

Flask also provides a range of extensions and dependencies that are essential for our needs. It facilitated seamless communication with our database of choice (MongoDB) and helped in setting up RESTful APIs efficiently.

4.5.3 MongoDB

MongoDB's flexibility and ease of use made it a natural choice for our project's data storage needs, allowing us to store the user accounts and some previous simulations if the user pretends to. Its JSON-like document structure allows for efficient data querying and indexing.

Additionally, MongoDB was already being used by the Aveiro Tech City Living Lab to persist data received from sensors over time in the cloud. This existing integration made our team more inclined to use it, as it ensured compatibility and allowed us to leverage the established infrastructure.

4.5.4 Mosquitto

We chose Mosquitto as our message broker to facilitate communication between the API and the adapters that handle task adjustments and simulation updates. Mosquitto's implementation of the MQTT protocol makes it ideal for lightweight, low-bandwidth communication, ensuring that messages are delivered promptly and reliably.

The Aveiro Tech City Living Lab also used Mosquitto in an example setup to retrieve data from sensors across the city. Given this existing use case, it made sense to leverage the same technology for our project. By using Mosquitto for both sending tasks from the API to the adapters and receiving real-time data, we maintained consistency in our messaging infrastructure. This approach simplified our development process and made it easier for our team to understand and manage the communication systems within our project.

5 Construction Phase

5.1 What have we done

5.1.1 Generation of 2D and 3D Maps:

The first step is importing the OSM (OpenStreetMap) file, which contains the geographical data of the desired map. This OSM file serves as the foundation for all subsequent transformations. Once the OSM file is obtained, it undergoes a cleaning process utilizing JOSM (Java OpenStreetMap Editor). During this phase, all elements not recognized by SUMO and CARLA are removed. This step is critical as it ensures the resulting OSM file is compatible with both simulators, eliminating unnecessary data that could cause issues during conversion.

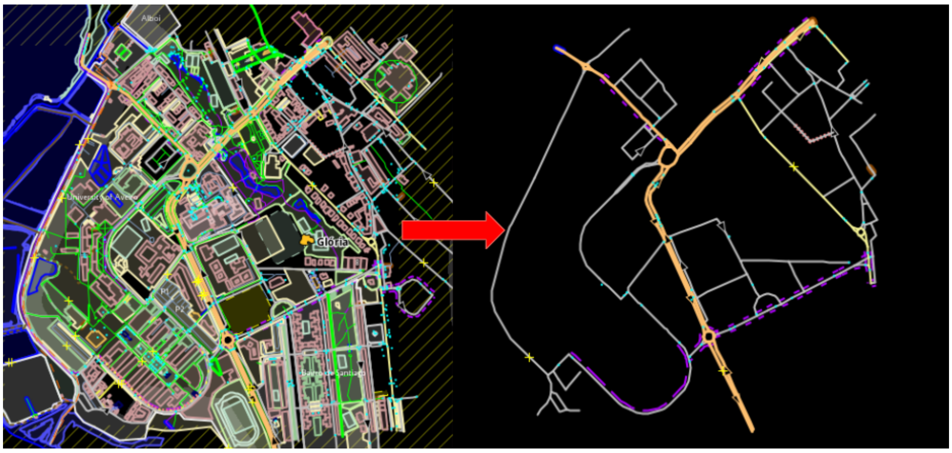


Fig. 8 - Map Cleanup Using JOSM

After cleaning the OSM file, we use the ”main.py” script located in the ”osmtoxdor” folder. This script performs two important functions: it generates an XDOR header that guarantees correct georeferencing in CARLA and creates a new modified OSM file. This new OSM file is specifically adjusted for use with CARLA, incorporating the necessary modifications for compatibility.

With the modified OSM file in hand, we move on to the XDOR format conversion stage. We use the ”converter.py” script, which utilizes the newly generated OSM file from the previous step, to create the final XDOR file. This XDOR file includes the previously generated header, ensuring all necessary georeferencing data is present.

Finally, to generate the SUMO network, we use a script located in the ”xdortoNetXml” folder. This script converts the final XDOR file into a traffic network usable by SUMO. The generated network represents the final map configuration, ready for use in traffic simulations within SUMO.

This transformation process, from the initial importation of the OSM file to the creation of the SUMO network, ensures that the maps are correctly prepared for integration and use in the SUMO and CARLA simulators, providing a robust platform for realistic and accurate traffic simulations.

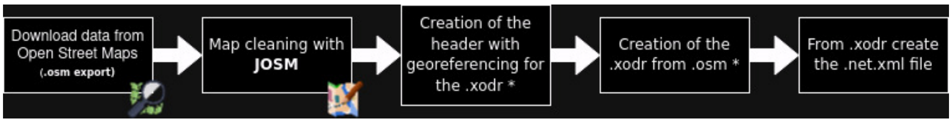

Download data from Open Street Maps
(.osm export)
Map cleaning with JOSM
Creation of the header with georeferencing for the .xodr *
Creation of the .xodr from .osm *
From .xodr create the .net.xml file

Fig. 9 - Map Generation Workflow

5.1.2 Adapters

The adapters are crucial to the development of our Digital Twin, concentrating on the implementation of the system's core functionalities. Developed as a key component of our project, these adapters enable the entire system to operate smoothly and allow users to perform traffic simulations and manage them.

The adapters are essential for running and co-simulating the SUMO and CARLA environments. This integration is pivotal as it combines the strengths of both systems, with SUMO handling detailed 2D traffic simulations and CARLA providing realistic 3D visualizations and advanced vehicle dynamics. To ensure continuous operation, the adapters implement an infinite loop that keeps the simulation active and responsive to real-time inputs.

A key function of the adapters is managing real-time data and user interactions. They subscribe to a local MQTT broker, receiving real-time change requests that enable dynamic modifications within the simulation. For example, when a user requests to insert simulated vehicles, block roads, or test specific scenarios, the adapters process these requests and communicate the necessary changes to the simulation environments using the TraCI API.

The development using the TraCI API was a major component of our project. This API allows the adapters to communicate with the simulation environments, ensuring that changes are accurately reflected in real-time. Whether it's adding new vehicles, blocking roads, or maintaining the integrity of simulation states, the TraCI API ensures that these operations are carried out efficiently.

In essence, the development of the adapters was the most substantial part of our project. They form the foundation that enables the entire system to function, providing users with the tools they need to simulate and manage urban traffic scenarios. Without the adapters, the integration of SUMO and CARLA and the dynamic capabilities of our Digital Twin would not be possible.

Additionally, it is noteworthy that three types of adapters have been developed, each for a specific type of simulation, which have distinct characteristics beyond the functionalities mentioned.

Types of adapters:

- **Live Data** - This adapter has a specific functionality responsible for connecting to the MQTT broker of the Aveiro Tech City Living Lab, receiving data collected by sensors, and adding it to the digital twin in real time.
- **Historic Data** - This adapter is designed to simulate real city traffic from a specific day and time, using data persisted in a database provided by the Aveiro Tech City Living Lab (ATCLL). It integrates actual traffic patterns into the simulation, allowing users to analyze and manage real-world scenarios effectively.
- **Simulated Data** - Considered the simplest, this adapter is responsible only for simulations with simulated vehicles.

TraCI API:

Another key functionality implemented in all adapters was the instantiation of the TraCI API (Traffic Control Interface), which enabled interaction with real-time traffic simulation. TraCI uses a TCP-based client/server architecture to provide access to SUMO, allowing for various operations such as defining routes for vehicles, modifying simulation parameters, obtaining information about the current traffic state, and much more.

SUMO-CARLA Co-simulation:

One more crucial responsibility of the adapters, especially those handling 3D simulations, was to facilitate the execution of a co-simulation between SUMO and CARLA, enabling all changes sent by TraCI to SUMO to be reflected in the 3D environment executed by CARLA. The implementation of co-simulation involves several steps, including:

- **Initial setup** - Firstly, it's necessary to configure both SUMO and CARLA instances to be ready to communicate with each other. This involves configuring network settings, such as communication ports and IP addresses.
- **Vehicle type configuration** - Since the two software platforms map vehicles differently (SUMO focuses more on vehicle classes - passengers, emergency, among others - while CARLA focuses more on the specific type of car to map which image should appear in the 3D environment), it's important to establish a correspondence between the vehicle types mapped by the software. For example, specifying that the vehicle identified by CARLA as "vehicle.audi.a2" is mapped by SUMO as "passenger".
- **Communication interface** - Instantiate the TraCI API responsible for manipulating traffic in the co-simulation.
- **Synchronization** - It's crucial to ensure that both simulators are synchronized during the co-simulation. This involves synchronizing simulation times, ensuring that actions in one simulator are properly reflected in the other.

Simulated Data Processing:

As previously mentioned, the adapters concentrate on the main components for the proper functioning of the Digital Twin. Among these components are the modules responsible for processing the simulated data and integrating it into the simulation, including:

- **Add a simulated car** - The *addSimulated* module is responsible for introducing a vehicle (car, motorcycle, or bicycle) with a defined route into the simulation. From two points, initial and final, selected on the interface map, the user can insert a virtual vehicle with a well-defined route. This module starts by using the *traci.simulation.convertRoad* method to find the roads that best represent these points based on the start and end coordinates and the type of vehicle. Then, using the *traci.simulation.findRoute* method, the system applies the Dijkstra algorithm to find the optimal path between these points. Finally, after the route is successfully defined, the system uses a timestamp to create a unique ID for the vehicle and, through the *traci.vehicle.add* method, inserts the vehicle into the system and makes it follow the defined route.

- **Add random traffic** - The module responsible for inserting random traffic, *addRandomTraffic* takes the desired number of cars as a parameter from the user. From there, it scans the edges (streets) to filter those allowing vehicle traffic and queries to find all recognized vehicle types. With this information, the system executes a certain number of cycles equal to the desired number of cars. Randomly, it selects the vehicle type, initial and final edges, and inserts the vehicle with random type and behaviour again using the method *traci.vehicle.add*".

- **Block road** - The module responsible for road blocking can be described as a method that first accesses information about which edges form a given street from the "road.json" file. Then, the module iterates through all the edges of the street being blocked and sets the maximum speed of these edges to 0. This manipulation occurs because of the algorithms utilized by the traffic system, which rely on costs associated with edges to determine routes. This approach ensures that vehicles will not use the blocked road segments for navigation, effectively simulating a road closure scenario within the traffic simulation environment. Additionally, it's worth noting that the simulation must not include any vehicles during the road closure period. This is because vehicles in SUMO have predefined routes, so they could still attempt to use the blocked road, resulting in inaccurate simulation outcomes.

- **Block roundabout** - The snippet of code responsible for blocking roundabouts bears a striking resemblance to that of blocking roads. Similarly, it accesses an external file to map out which edges constitute the roundabout slated for closure. Again, as a result of the traffic algorithms employed, it sets the maximum speed for these edges to zero, effectively preventing vehicles from traversing through them. Similar to the "Block Road" feature, no vehicles can be present in the simulation to block roundabouts.

5.1.3 Flask API

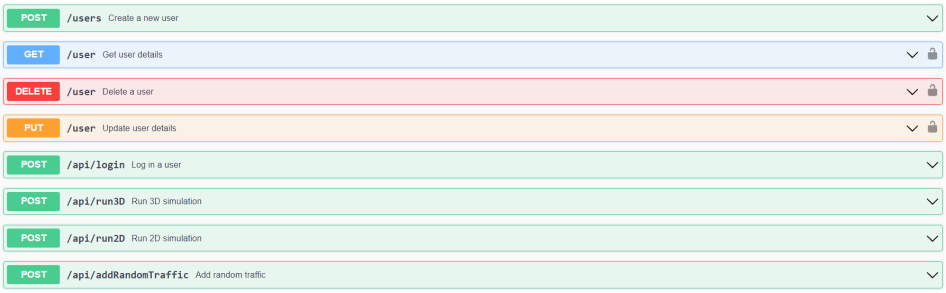
Constructed upon the robust foundation of the Flask microframework, the API emerges as a pivotal conduit for communication between the intricate simulation system and the Electron-based desktop application. The inherent adaptability and efficiency of Flask play a critical role in enabling the seamless deployment of an architectural framework that adeptly addresses the multifaceted challenges associated with data management and manipulation within the context of large-scale simulations.

Communication Endpoints: A Core Component At the heart of the API's functionality lies a meticulously designed suite of RESTful endpoints. These endpoints are strategically crafted to encourage and facilitate targeted interactions with the simulation system. By supporting a comprehensive array of Create, Read, Update, and Delete (CRUD) operations on data, these endpoints ensure a seamless and precise exchange of information between the Electron application and the server. This interoperability is paramount in maintaining a dynamic and responsive system capable of adapting to the evolving demands of real-time simulations.

Enhanced Data Handling Through Integration Furthermore, the API's integration with Flask-PyMongo introduces a level of sophistication that significantly enhances the API's ability to interact directly and efficiently with MongoDB databases. This integration unlocks the potential for the manipulation of vast quantities of data with unparalleled speed and precision. The result is a highly performant system that can effectively manage and manipulate large volumes of data, thereby supporting the sophisticated requirements of modern simulation systems.

By leveraging the strengths of Flask and integrating advanced technologies such as Flask-PyMongo, the API stands as a testament to the power of modern web frameworks in addressing the complex needs of large-scale simulations. Its design and functionality underscore the importance of flexible, efficient, and scalable solutions in the realm of data-intensive applications.

The API provides a series of RESTful endpoints, designed to facilitate specific interactions with the simulation system. These endpoints support CRUD operations on data, ensuring a fluid and accurate interaction between the Electron application and the server. Additionally, integration with Flask-PyMongo promotes direct and efficient interaction with MongoDB databases, allowing for the manipulation of large volumes of data with high performance.



POST `/users` Create a new user

GET `/user` Get user details

DELETE `/user` Delete a user

PUT `/user` Update user details

POST `/api/login` Log in a user

POST `/api/run3D` Run 3D simulation

POST `/api/run2D` Run 2D simulation

POST `/api/addRandomTraffic` Add random traffic

Fig. 10 - Swagger REST API Documentation

5.1.4 User Management Endpoints in the Flask Application

The Flask application incorporates several RESTful endpoints designed to manage user accounts, ensuring secure and efficient interactions with the underlying database. These endpoints cater to the creation, retrieval, authentication, deletion, and modification of user profiles, playing a vital role in the overall user experience and system security.

User Registration (POST /users) The registration endpoint facilitates the creation of new user accounts. Upon receiving a request with a username, email, and password, the endpoint validates these inputs and securely stores them in the database after hashing the password. This process not only ensures the privacy of user passwords but also sets the groundwork for future user interactions with the system.

User Authentication (POST /api/login) Authentication is a cornerstone of secure application usage. The login endpoint verifies user credentials by comparing the submitted password with the hashed version stored in the database. Successful authentication grants the user an access token, which is then used for subsequent requests requiring authorization. This mechanism ensures that only authenticated users can access protected resources.

User Profile Retrieval (GET /user) Retrieving user profile information is a common operation that requires authentication to prevent unauthorized access. The endpoint decodes the access token included in the request to identify the user. If the token is valid, the endpoint retrieves the corresponding user profile from the database and returns it, ensuring that users can view and manage their own information securely.

User Account Deletion (DELETE /user) Deleting a user account is a sensitive operation that should be performed securely. The endpoint authenticates the request using an access token, ensuring that only the account owner or an authorized administrator can initiate the deletion process. Upon confirmation, the user's account is removed from the database, permanently deleting the user's information from the system.

User Profile Update (PUT /user) Updating user profiles is a frequent requirement that must be handled carefully to maintain data integrity and security. The endpoint supports updates to a user's username and email, requiring authentication through an access token. This ensures that users can safely update their personal information without risking unauthorized changes.

These endpoints collectively form a robust user management system within the Flask application. They provide a secure and intuitive way for users to interact with their accounts, from initial registration to ongoing profile maintenance. The use of access tokens for authentication and authorization across these endpoints underscores the application's commitment to security and user privacy.

Configuration and Launch of the Local MQTT Broker

The API also plays a crucial role in configuring and launching a local MQTT broker. This protocol, ideal for the context of real-time simulations and IoT systems, is known for its lightness and effectiveness. The API ensures the correct definition of topics and efficient management of message flows, which is fundamental for the synchronization of data produced and consumed by various components of the system.

5.1.5 STH Comet API integration with the Flask API

The Flask API serves as a critical bridge between the simulation system and the Electron desktop application. Its connection with the STH Comet API is vital, as it plays a crucial role in the collection and manipulation of historical sensor data. This capability is essential for conducting simulations based on real and historical data, making it foundational for accurate and effective simulations in smart city and IoT development environments.

REST
hapi
mongoDB

Fig. 11 - STH Comet API architecture

Historical Data Retrieval

Through endpoints defined in the Flask API, requests can be made to the STH Comet API to fetch specific historical data. This data is used to fuel simulations that require contexts based on past events or conditions. Such a mechanism is crucial for trend analysis and the conduct of predictive simulations.

The provided code snippets demonstrate two main endpoints within a Flask application that interact with the STH Comet API to fetch data based on specific parameters. These endpoints are /api/run3D and /api/run2D, which are designed to initiate simulations either in 3D or 2D mode, respectively. The choice between running a simulation in real-time (live) or using historical data (real data) is determined by query parameters passed to these endpoints.

Endpoint: `/api/run3D`

This endpoint accepts POST requests and initiates a 3D simulation. It checks for two query parameters: live and real data. If live is set to True, it starts a Carla simulation in real-time mode. If real data is set to True, it fetches historical data from the STH Comet API for each entity specified in the request payload, processes this data, and then initiates a Carla simulation using this historical data.

Endpoint: `/api/run2D`

Similar to the 3D simulation endpoint, this endpoint also accepts POST requests and initiates a 2D simulation. It checks for the same query parameters (live and real data). Depending on these parameters, it either starts a 2D simulation in real-time mode or fetches historical data for the specified entities and uses this data to initiate a 2D simulation.

5.1.6 Fetching Data from STH Comet API

Both endpoints utilize a common function, `fetch_sth_comet_data`, to retrieve data from the STH Comet API. This function constructs a GET request to the STH Comet API's endpoint, passing along necessary headers and parameters such as the type of entity, the number of recent records (`lastN`), and the date range (`dateFrom` and `dateTo`). The response from this API call is then processed and returned to the calling function.

Data Workflow

When a request is initiated through the Electron application (e.g., for a 3D or 2D simulation using real data), the Flask API processes the provided arguments (such as `live` or `real_data`) and makes the appropriate calls to the STH Comet API. Historical data is then collected within specified time intervals (`start_time` and `end_time`), which are subsequently used to configure and execute the simulation.

Requests to the STH Comet API are set up to return the necessary data based on the type of entity and specified time interval, ensuring that only relevant data is retrieved. This method guarantees efficiency and relevancy in the data used for each simulation.

Response Handling and Simulation

After receiving the data from the STH Comet API, the Flask API proceeds to process this data, preparing it for use in simulations. Depending on the input parameters (live data or real historical data), different simulation scripts are triggered. Each script is responsible for setting up the corresponding simulated environment, whether in 3D or 2D, and starting the simulation with the retrieved data.

Error Management and Feedback

In the event of errors during the request to the STH Comet API or during data processing, the Flask API is equipped to handle these exceptions and provide appropriate feedback to the user. This robustness is crucial for maintaining the reliability of the simulation system and ensuring that users can trust the accuracy and efficacy of the simulations conducted.

5.2 Desktop Application

The desktop application was developed with the primary goal of providing an intuitive and powerful interface for traffic simulation and analysis. This application allows users to create, manipulate, and analyze traffic scenarios in both 2D and 3D visualizations. Users can add vehicles and pedestrians, set specific routes, block roads or roundabouts, and observe the impact of these actions in real-time.

The application is designed to enhance user experience by offering a seamless and responsive interface, facilitating complex simulations through simple and accessible interactions.

5.2.1 Directory Structure

The front-end architecture of our desktop application is organized in a way that promotes modularity and maintainability. The main directories and their purposes are as follows:

- **src:** Contains all the source code for the application.
  - **components:** Reusable React components that form the building blocks of the user interface.
  - **pages:** Main pages of the application, each representing a distinct view or functionality.
  - **assets:** Static resources such as images, icons, and styles.
- **index.js:** The entry point of the application, where the root component is rendered.
- **App.js:** The root component that sets up routing and global layout.
- **package.json:** Lists the project dependencies and scripts for building and running the application.

5.2.2 User Interface Overview

Home Page

The ‘HomePage.js‘ page serves as the landing page for the application. It introduces users to the Aveiro Digital Twin project and provides options to start simulations in both 2D and 3D formats. The component imports React and the ‘Tooltip‘ component from ‘react-tooltip‘ to enhance user interaction, particularly to remind users that SUMO and CARLA need to be installed for simulations to run.

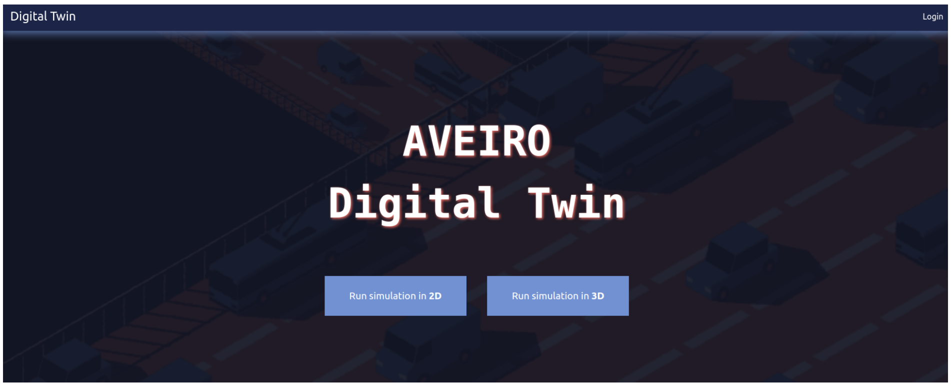


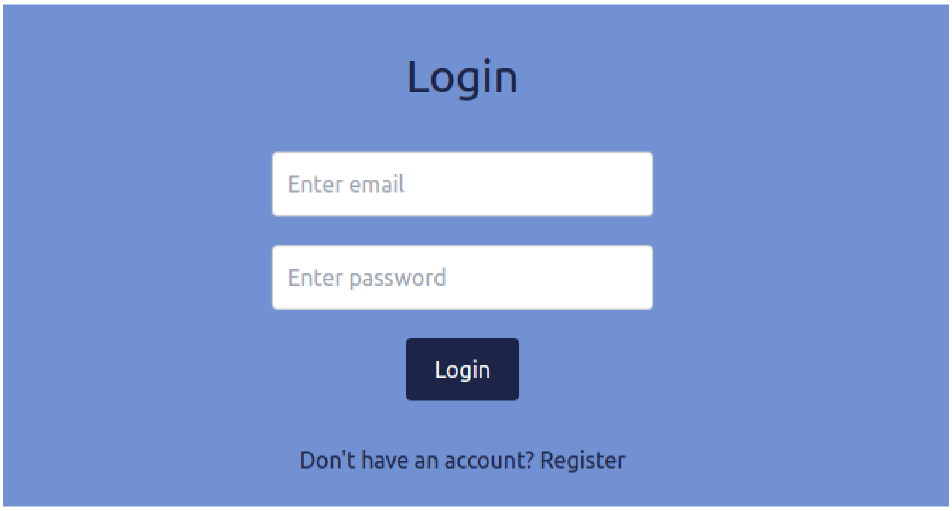
Fig. 12 - Home Page

Login and Register Pages

The ‘Login.js‘ and ‘Register.js‘ pages handle user authentication and registration within the application.

The ‘Login.js‘ page allows users to log in to the application. It uses local state to store the user’s email and password credentials and displays error messages if the credentials are invalid. Upon successful login, the access token is stored in sessionStorage and the user is redirected to the home page. If a non-logged-in user attempts to access a simulation, they are redirected to the login page with a message indicating that login is required.

The ‘Register.js‘ page allows users to register for the application. It uses local state to store the username, email, and password of the user. Basic validation is performed to ensure all fields are properly filled out, including a valid email and a password of at least 6 characters. Upon successful registration, the user is redirected to the login page. Otherwise, an error message is displayed indicating that registration failed.



Login

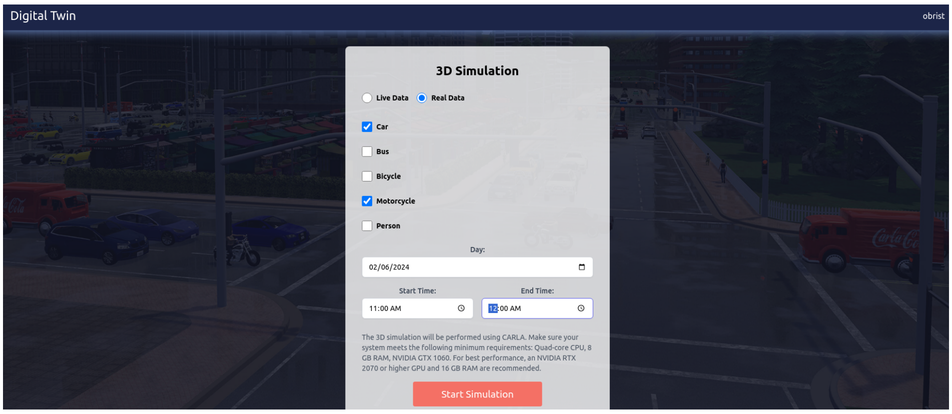
Enter email

Enter password

Fig. 13 - Log In Page

Run 2D and 3D Pages

The ‘Run2DPage.js‘ and ‘Run3DPage.js‘ are designed to provide a cohesive user experience for initiating traffic simulations in both 2D and 3D environments. They achieve this through the use of integrated forms, which allow users to select data sources such as live data, historical data, or simulated data. When the user selects historical data, they are required to specify the day, start time, and end time for the simulation. The program disables the option for users to select days that have not yet occurred, ensuring realistic simulation conditions.



3D Simulation

○ Live Data ● Real Data

☑ Car

□ Bus

□ Bicycle

☑ Motorcycle

□ Person

Day: 02/06/2024

Start Time: 11:00 AM

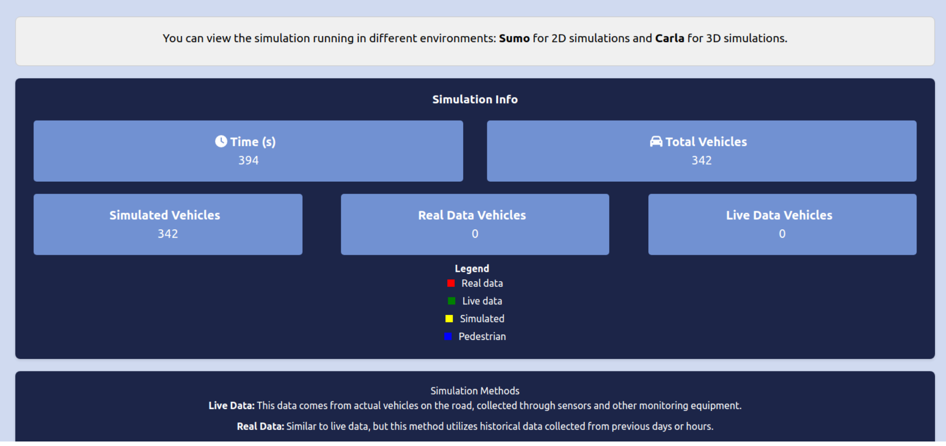
End Time: 12:00 AM

The 3D simulation will be performed using CARLA. Make sure your system meets the following minimum requirements: Quad-core CPU, 8 GB RAM, NVIDIA GTX 1060. For best performance, an NVIDIA RTX 2070 or higher GPU and 16 GB RAM are recommended.

Fig. 14 - Run 3D Page

Simulation Page

The ‘SimulationPage.js‘ page is responsible for displaying detailed information about traffic simulation in both 2D and 3D environments, using data received from the server via WebSocket. Upon receiving data from the server, the component displays information such as simulation time, total number of vehicles, the distribution among simulated vehicles, historical data, and live data. The information is categorized by vehicle type, including simulated cars, historic data, and live data.



You can view the simulation running in different environments: **Sumo** for 2D simulations and **Carla** for 3D simulations.

Simulation Info

Time (s) 394

Total Vehicles 342

Simulated Vehicles 342

Real Data Vehicles 0

Live Data Vehicles 0

Legend

🟥 Real data

🟩 Live data

🟨 Simulated

🟦 Pedestrian

Simulation Methods

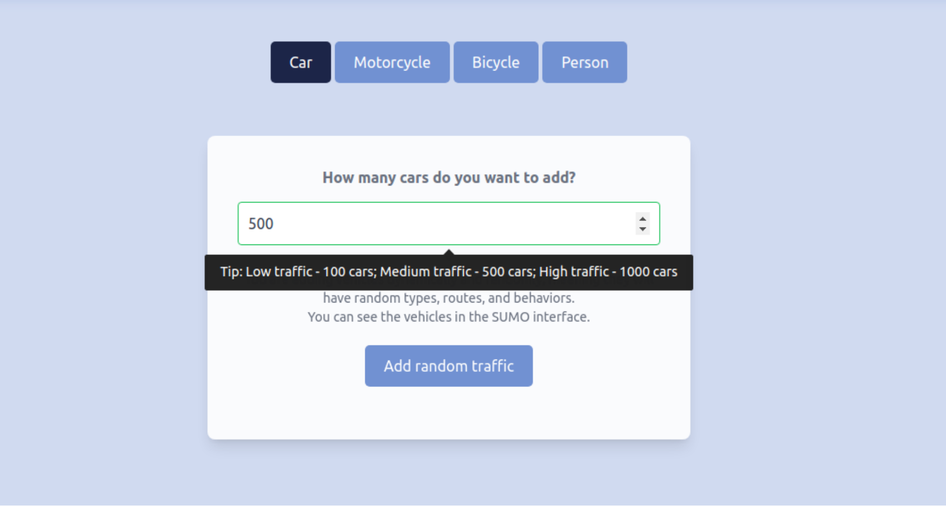
Live Data: This data comes from actual vehicles on the road, collected through sensors and other monitoring equipment.

Real Data: Similar to live data, but this method utilizes historical data collected from previous days or hours.

Fig. 15 - Simulation Page

Add Random Page

The ‘AddRandom.js‘ page is designed to enable users to add various types of entities dynamically to the traffic simulation environment, each with randomized routes. The user interface includes multiple components, each representing different types of entities that can be added: cars, motorcycles, bicycles, and pedestrians. Users only need to specify the quantity they want to insert for the chosen entity type and all these entities will be visible in the simulation environments.



How many cars do you want to add?

500

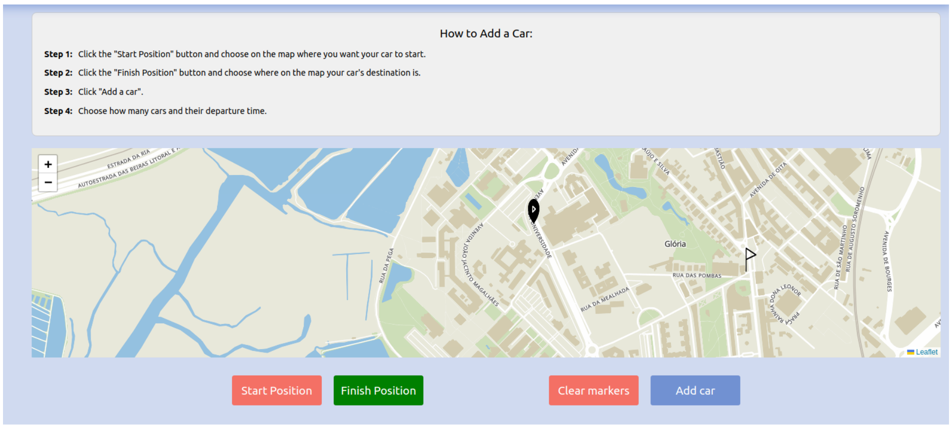
Tip: Low traffic - 100 cars; Medium traffic - 500 cars; High traffic - 1000 cars

have random types, routes, and behaviors.
You can see the vehicles in the SUMO interface.

Fig. 16 - Add Random Page

Add Car, Motorcycle, Bike and Pedestrian Pages

The ‘CarPage.js‘, ‘MotorcyclePage.js‘, ‘BicyclePage.js‘, and ‘PedestrianPage.js‘ are designed to facilitate the addition of vehicles and pedestrians to the traffic simulation environment with specific routes. To achieve this, users interact with the map interface where they select both the starting and ending positions for their chosen vehicle or pedestrian. Once the start and end positions are marked, users can submit the vehicle’s route. This process involves constructing a vehicle or pedestrian object with the specified positions and sending it to a back-end service for processing. The component handles both single-position submissions (for scenarios where only the start or end position is known) and dual-position submissions (where both positions are specified). Additionally, users specify the quantity of vehicles they wish to simulate with that route and set the departure time, which refers to the time when the vehicle or pedestrian begins its journey in the simulation.



How to Add a Car:

Step 1: Click the "Start Position" button and choose on the map where you want your car to start.

Step 2: Click the "Finish Position" button and choose where on the map your car's destination is.

Step 3: Click "Add a car".

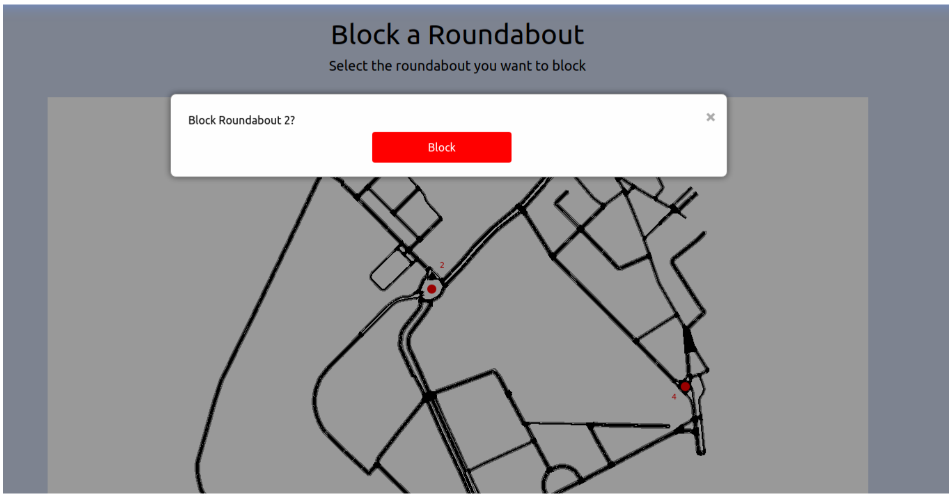
Step 4: Choose how many cars and their departure time.

Fig. 17 - Add Car Page

Block Road and Roundabout Pages

The ‘Block.js‘ page is responsible for allowing users to block and unblock segments of roads and roundabouts in the traffic simulation environment. It displays two sections on the page: ”Blocked Roundabouts” and ”Blocked Roads”. For each type of blocked segment, it iterates through the ‘blockedRoundabouts‘ and ‘blockedRoads‘ arrays (if they are defined) and renders a Card component for each segment. Each Card displays the segment ID and provides a button to unblock it.

In the ‘BlockRoad.js‘ and ‘BlockRoundabout.js‘ pages, a map is displayed with all the roads and roundabouts, respectively, where users can click to block those specific road segments and roundabouts.



Block a Roundabout

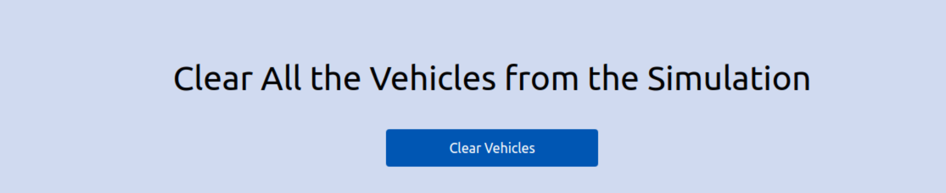
Select the roundabout you want to block

Block Roundabout 2?

Fig. 18 - Block Roundabout Page

Clear Simulation Page

The ‘ClearSimulation.js‘ page is responsible for allowing users to clear all vehicles from the traffic simulation environment. It consists of a single button labeled ”Clear Vehicles”. When clicked, this button triggers a request to the server to clear all vehicles from the simulation. This approach ensures that users can easily reset the simulation environment, clearing all vehicles and preparing for a new simulation run.



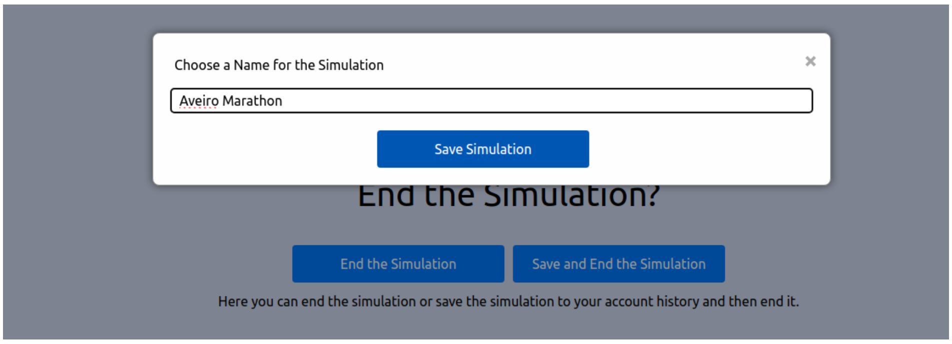
Clear All the Vehicles from the Simulation

Fig. 19 - Clear Page

End Simulation Page

The ‘EndSimulation.js‘ page provides functionality for ending and optionally saving a simulation in the traffic simulation environment. It features two main actions: ending the simulation without saving and saving the simulation with a specified name before ending it.

The ”End the Simulation” button immediately terminates the current simulation session when clicked. The ”Save and End the Simulation” button triggers the display of a modal dialogue where users can specify a name for the simulation before saving it. Once a name is entered and the ”Save Simulation” button within the modal is clicked, it saves the simulation with the provided name to the user’s account history and then ends the simulation session.



Choose a Name for the Simulation

Aveiro Marathon

End the Simulation?

Here you can end the simulation or save the simulation to your account history and then end it.

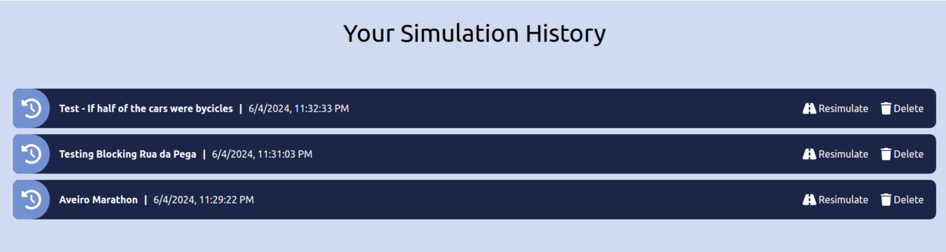
Fig. 20 - End Simulation Page

History Page

The 'History.js' page is designed to manage and display a user's simulation history, providing options to view details, re-simulate, and delete previous simulations.

Upon loading, the page sends an HTTP GET request to retrieve the user's simulation history. The history is then displayed in reverse chronological order, showing the most recent simulations first. Each simulation entry includes details such as the simulation name, date, and options to re-simulate or delete.

When clicking on the "Re-simulate" button of a specific simulation entry, the system checks if there is a simulation currently running. If not, a POST request is sent to initiate the re-simulation with the specified simulation ID.



Your Simulation History

Test - If half of the cars were bycicles | 6/4/2024, 11:32:33 PM

Testing Blocking Rua da Pega | 6/4/2024, 11:31:03 PM

Aveiro Marathon | 6/4/2024, 11:29:22 PM

Fig. 21 - History Page

5.2.3 Statistics

To equip users with valuable insights for informed decision-making, at the end of our project we decided to create statistics for the simulations if the user decides to save them in the history. These statistics are made using the TraCI API, a powerful Python library for interacting with SUMO. Through TraCI methods, the system gathers a range of key metrics during simulations. These metrics provide a comprehensive picture of traffic flow and environmental impact within the simulated city. The metrics captured are referent to portions of roads and roundabouts.

Here's a breakdown of the various statistics the system is capable of generating:

- **Average CO_2 Emissions:** This metric helps assess the environmental impact of different traffic scenarios. TraCI provides methods to access vehicle emission data, allowing us to calculate the average CO2 emissions.
- **Average Waiting Time:** Understanding wait times at intersections is crucial for optimizing traffic flow. TraCI offers methods to track vehicle positions and speeds, enabling us to calculate the average time vehicles spend waiting at specific locations, such as intersections or roundabouts.
- **Average Fuel Consumption:** This metric is another valuable indicator of environmental impact. TraCI provides access to vehicle fuel consumption data, allowing us to calculate the average fuel consumption.

- **Total and Maximum Number of Vehicles:** Tracking vehicle volume is essential for analyzing traffic density. TraCI offers methods to retrieve the total number of vehicles within a specific road segment or roundabout at any given time.

Utilizing these given statistics, the system is able to make graphs to compare these metrics between two simulations. For example, we can simulate a normal traffic flow on a Sunday in the city of Aveiro. Then, to simulate the impact that the "Aveiro's Europa Marathon 2024" will have on the city we make another simulation blocking the necessary roads and roundabouts. By obtaining the statistics of both simulations, the application can generate graphs on the same:

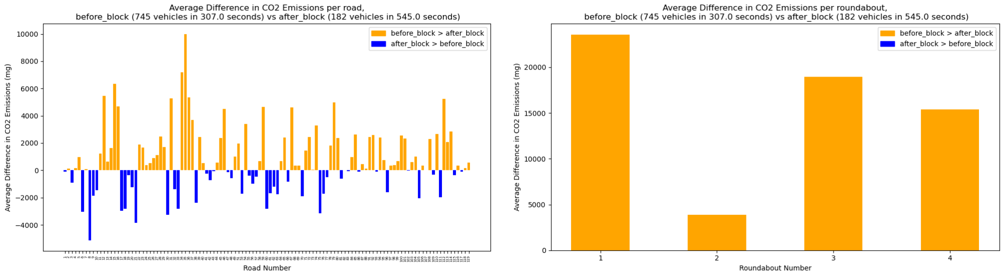


Fig. 22 - Road and Roundabout Average CO_2 Consumption Graphs

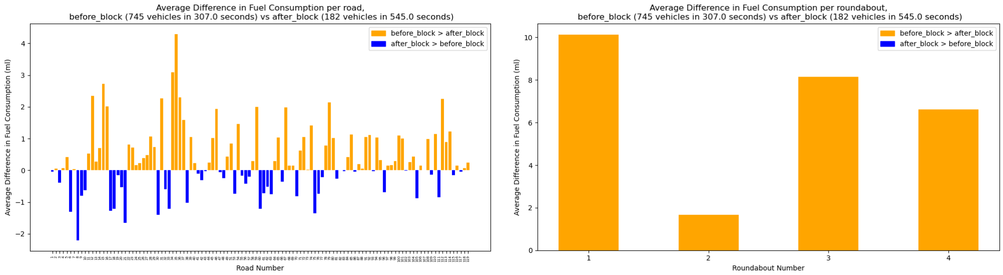


Fig. 23 - Road and Roundabout Average Fuel Consumption Graphs

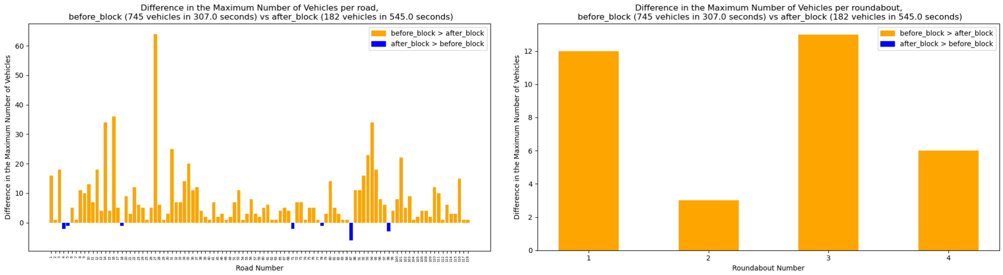


Fig. 24 - Road and Roundabout Maximum Number of Vehicles Graph

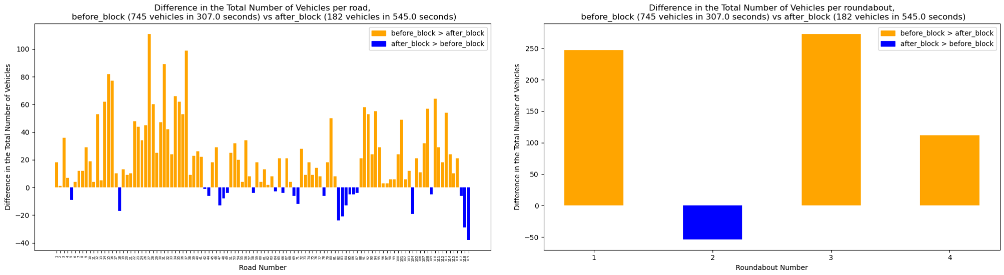


Fig. 25 - Road and Roundabout Total Number of Vehicles Graph

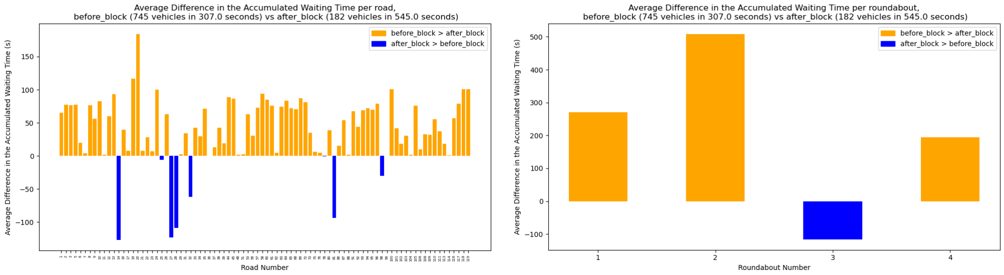


Fig. 26 - Road and Roundabout Average Waiting Time Graph

5.3 Fears and Difficulties

5.3.1 Fears

Fear

Limited Time for Task Completion The fear of insufficient time to accomplish all objectives poses a significant challenge, jeopardizing project deadlines and quality.

Our attempt to solve it

Implement efficient time management strategies, prioritizing tasks based on their importance and allocating resources judiciously.

Fear

Challenges in Incorporating Real and Simulated Data

How did we solve it

In conjunction with the capabilities of SUMO, the group managed to streamline interactions between real and simulated entities. Consequently, the data interaction occurs in a more natural and straightforward manner.

Fear
Aveiro city map Not Supported by CARLA Simulation The unavailability of support for a map so big as Aveiro within the CARLA simulation environment was a significant problem.

How did we solve it
Initially, our mapping efforts focused solely on Rua da Pega (Figure 7) due to upload errors with larger maps. Utilizing editing tools, we streamlined and lightened the map, enabling the successful integration of a larger Aveiro map (Figure 8) into CARLA.

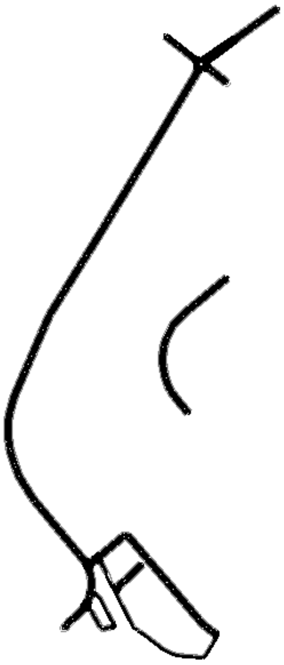


Fig. 27 - Our Rua da Pega Map

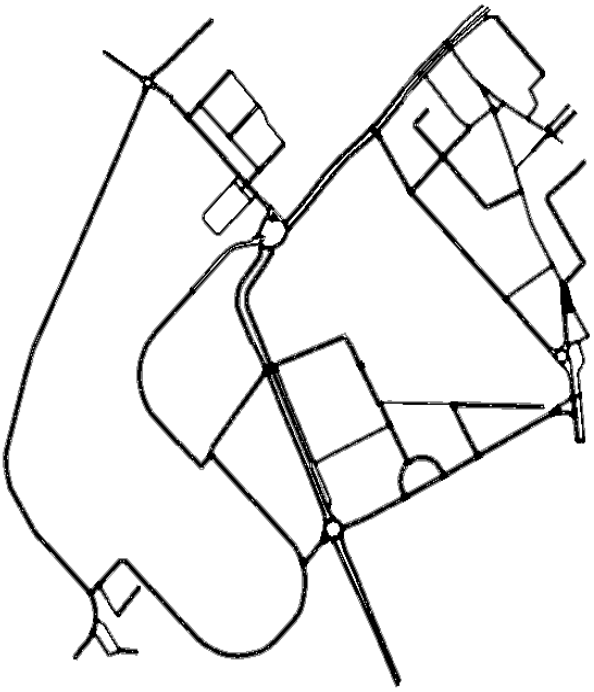


Fig. 28 - Our Aveiro Map

5.3.2 Difficulties

Difficulty: Autoware Integration
Due to recent developments, the group was advised to explore integrating the project with autonomous vehicles from Autoware. The objective was to incorporate an Autoware vehicle into our simulation to test certain behaviours and interactions between the vehicle, the simulation, and the real world.

A more advanced idea was to have the autonomous vehicle drive on real-world streets while reacting to conditions applied in the simulation. However, after investigation, the group concluded that such integration would not be an easy task, as it would require numerous changes, including modifications to our base map.

A more focused research effort would be necessary to address the various problems we identified, the main one being the need for a map compatible not only with SUMO and CARLA but now also with Autoware. The latter requires a point cloud file and a lanelet file. These two files would need to be somehow linked with the files used to create the map for SUMO and CARLA, a task that, in terms of scope, would still involve significant work and research.

Difficulty: People only in 2D

Another difficulty we faced was representing people in a 3D environment, not just in 2D. This challenge arises from the fact that on the CARLA side (3D), people are not defined as an entity type, making their representation not possible.

5.4 Changes in the initial plan

5.4.1 Architecture

During the presentation of our project idea, the professors expressed concerns about our architecture, fearing it didn't align with the course's emphasis on standard software development practices. To address this, we modified our architecture to conform to these standards, as shown in Figure 9.

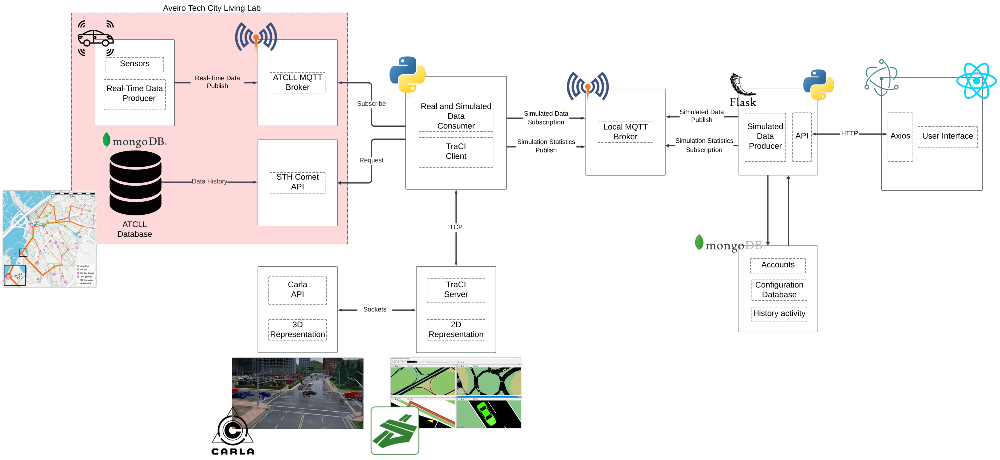


Fig. 29 - Project's Architecture

5.4.2 New User Interaction

As shown in the previous image (Figure 9), we had to adapt our architecture to include a new user interaction interface. This new component is a desktop application. This change required us to create several new components in the architecture.

An MQTT broker was introduced to handle subscriptions and the creation of new topics that will be shared between the TraCI API and our desktop-side API (Flask). This API will contain the endpoints that will be called by the user interface (Electron-React).

Additionally, a database component (MongoDB) was created to store data such as users, their passwords, simulation history, and more.

5.4.3 Types of Simulations

Another change in our plan involved altering the types of simulations possible, particularly concerning the use of real data. Due to issues with handling large volumes of high-velocity data, the group, along with the advisors, revised the initial plan. The use of real-time data was limited to one of the less busy sensors (the Rua da Pega sensor), making this simulation feasible only with the map of that area. The goal was to isolate the use of real-time data in a smaller environment to reduce the data processing load coming through the MQTT Broker from ATCLL.

Alongside this change, we introduced a new use of real data. By accessing ATCLL's database, we created a new feature: simulating with historical real data. This means we can now use real data from a past period to test simulations using previously collected real data.

6 Results

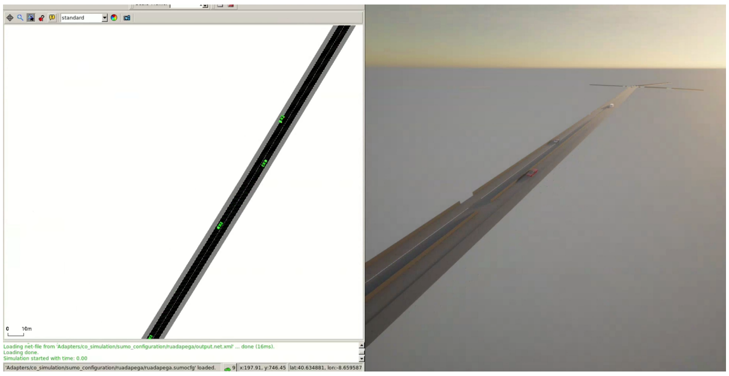
Overall, the main goals and requirements of our project were successfully met. We managed to build a responsive platform that enables users to simulate and assess various scenarios using both simulated and real-world vehicles. This platform serves as a powerful tool for urban managers and mobility researchers, allowing them to understand what will happen to the traffic when changes occur in the city and how to improve the mobility of the city with this information.

However, it's important to note that not all requirements were fully realized. Time constraints and a lack of detailed information on certain topics, such as integrating **Autoware** into our system, led to partial implementation of some features. Additionally, while we made a useful implementation in utilizing city-wide sensor data, there's room for improvements in terms of maximizing the use of the resources provided by the sensors across the city.

6.1 Co-Simulation between SUMO and CARLA

The initial phase of our project involved a comprehensive exploration of the SUMO and CARLA platforms. These platforms were a core component of the project, and understanding how they work and their functionalities was paramount to achieving our goals. However, this initial learning curve proved to be steeper than anticipated. While information on using these platforms individually was plentiful, resources on how to co-simulate them with custom maps were limited.

This lack of existing resources meant we had to delve deeper into the technical aspects of both platforms and experiment with different configurations. Nonetheless, through persistence and a commitment to problem-solving, we successfully established the co-simulation. This crucial achievement allowed us to visualize traffic flow in both 2D and 3D, fulfilling our initial objective for this phase and laying the groundwork for further analysis.



```
Loading net-file from 'Adapters/co_simulation/sumo_configuration/ruadapega/output.net.xml' ... done (16ms).
Loading done.
Simulation started with time: 0.00
```

```
'Adapters/co_simulation/sumo_configuration/ruadapega/ruadapega.sumocfg' loaded.   9   x:197.91, y:746.45   lat:40.634881, lon:-8.659587
```

Fig. 30 - Co-Simulation between SUMO and CARLA

6.2 Simulated Data

Our next challenge involved bringing the co-simulation to life by adding simulated vehicles to it. Research led us to TraCI, a Python API that integrates with SUMO, enabling it to launch an instance of the 2D interface and apply the available methods to it. This powerful tool granted us direct control over SUMO's functionalities within our Python code.

Using TraCI, we could dynamically add vehicles to the simulation and even manipulate their behaviour, such as adjusting their speed. The beauty of this approach lies in the seamless integration between SUMO and CARLA. Vehicles added to SUMO also appear within CARLA due to the co-simulation established before. This successful implementation marked a significant milestone, allowing us to finally visualize simulated cars through both SUMO and CARLA interfaces. We now had a dynamic and controllable virtual environment ready for further experimentation.

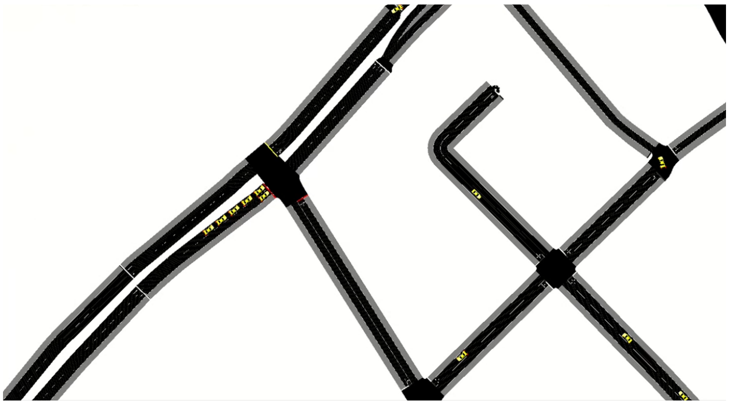
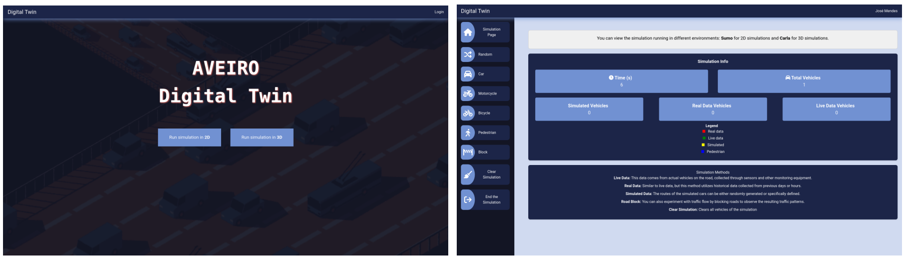


Fig. 31 - Visualization of Simulated Vehicles in SUMO

6.3 Desktop Application

Recognizing the inconvenient nature of launching and managing the individual SUMO and CARLA interfaces, we prioritized user experience by developing a dedicated desktop application. This decision originated from our desire to make the system more accessible for urban planners and mobility researchers, the target audience for our project.

Brainstorming sessions led us to conclude that a desktop application offered the most intuitive solution, especially since the co-simulation already resided on the user's PC. This application acts as a central hub, providing a simple and friendly interface for users to interact with and manage the simulated city environment.



You can view the simulation running in different environments: **Sumo** for 2D simulations and **Carla** for 3D simulations.

Simulation Methods

Live Data: This data comes from actual vehicles on the road, collected through sensors and other monitoring equipment.

Real Data: Similar to live data, but this method utilizes historical data collected from previous days or hours.

Simulated Data: The routes of the simulated cars can be either randomly generated or specifically defined.

Road Block: You can also experiment with traffic flow by blocking roads to observe the resulting traffic patterns.

Clear Simulation: Clears all vehicles of the simulation

Fig. 32 - Digital Twin Desktop Application

6.4 Live Data

Our project wasn't just about creating a system that allowed us to add simulated vehicles. A key goal was to bridge the gap between simulation and reality by incorporating real-world traffic data from Aveiro's city sensors into our digital twin. This integration proved crucial. It allowed for not only visualizing simulated traffic but also to interact with and analyze actual traffic flow data from the city.

This capability elevates the platform's power by enabling users to create simulations that are informed by real-world conditions, leading to more accurate and practical insights for urban planning and mobility research.

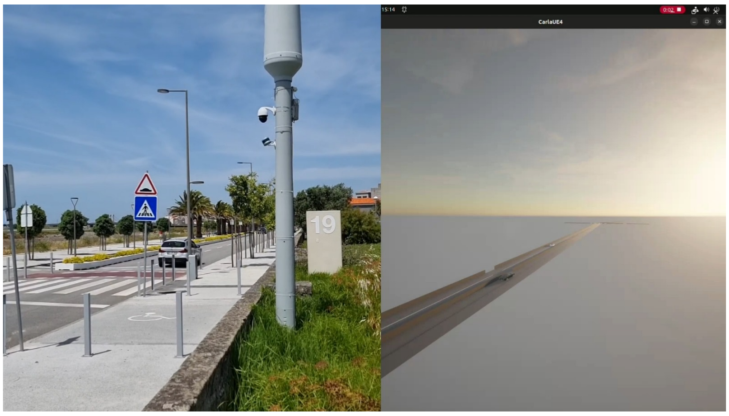
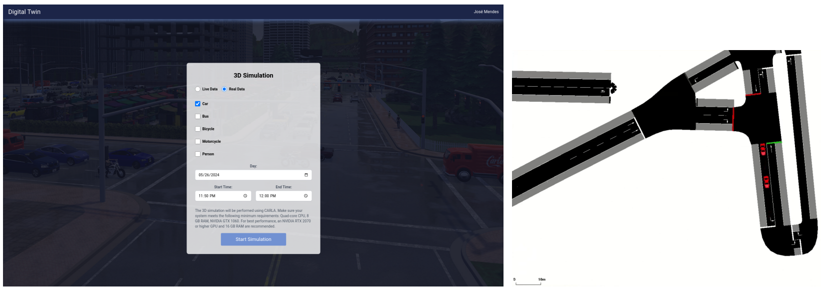


Fig. 33 - Live Data from the real world represented in the CARLA Interface

6.5 Historical Data

Our initial plan aimed to make an integration between real-time traffic data alongside simulated data. However, processing the high-speed sensor data streaming from Aveiro's city infrastructure proved challenging. The real-time sensor data was too much for our system to handle, so we came up with a new solution. We used historical traffic data stored in the cloud instead.

This solution allows users to select a specific date and time, essentially using real traffic data from previous days within their simulations. This approach not only preserved the value of real-world data integration but also offered a practical solution to the limitations of real-time processing.



3D Simulation

○ Live Data ● Real Data

☑ Car

□ Bus

□ Bicycle

□ Motorcycle

□ Person

Day: 05/26/2024

Start Time: 11:50 PM

End Time: 12:00 PM

The 3D simulation will be performed using CARLA. Make sure your system meets the following minimum requirements: Quad-core CPU, 8 GB RAM, NVIDIA GTX 1060. For best performance, an NVIDIA RTX 2070 or higher GPU and 16 GB RAM are recommended.

Fig. 34 - Historical Data Dashboard and Representation in SUMO

6.6 Interaction Between Real and Simulated Vehicles

A crucial aspect of our project involved ensuring smooth interaction between real and simulated data. We wanted to mirror real-world traffic, where collisions are avoided at all costs. To do so, our solution involved treating real vehicles, detected by sensors, as simulated vehicles within the platform. These "real-time simulated" vehicles adjust their speed based on actual radar data from the city. This gives them virtual awareness of all surrounding vehicles, both simulated and real.

While they aim to maintain their real-world speed, they are programmed to predict and avoid potential collisions by using emergency braking, thus ensuring no collisions occur within the simulation.

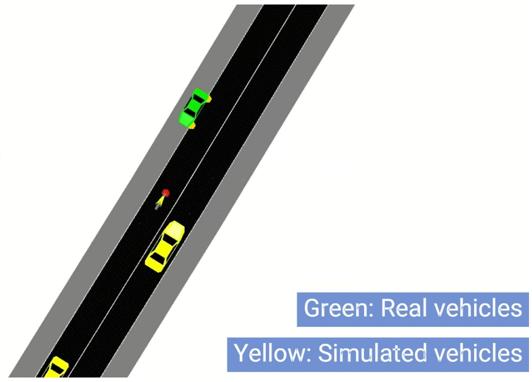


Fig. 35 - Interaction between real and simulated vehicles

6.7 Statistics and Graphs

To wrap up the project, our team focused on providing valuable tools for urban planners. The system generates data which allows us to make statistics about the simulations, like the number of vehicles on a specific road or total CO_2 emissions from a roundabout. But it goes beyond raw data, the platform allows users to create graphs comparing two simulations. Imagine comparing traffic flow before and after blocking a road. This "before and after" visualization empowers users to see the impact of changes on the city and make informed decisions about urban mobility.

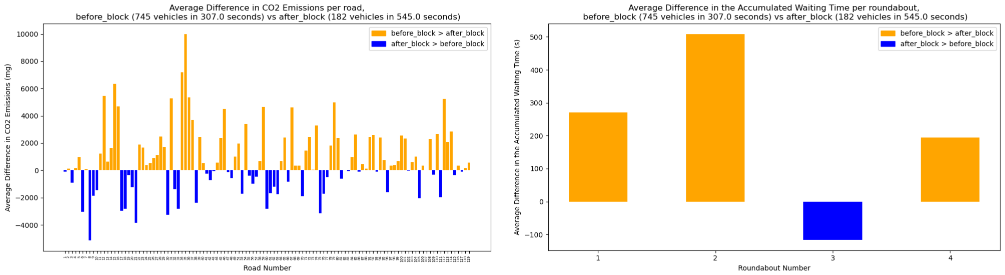
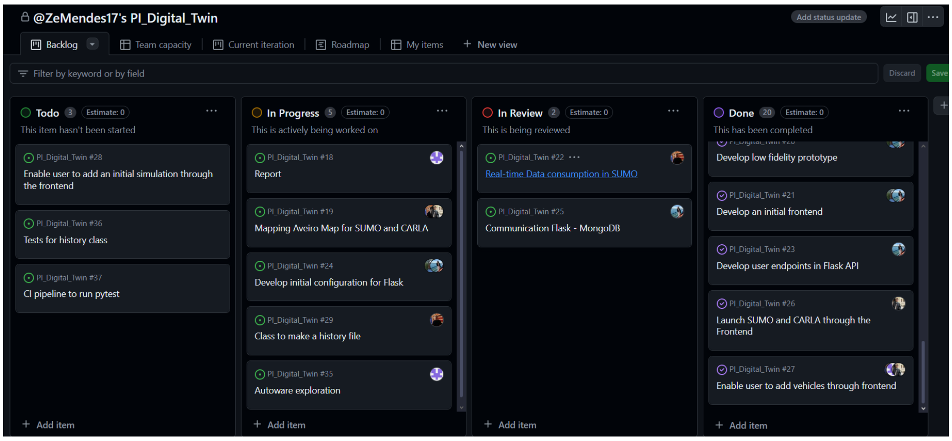


Fig. 36 - Generated graphs relative to Road CO_2 emissions and the waiting time to enter a roundabout, before and after block

7 Project Management

7.1 Project Management Tool

To efficiently track work tasks and quickly assign them to team members, we decided to use the project management tool **GitHub Projects**. This agile platform allows us to effectively manage tasks and issues related to the project while facilitating necessary planning. We can organize tasks, assign them to team members, and track issues through various stages of development. Additionally, since our team uses GitHub as our repository, having project management integrated into the same platform enhances convenience and streamlines our workflow.



Todo 3 Estimate: 0
This item hasn't been started

PI_Digital_Twin #28
Enable user to add an initial simulation through the frontend

PI_Digital_Twin #36
Tests for history class

PI_Digital_Twin #37
CI pipeline to run pytest

In Progress 5 Estimate: 0
This is actively being worked on

PI_Digital_Twin #18
Report

PI_Digital_Twin #19
Mapping Aveiro Map for SUMO and CARLA

PI_Digital_Twin #24
Develop initial configuration for Flask

PI_Digital_Twin #29
Class to make a history file

PI_Digital_Twin #35
Autoware exploration

In Review 2 Estimate: 0
This is being reviewed

PI_Digital_Twin #22
Real-time Data consumption in SUMO

PI_Digital_Twin #25
Communication Flask - MongoDB

Done 20 Estimate: 0
This has been completed

PI_Digital_Twin #20
Develop low fidelity prototype

PI_Digital_Twin #21
Develop an initial frontend

PI_Digital_Twin #23
Develop user endpoints in Flask API

PI_Digital_Twin #26
Launch SUMO and CARLA through the Frontend

PI_Digital_Twin #27
Enable user to add vehicles through frontend

Fig. 37 - Team's Github Projects

7.2 Development Workflow

To ensure that all developed code meets the requirements and does not break existing functionality, we adopted the GitHub Flow strategy. This strategy outlines a clear process for developing and integrating new features, ensuring a smooth and controlled workflow. Here's how we handle new user stories or features:

1. **Create a New Branch**

   We start by creating a new branch with a short, descriptive name. This provides a stable and isolated environment for development, preventing interference with ongoing work.

2. **Implement the Pretended Changes in the Branch**

   All development work related to the new feature or task is done within this branch, ensuring that changes are contained and manageable.

3. **Submit a Pull Request**

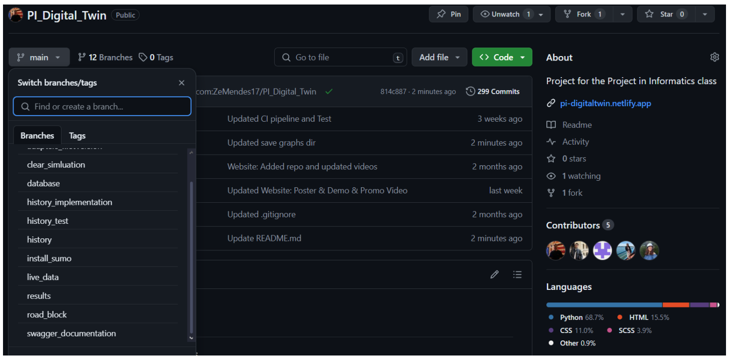
   Once development is complete, a pull request is submitted to the "main" branch. The pull request includes a clear title and a brief description of the changes.

4. **Review and Approval**

   Another team member reviews the pull request and either approves or disapproves it. This peer review process helps ensure code quality and functionality.

5. **Merge the Pull Request**

   Once the pull request is approved, it is merged into the ”main branch.



clear_simluation
database
history_implementation
history_test
history
install_sumo
live_data
results
road_block
swagger_documentation

com:ZeMendes17/PI_Digital_Twin ✓ 814c887 · 2 minutes ago 299 Commits

Updated CI pipeline and Test 3 weeks ago
Updated save graphs dir 2 minutes ago
Website: Added repo and updated videos 2 months ago
Updated Website: Poster & Demo & Promo Video last week
Updated .gitignore 2 months ago
Update README.md 2 minutes ago

About

Project for the Project in Informatics class

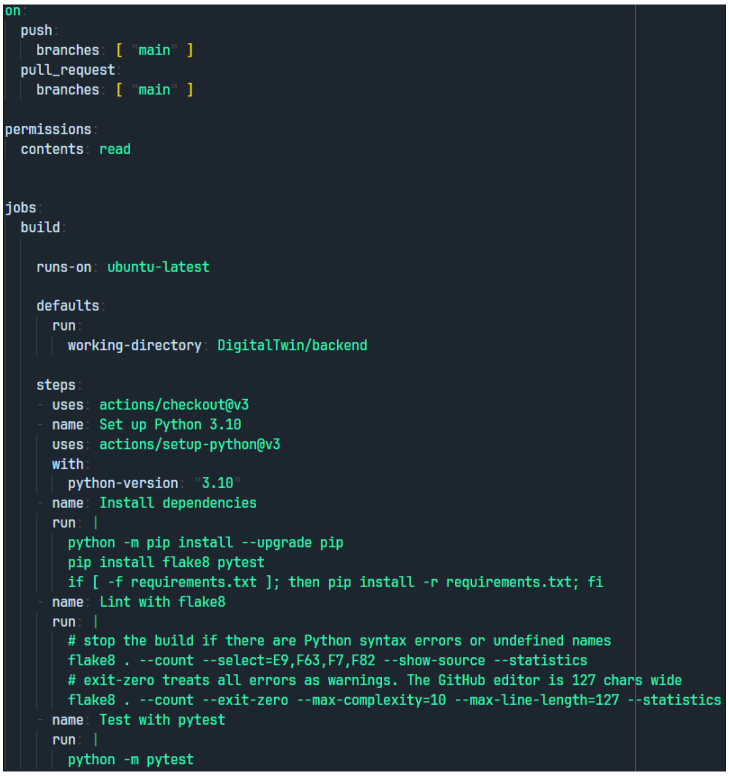
pi-digitaltwin.netlify.app

Fig. 38 - Github - Branches Example

7.3 CI/CD Pipelines

Continuous Integration and Continuous Delivery/Deployment are crucial practices in modern software development. They enable teams to integrate changes frequently, test comprehensively, and deploy updates swiftly. This approach not only accelerates the release of new features but also ensures that bug fixes and improvements are delivered efficiently and reliably, maintaining high standards of software quality and user satisfaction.

To streamline our development process and ensure the continuous integration of high-quality software, we have implemented a CI (Continuous Integration) pipeline. This pipeline automates the running of unit tests for some crucial features, allowing us to quickly detect and address issues early in the development cycle. By leveraging CI practices, we maintain a consistent development workflow and ensure that our updates and new features are reliable and stable.



```yaml
on:
  push:
    branches: [ "main" ]
  pull_request:
    branches: [ "main" ]

permissions:
  contents: read


jobs:
  build:

    runs-on: ubuntu-latest

    defaults:
      run:
        working-directory: DigitalTwin/backend

    steps:
    - uses: actions/checkout@v3
    - name: Set up Python 3.10
      uses: actions/setup-python@v3
      with:
        python-version: "3.10"
    - name: Install dependencies
      run: |
        python -m pip install --upgrade pip
        pip install flake8 pytest
        if [ -f requirements.txt ]; then pip install -r requirements.txt; fi
    - name: Lint with flake8
      run: |
        # stop the build if there are Python syntax errors or undefined names
        flake8 . --count --select=E9,F63,F7,F82 --show-source --statistics
        # exit-zero treats all errors as warnings. The GitHub editor is 127 chars wide
        flake8 . --count --exit-zero --max-complexity=10 --max-line-length=127 --statistics
    - name: Test with pytest
      run: |
        python -m pytest
```

Fig. 39 - CI Pipeline

While we have not developed a CD (Continuous Deployment) pipeline due to the complexities of using CARLA and SUMO, which need to be installed on the user's computer, we have explored containerization as a future solution. Docker images for both CARLA and SUMO have been found, offering a promising approach for simplifying deployments. However, due to time constraints, we have not yet implemented a fully containerized version of our pipeline.

8 Conclusion

The development of the ”Digital Twin - Aveiro Tech City Living Lab” project represented a significant advancement in the application of innovative technologies within the context of smart cities. Throughout this journey, we not only succeeded in designing a platform that integrates urban scenario simulation with autonomous vehicles and multimodal transportation but also demonstrated the practical viability and utility of digital twin models in complex urban environments.

The interaction between real and simulated components, enabled by the integrated use of urban sensor data and detailed simulations, proved to be fundamental for the effective analysis and management of urban mobility. This approach allowed for accurate visualization and a deep understanding of traffic dynamics, which are essential for the planning and implementation of more efficient and sustainable traffic policies.

The challenges faced, particularly in integrating real-time data and adapting existing infrastructure to support large-scale simulation, were overcome with technical innovations and adjustments in the project's architecture. These solutions not only reinforced the pioneering character of the project but also set a precedent for future smart urban development initiatives.

Furthermore, the continuous collaboration with institutional partners and the contributions from various team members were indispensable for the success of the project. The ability to quickly adapt to new demands and the integration of constructive feedback were crucial for the ongoing refinement of the platform.

In summary, the ”Digital Twin - Aveiro Tech City Living Lab” project achieved its initial objectives, establishing a solid foundation for future expansions and research in the area of digital simulation and urban mobility management. The lessons learned and the technologies developed will significantly contribute to transforming Aveiro into a smarter and more connected city, highlighting the essential role of technology in the sustainable development of urban areas.

9 Future Work

For future work to continue this project, there are several key points that could be further developed. One aspect is **enhancing scenario capacity with real-time data**, which would involve expanding the capability of scenarios by incorporating real-time data streams. Another important area is **integrating Autoware**, which would require updating the mapping infrastructure to ensure compatibility not only with SUMO and CARLA but also with Autoware. This integration would necessitate implementing a new map construction pipeline to facilitate Autoware integration seamlessly.

Looking further ahead, there are **future integration possibilities** that could significantly advance the project. This could involve not only incorporating Autoware but also leveraging this environment to understand and utilize communication between autonomous vehicles, thereby enhancing their performance and coordination.

Moreover, a critical aspect of future work would be **utilizing simulation results for real-world changes**. This involves employing simulation results to drive real-world changes and inform decision-making processes. For example, one could conduct experiments to assess the impact of shifting from car usage to bicycles on infrastructure, such as traffic lights and bike lanes. Analyzing the implications of reduced car traffic and increased bicycle usage could provide valuable insights for urban planning and transportation policy.

In summary, the future work for this project entails advancing key areas such as scenario capacity, Autoware integration, exploring futuristic integration possibilities, and utilizing simulation results for practical real-world applications. These efforts aim to push the boundaries of autonomous vehicle technology and contribute to sustainable transportation solutions.

10 References

References

[1] Krešimir Kušić, Rene Schumann, Edouard Ivanjko, ”Building a Motorway Digital Twin in SUMO”, September 2022. Last accessed on June 3, 2024. https://ieeexplore.ieee.org/document/9899796

[2] Kui Wang, Zongdian Li, Tao Yu, Kei Sakaguchi, ”Smart Mobility Digital Twin for Automated Driving”, August 2023. Last accessed on June 3, 2024. https://ieeexplore.ieee.org/document/10200728

[3] Talha Azfar, Jeffrey Weidner, Adeeba Raheem, Ruimin Ke, Ruey Long Cheu, ”Efficient Procedure of Building University Campus Models for Digital Twin Simulation”, October 2022. Last accessed on June 3, 2024. https://ieeexplore.ieee.org/document/9913679