|  |  |  |
| --- | --- | --- |
| A blue and black logo  Description automatically generated | **Departamentul Automatică şi Informatică Industrială**  **Facultatea Automatică şi Calculatoare**  **Universitatea POLITEHNICA din Bucureşti** | A blue and black logo  Description automatically generated |

LUCRARE DE LICENŢĂ

Titlu lucrare

Coordonator Absolvent

Prof. Anca Daniela Ionita Andrei SARANDAN

2024

Contents

[Contents 2](#_Toc167346049)

[Part I - General Aspects 5](#_Toc167346050)

[1 Introduction 6](#_Toc167346051)

[2 Inclusion of the study's domain of interest 7](#_Toc167346052)

[2.1 Urban Vehicle Access Regulations 7](#_Toc167346053)

[2.2 Low Emission Zones 7](#_Toc167346054)

[2.3 LEZs throughout Europe 8](#_Toc167346055)

[2.4 Current situation in Europe eu as prezenta diferite cazuri aici 9](#_Toc167346056)

[2.5 LEZs in Northern Europe 9](#_Toc167346057)

[2.6 Forecasts 10](#_Toc167346058)

[2.7 Current context regarding LEZs 11](#_Toc167346059)

[3 Thesis Statement 11](#_Toc167346060)

[3.1 Addressing the Complexity of Navigation throughout LEZs 11](#_Toc167346061)

[3.2 Similar applications 12](#_Toc167346062)

[3.2.1 Green-Zones.eu 12](#_Toc167346063)

[3.2.2 Urban Regulations 12](#_Toc167346064)

[3.3 Thesis Impact: Contributions and Implications 13](#_Toc167346065)

[4 Technical Documentation 14](#_Toc167346066)

[4.1 Software Technologies 14](#_Toc167346067)

[4.1.1 Hypertext Markup Language 14](#_Toc167346068)

[4.1.2 Python 15](#_Toc167346069)

[4.1.3 Flask 15](#_Toc167346070)

[4.1.4 SQLite 16](#_Toc167346071)

[4.1.5 JavaScript 17](#_Toc167346072)

[4.1.6 APIs 18](#_Toc167346073)

[4.1.7 UML 19](#_Toc167346074)

[4.2 Backend Development 19](#_Toc167346075)

[4.2.1 Foundations 19](#_Toc167346076)

[4.2.2 Route handling in Flask 20](#_Toc167346077)

[4.2.3 Database 21](#_Toc167346078)

[4.2.4 Access validation algorithms 23](#_Toc167346079)

[4.2.5 Dynamically rendered web pages 24](#_Toc167346080)

[4.3 Frontend Development 25](#_Toc167346081)

[4.3.1 General aspects 25](#_Toc167346082)

[4.3.2 Maps API 26](#_Toc167346083)

[4.3.3 Asynchronous function for handling form submission 26](#_Toc167346084)

[4.3.4 Notification display 26](#_Toc167346085)

[4.3.5 Dynamic elements 27](#_Toc167346086)

[4.3.6 Conditional selection 28](#_Toc167346087)

[4.4 Software Development Moldes and Tools 29](#_Toc167346088)

[4.4.1 Client-Server Architecture 29](#_Toc167346089)

[4.4.2 Integrated Development Environment (IDE) 29](#_Toc167346090)

[4.4.3 Virtual Environment 30](#_Toc167346091)

[4.4.4 Database Management Tools 30](#_Toc167346092)

[4.5 Development Techniques 31](#_Toc167346093)

[4.5.1 REST APIs 31](#_Toc167346094)

[4.5.2 JSON 32](#_Toc167346095)

[4.5.3 CRUD in Web Applications 32](#_Toc167346096)

[4.5.4 AJAX 33](#_Toc167346097)

[4.5.5 Document Object Model 33](#_Toc167346098)

[4.5.6 ORM 34](#_Toc167346099)

[4.5.7 Authentication and Authorization 34](#_Toc167346100)

[4.5.8 Templating Engine 35](#_Toc167346101)

[PART II – PROJECT CONTRIBUTIONS 36](#_Toc167346102)

[*5* Development Rationale and Methodology 37](#_Toc167346103)

[*5.1* Addressing the Issue 37](#_Toc167346104)

[5.2 Proposed Solution 37](#_Toc167346105)

[6 Development Methodology 37](#_Toc167346106)

[6.1 Analysing Requirements and Specification 37](#_Toc167346107)

[6.1.1 Information Sources 38](#_Toc167346108)

[6.2 Design elements 39](#_Toc167346109)

[6.2.1 Application Use Cases 39](#_Toc167346110)

[6.2.2 Activity diagram 41](#_Toc167346111)

[6.3 Non-functional Requirements 42](#_Toc167346112)

[6.4 Application Constraints 42](#_Toc167346113)

[7 Description of the Practical Application 42](#_Toc167346114)

[7.1 User Guide 42](#_Toc167346115)

[7.2 Application Verification and Validation 42](#_Toc167346116)

[7.3 Results Obtained 42](#_Toc167346117)

[8 Conclusions 42](#_Toc167346118)

[Bibliography 43](#_Toc167346119)

[Appendix 1. Code Snippets 45](#_Toc167346120)

# Part I - General Aspects

# 1 Introduction

The World Wide Web represents without a question an indispensable tool in the daily life during present times. On the other hand, the escalating threat of global warming is more prominent than ever before, with recent events occurring across the globe. Ideally, the internet’s capabilities must be focused on creating a better world for future generations.

The scope of this project is to present a web-based application that helps users comply to traffic regulations mandated by The European Union, primarily focusing on Low Emission Zones (LEZs). This web application seeks to raise awareness and trust in this scope, enabling users to make informed planning and consider the environmental impact of their travel and vehicle choice. It represents more than just a text repository or a presentation website as it is a comprehensive platform, which includes modern technologies like Application Programming Interfaces (APIs), extensive database model and complex access validation algorithms. All of these aim to deliver a great user experience, from account creation processes, thus raising the possibility of personal vehicle registration, to trip planning features. To improve the user experience, all the complex logic methods, data sets and algorithms are hidden behind the friendly and interactive interface.

The creation of this web application was done in three main steps. Firstly, the application development was based on a tight connection between backend and frontend parts, involving different programming languages and frameworks. Secondly, the extensive relational database model was designed to store relevant information about both the Low Emission Zones and application users. Lastly, in creating the database, a prolongated research was needed to cover the different complexities of the hundreds of LEZs throughout Europe. A key factor was to also fit the everchanging character of the LEZs. The information presented on the web application must be correct, up-to-date, and presented in an understandable manner. By doing so, the web application aims to contribute to broader efforts to create a more sustainable and resilient future for all.

The scope of this thesis revolves around developing an application geared towards improving user experience in navigating Low Emission Zones (LEZs), with a focus on providing information and validating access within LEZ cities. Since this concept is very new and still in development, the main objective is to raise awareness and promote their acceptance. LEZs may pose complexities for many individuals, hence efforts are needed to simplify comprehension and facilitate access.

The aim is to create an intuitive web application that provides information and help to access these Low Emission Zones throughout Europe. Since this is an ever-changing topic, the database system is incorporated to cover the regulations that are effective in July 2024. It is designed so that it can be adapted to ongoing updates, in order to provide valid information to the end-user.

# Inclusion of the study's domain of interest

## 2.1 Urban Vehicle Access Regulations

The increasing concerns surrounding pollution require the creation various strategies aimed to improve air quality. Pollution stems from a multitude of sources and presents in various forms. It possesses significant threats to both human health and the environment. Among the causes of pollution, vehicular emissions stand out as one of the most prominent. This is mostly concerning in larger urban settlements and metropolises that are suffocated by the raising number of cars. While public transport is a viable option in most cases, studies show a continuing increase in the number of vehicles, with an average of nearly one car per 1-2 individuals[SURSA]. Passenger cars stand out as a major polluter, accounting for 61% of total CO2 emissions from EU road transport [1].

The European Union is making efforts to drastically reduce air pollution through the adoption of new legislation aimed at pushing modern vehicles towards achieving zero CO2 emissions. However, the actual progress has proven to be much slower than hoped. The primary methods for reducing CO2 emissions from vehicles are manufacturing more efficient cars and changing the fuel type by transitioning to more eco-friendly ones, ideally derived from green or regenerative energy [1]. Even though electric and hybrid cars are taking a considerable part of the newly registered vehicles, the issue of older, higher-emission vehicles must not be overlooked.

Designing a strategy to reduce the pollution proves to be a challenging yet necessary measure. Such strategies, known as Urban Vehicle Access Regulations, have been adopted by several countries, predominantly in the West European. These regulations include schemas such as Low Emission Zones, Zero Emission Zones, tolls for driving in congested areas, creating more space for pedestrians, changing parking rules, and limiting traffic in certain areas. The goal is to comply with the newly adopted air quality standards, while also improving traffic. Additionally, some of these regulations usually impose taxes or fees that must be paid in order to gain access into the geographical area. Out of the many existing forms of Urban vehicle Access Regulations, most of them are known as Low Emission Zones (LEZs), which account for 73% of the current regulations [2].

Generally, vehicle categorization and access are based on the vehicle type, fuel type and emission class (Euro emission standard).

## 2.2 Low Emission Zones

Low Emission Zones (LEZs) are geographical areas, usually located in large urban settlements, where specific restrictions are imposed on more polluting vehicles. Typically, this kind of vehicles are prohibited from entering the zone, although in certain instances, a fee can be paid for access. Low Emission zones have proved to be a great method of reducing air pollution, especially targeting fine particles such as NO2, which are highly correlated to several respiratory diseases.

The concept of environmental zones originated in Sweden in 1996 and were initially created to reduce both air pollution and large vehicles noise. These zones were eventually replaced by national regulations that comply with EU environmental standards and now only cover heavy-duty vehicles [3]. Such examples were later adopted by other countries and are now widely spread throughout Western Europe.

In Europe, the three main air pollutants of concern are represented by nitrogen dioxide (NO2), particulate matter (PM, also referred to as PM10 or PM2.5 depending on the particle diameter), and ozone. The principle behind the functionality of Low Emission Zones is to restrict as much as possible the access of highly polluting vehicles. This can be achieved either by raising the costs of access through fees or tolls or by completely forbidding the access. Another approach would be retrofitting older vehicles to meet modern regulations, by adding diesel particle filters (DPF). These filters could elevate the vehicle to a slightly higher emission standard [5].

## 2.3 LEZs throughout Europe

Considering the transformative nature of a Low Emission Zone, defined by its main purpose of reducing pollution through restrictions on vehicle access in designated areas, such a solution cannot be swiftly implemented.

The European Union proposes a strategy to gradually implement such areas in the following years [4]. This approach facilitates the acceptance and adoption of a Low Emission Zone and allows each nation to fine-tune the schemas to better fit the needs and possibilities of the country and its citizens. Local authorities can effectively manage the requirements and priorities, enhancing the efforts to minimize pollution as much as possible. The Low Emission Zone term can be found under a different name, depending on the national law it has been regulated by. In Germany, Low Emission Zones are known as "Umweltzonen", while in France they are referred as ZCR ("Zone à Circulation Restreinte"). Both terms define strict criteria in order to separate vehicles based on Euro standard. While there is no clear recipe for implementing a Low Emission Zone in a big urban settlement, giving free-will for the nations to create and personalize theirs, has led to the existence of many such forms. Having more variations of the same regulation results in more case-studies and faster evolution in this field. In a relatively short span, regulations have taken on many different forms, some being more effective than other and some being easier to implement than other. Even though each approach presents its own set of merits and drawbacks, the end objective remains the same.

For example, one of the main differences between the LEZ regulations in Germany and France lies in the criteria selected for classification. To avoid confusion, each vehicle is assigned a sticker, which is required to be displayed on the windshield for enforcement by authorities. In Germany, vehicles can be assigned one of the three available emission stickers based on their Euro emission standards. Green represents the cleanest and most environmentally friendly out of them all, followed by yellow and, lastly, red for the highest emissions. These stickers were first introduced in 2008 and because of the regulations in place at that time, a diesel vehicle now needs to be at least EURO 4 to be eligible for the green sticker [6]. On the other hand, France introduced the Crit'Air vignette system in 2017. Crit'Air stickers are also color-coded and classify vehicles based on emissions. However, France has designated six categories ranging from green, which is only assigned to 100% electric and hydrogen vehicles, to dark gray for the most polluting vehicles.

While both systems serve a similar purpose of regulating vehicle emissions within urban areas, the specific criteria for classification and the appearance of the stickers are quite different. The Crit'Air vignette also includes a long-term strategy to introduce this schema throughout the cities with more than 150.000 inhabitants [7]. Additionally, the plan is to gradually tighten the standards as the years advance, by switching the minimum required sticker for access to the next one. As an example, in Paris, regulations in place since 1 June 2021 impose minimum Crit'Air sticker 3 (diesel EURO 4 and EURO 2 petrol cars). Starting with the year of 2025, Crit'Air sticker 2 (minimum diesel EURO 5 and EURO 4 petrol cars) will be required. Initially, this restriction was scheduled to be implemented on the 1st of July 2023, but it had to be postponed [8].

## 2.4 Current situation in Europe eu as prezenta diferite cazuri aici

The European Union legislation encourages member states to establish their own regulations for Urban Vehicle Access Regulations, such as Low Emission Zones (LEZs). National authorities must take into consideration different factors such as population density, existing infrastructure, public transport etc. Furthermore, within each country, the local responsible set specific regulations within their jurisdiction and the national framework ????????.

This decentralized structure offers several advantages. Firstly, it facilitates the implementation of diverse strategies, providing a range of test cases to evaluate the efficacy of different models in addressing air pollution and promoting sustainable urban mobility. Additionally, it acknowledges that national governments are best positioned to understand their own environmental challenges and devise targeted solutions accordingly.

However, this system can also result in complexity and inconsistency for the population. With regulations varying, not only between countries, but also among different municipalities within the same country, drivers may encounter a patchwork of rules and requirements as they navigate through different regions. This can lead to confusion and logistical challenges, particularly for individuals who frequently travel between areas with differing regulations.

The Crit'Air sticker system in France and the Umweltplakette system in Germany exemplify this diversity, with nine stickers utilized across two countries. Each of them available exclusively within its respective jurisdiction, governed by distinct criteria. This variety can pose challenges for most people. The proliferation of unique standards and administrative processes makes things more complicated and challenging, especially when it comes to coordinating environmental policies across different areas and governments.

## 2.5 LEZs in Northern Europe

The varying standards and regulations across these Northern European countries add another layer of complexity for individuals seeking to access Low Emission Zones. This divergence underscores the need for a comprehensive understanding of the specific requirements and sticker systems in place in each country, further complicating the experience for users and necessitating careful planning and compliance measures when traveling across borders.

Considering the aforementioned context, accessing a low emission zone becomes a significant challenge for the average individual. The multitude of European countries that have adopted a form of Low Emission Zone and the diverse range of regulations taken into consideration create a complex navigation landscape.

In addition to the stickers present in countries like France and Germany, the European Low Emission Zone schema is further compounded by a different strategy in Northern Europe. Nations in this region, such as Denmark (Miljøzone), Sweden(Miljözon) and the Netherlands(Milieuzone) implement their own systems for emission regulations. The core principle remains the same, to reduce pollution by restricting the highly polluting vehicles access in city centres. Also, the entry requirements for low emission zones in each country differ.

Within these frameworks, nationally registered vehicles are automatically included in a nation-central database, drastically facilitating the process of complying and also the efficiency of the LEZ. However, foreign vehicles, need to register online, typically prior to entering the LEZ to avoid penalties. This registration only enables access to the zone for vehicle that actually comply with emissions standards set by local authorities.

Northern Europe Low Emission Zones are primarily enforced through camera systems, designed to capture the vehicle’s registration number before entering the designated area. This system cross-references the captured registration number with the central database to verify the vehicle’s eligibility for entry. This automated approach enhances efficiency and accuracy while reducing reliance on and costs for manual inspections.

Overall, this digital registration and enforcement model streamlines the process for registered vehicles while promoting compliance with LEZ regulations, ensuring effective management of air quality within urban environments.

## 2.6 Forecasts

According to the report made by statista.com [9], forecasts expect a significant expansion of Low Emission Zones (LEZs) throughout Europe. Projections indicate a substantial increase in the number of LEZs, with an estimated total of 510 zones across 17 countries. This increase highlights the European Union's commitment to mitigating air pollution and promoting sustainable urban development throughout the continent.

|  |  |  |
| --- | --- | --- |
| Country | Number of LEZ in 2022 | Project number of LEZs by 2025 |
| Italy | 172 | 172 |
| Germany | 78 | 78 |
| United Kingdom | 17 | 18 |
| Netherlands | 14 | 14 |
| France | 8 | 42 |
| Sweden | 8 | 8 |
| Austria | 6 | 6 |
| Denmark | 4 | 4 |
| Spain | 3 | 149 |
| Belgium | 3 | 4 |
| Norway | 3 | 3 |
| Czechia | 1 | 1 |
| Finland | 1 | 1 |
| Greece | 1 | 1 |
| Portugal | 1 | 1 |
| Poland | 0 | 2 |
| Bulgaria | 0 | 3 |

Tabel 1. Number of Low-Emissions Zones in Europe 2022-2025[9]

## 2.7 Current context regarding LEZs

The multitude of terms used to designate Low Emission Zones (LEZs) across Europe drives up the complexity for regular individuals when navigating these environmental zones. While the purpose for each designation remains to reduce emissions and improve air quality, the linguistic and cultural diversity adds layers of intricacy.

From "Umweltzonen" in Germany to " Miljøzone " in the Denmark, and " Distintivo Ambienta" in Spain, the wide terminology presents a challenge for users seeking clarity and understanding. This complexity extends with regulations implemented in France, Belgium, England, Denmark, Sweden, Norway, Hungary, and Italy, each with its unique terminology and approach.

The effectiveness of Low Emission Zones is directly linked to both the rigor of the implementation but also the level of compliance. While the establishment of a LEZ is a great proactive first step towards improving air quality in urban settlements, in reality, the impact relies on the degree individuals comply to the regulations enforced in such areas.

It is essential to keep in mind that even the best designed LEZ strategy can fail to reach it’s intended goals if control measures are lacking or if vehicle owners do not comply. As The Deputy Mayor for Transport has confirmed in [10], automated monitoring systems, such as the use of plate recognition cameras, prove to be an effective enforcement strategy.

Over more, in order to build popularity and trust in any form of Urban Vehicle Access Regulations, public awareness campaigns serve a crucial role in ensuring that LEZ regulations are upheld. All Nations need to pay effort into conveying public that the only objective is not to create inconveniences or to impose financial burden, but rather to mitigate pollution in highly populated urban centres. Such initiatives are part of a long-term plan designed to achieve lasting benefits for public health and environment.

For individuals to fully comply and embrace the notion of Urban Vehicle Access Regulations imposed through Low Emission Zones, it is imperative to be provided a clear overview of the existing LEZ structures and their benefits. The large number of LEZs across Europe, navigating across the continent, throughout major cities, can become challenging and may create confusion and inconvenience for drivers.

# 3 Thesis Statement

## 3.1 Addressing the Complexity of Navigation throughout LEZs

Taking into consideration the facts outlined above, I have identified a critical gap in the landscape of Urban Vehicle Access Regulations: a lack comprehensive platforms, tailored for the needs of regular individuals, willing to navigate throughout Low Emission Zones (LEZs) with ease and while also respecting the regulations in place. At this time, available resources often fall short in providing users clear details about LEZ restrictions and compliance. Considering the inherent complexities and long-term evolution within the LEZ landscape, we can identify this gap as a significant opportunity ripe for exploration.

While online resources and information are indeed available, usually the information is dispersed across many different platforms, with segmentation made by countries or even by cities. Fragmented landscapes can be overwhelming for regular users, who would often find themselves spending their significant time navigating through multiple websites when gathering relevant information for their specific situation and needs.

## 3.2 Similar applications

### 3.2.1 Green-Zones.eu

The Green Zones app provides comprehensive information regarding the Low Emission Zones (LEZs) and accessing criteria across Europe. The application is split on both mobile and the web platform, providing valuable resources for individuals seeking to understand and comply to the regulations in LEZs.

The mobile app offers the users the possibility to conveniently add relevant details about their vehicle. Then, in order to check compliance with a LEZ, the user needs to select one of the points highlighted on the map. This feature is offered free of charge and accessible to all users. Additionally, the app provides air quality information on a number of cities, which is an important factor taken into consideration when evaluating the efficiency of a LEZ.

However, to access additional features and services, like saving different vehicles to an account, users are required create a subscription-based account, which require a monthly fee. While the premium services offer added functionality, this is rather useful for enterprise users.

For regular users who only plan a few trips each year, the associated subscription fee could cause significant drawbacks. While the app offers valuable features and services, like the vehicle registration, the app only promises the latest data on for the three available subscription plans. In many cases, the subscription fee may outweigh the benefits for individuals that rarely travel through Low Emission Zones. This subscription-based model may discourage casual users who seek to information regarding LEZ regulations compliance.

The web platform is a great complement to the mobile app and is designed to provide comprehensive information about Urban Vehicle Access Regulations, such as LEZs, actual environmental regulations. The website can also be used to purchase registrations for several countries which have LEZ schemas active.

Overall, the Green Zones app offers a user-friendly interface and valuable features for navigating LEZs and promoting environmental awareness. However, the subscription-based model may present a barrier to entry for some users, particularly those who do not require frequent access to premium features.

### 3.2.2 Urban Regulations

UrbanAccessRegulations.eu is designed as a comprehensive online platform for information on different urban access regulations throughout Europe. A significant advantage is the fact that the website offers a wealth of information, organized in an intuitive manner, accessible to anyone with internet access. The website covers the complete range of regulations, from Low Emission Zones (LEZs) to road tolls and emergency schemas, creating a place to find relevant information for any situation.

One very notable feature of UrbanAccessRegulations.eu is the extensive and well-structured database. Users can navigate easily across different categories to find the information relevant to their specific situation or curiosity. Over more, the website offers very detailed information for different categories of vehicles, which assures accessibility for a wide range of users.

However, while offering a comprehensive coverage, the UrbanAccessRegulations.eu may overwhelm some users with the extensive volume of information at hand. While the structure is well-organized, users might need to spend long periods of time diving deep into the website to find the specific details for their needs. Further, the addition of personalization features, such as the saving vehicle details and registrations, would expend the possibilities of the website's appeal to users. The large variety of vehicles and regulations could be eased by a tailored application for individual circumstances.

Another positive note, UrbanAccessRegulations.eu offers the information and services free of charge, available online to all users regardless of their financial resources. This free of cost model creates a tremendous advantage, as it eases access to crucial information about Low Emission Zones. Providing so much information at no cost, the website drops financial burdens and facilitates the learning about LEZs. This serves as a fundamental step towards fostering compliance. As individuals become more informed about LEZs, they are better prepared to adhere to regulations, therefore improving the effectiveness of the already existing LEZs and potentially creating a social context for the establishment of additional zones in the future.

In summary, UrbanAccessRegulations.eu provides a large number of valuable resources for users seeking information on urban access regulations throughout Europe. While the offer compounds comprehensive coverage and is free to use, the website's diverse volume of information and less user-friendly navigation may pose challenges for some unexperienced web users.

## 3.3 Thesis Impact: Contributions and Implications

The development of a new web-based application arises from the identification of a specific niche within the market, tailored for to the needs of regular individuals. This app has been purposefully designed to be intuitive and accessible to everyday users, creating a seamless experience that is both user-friendly and based on a cost-free model. The goal of creating the app is to eliminate all barriers of entry, such as subscription fees or even prior expertise in the domain, in order to democratize access to information about Low Emission Zones. This will empower a wider audience to make informed decisions before buying a new vehicle, while also expanding the impact and reach of LEZs.

One of the key aspects of this app is the emphasis on tailored solutions, allowing users to create accounts and customize their experience according to their needs. Through account creation, users will save their vehicles to the platform and make use of the route planner feature in order to facillitate navigation across European LEZs. This personalized approach enhances user engagement and ensures that individuals can access relevant information tailored to their specific needs.

In line with the user-centric design, the app adopts a guided interface, streamlining the user experience while also reducing the time required to obtain relevant information for the situation at hand. The intuitive design allows users to quickly obtain the information they require without navigating through extensive pages of content. Over more, the app features informative text-based pages, for comprehensive insights into LEZs.

At the core, the application makes use of a robust database and extensive algorithms for assigning the correct registration to a vehicle and LEZ access validation. These technical features ensure the accuracy, reliability, timeliness, and relevance of the provided information, in order to ensure confidence in users and promote compliance to LEZs regulations. While the app is designed to be oriented towards individual consumers, the database model and it’s free-to-use nature are also well-suited for enterprises, accommodating a larger number of scenarios, vehicles and routes within the same or different accounts.

Despite the focus set on simplicity and accessibility, the app keeps the same high standards for information quality, consistent with the other applications already established on the market. The purpose is to deliver up-to-date, accurate information, while the application also contributes to raising awareness about Low Emission Zones and spreading the culture of environmental responsibility through its users. Essentially, the implementation of this application takes a significant step by advancing the effectiveness and accessibility of Low Emission Zones. Through its unique manner, the application ultimately contributes to the larger scope of improving air quality in European cities by promoting sustainable mobility.

# 4 Technical Documentation

## 4.1 Software Technologies

### 4.1.1 Hypertext Markup Language

Hypertext Markup Language (HTML) is designed as the core foundation of web application development and is used to create the structure and content of the web pages. As a markup language, HTML makes use of a system based on tags and codes inserted inside the text to indicate the way of displaying the text or how software applications should process it. The tags are generally enclosed within angle brackets (</>) and contain attributes to specify additional information or characteristics about the content.

HTML is used in web documents to define the semantics and layout because it enables developers to compose structured documents, making use of elements such as headings, paragraphs, forms, lists, images, anchor tags, etc. Having a clear and concise syntax, HTML facilitates the process of creating well-structured and user-friendly web pages that can be interpreted by most web browsers and across all available devices and platforms. Being the backbone of the web applications, HTML creates a strong foundation for building visually appealing and interactive web applications, laying the groundwork for integrating dynamic features with other technologies like CSS and JavaScript.

HTML5 represents the latest evolution of the HTML standard, adding new modern features and capabilities designed to enhance web applications development. Among the newer elements introduced by HTML5 count <header>, <nav>, <article>, and <footer>. Using these tags excels in providing a better organized structure and a clearer meaning to web documents. Additionally, HTML5 offers support for multimedia elements, with the tags like <video> or <audio>, but also brings advanced form controls and APIs for geolocation [13] and many other functionalities. This latest HTML iteration, allows developers to leverage its capabilities in order to create interactive online applications, implementing a seamless user experience across different devices.

Markup languages are commonly used in various contexts, including web development, document processing, and data interchange. For example, HTML (Hypertext Markup Language) is a markup language used to create and structure web pages, defining elements such as headings, paragraphs, links, images, and forms. XML (Extensible Markup Language) is another widely used markup language that allows users to define their own tags and document structures, making it suitable for data storage, document interchange, and configuration files.

Markup languages provide a standardized way to represent and communicate information, enabling interoperability between different software systems and platforms. By separating content from presentation, markup languages facilitate the creation of structured and semantically meaningful documents that can be easily processed, interpreted, and displayed by software applications.

### 4.1.2 Python

Python is a very powerful programming language known for its simplicity in syntax and yet very versatile and reliable, making it an excellent choice for a wide selection of applications. Taking advantage of an extensive standard library and vast ecosystem of third-party packages, Python allows developers to tackle, in a very efficient manner, diverse tasks from data analysis to artificial intelligence and automation to web development.

One of Python's key aspects lies in its flexibility and ease of use, allowing developers to quickly test ideas and implement solutions. The clean and intuitive syntax helps goes for readability and maintainability, making it accessible to both junior and senior programmers alike. Over more, Python's dynamic typing and automatic memory management simplify development, reducing the time and resources required to build and manage complex systems.

When it comes to web development, Python shines as a robust and scalable backend solution. Its main web frameworks are Flask and Django, which provide developers with powerful tools for building secure web applications with lots of features. Over more, Python's extended support for database integration and RESTful APIs makes it ideal for developing backend systems with seamless frontend interaction through the interfaces and external services.

Overall, Python's versatility and large ecosystem make it a popular choice for backend development, offering users the flexibility and power to create elaborated yet reliable web applications. Its popularity in web development was highlighted by various surveys and rankings conducted by reputable organizations. For instance, the Stack Overflow Developer Survey [15] consistently ranks Python among the top programming languages, highlighting its widespread adoption and relevance in the industry.

### 4.1.3 Flask

Flask is a flexible and lightweight Python framework, designed to make web development efficient yet simple and. It was developed by Armin Ronacher, and became popular for its minimalist design philosophy, allowing developers to build full-scale web applications quickly while also reducing the amount of repetitive code. At its core, Flask provides developers the essential tools for handling HTTP methods, routing requests, but also to render templates, while keeping scalability through its extensive ecosystem and modular architecture [16].

Developing a complex Flask application involves creating rigorous file structure, which, although less stringent than that present in other frameworks, offers flexibility and ease of customization. This structured approach fosters scalability and adaptability, enabling potential to craft web applications aligned with modern standards, requirements and personal preferences. Despite the need for meticulous planning, Flask's minimalist design and structure gives developers full power, dropping excessive framework constraints, allowing them to focus their resources on innovation and problem-solving.

Another advantage of Flask is its performance and scalability. Making use of Python's asynchronous programming capabilities, Flask applications efficiently handle concurrent requests, while also assuring optimal performance even under heavy loads [11]. Over more, Flask's lightweight footprint and resource usage make it suitable for building APIs, microservices, or other lightweight web applications where resource efficiency is essential.

Flask applications are found in a wide range of industries and business sectors, from small businesses and startups to large enterprises. Flask's simplicity and versatility, receives appreciation from many web developers building prototypes, Minimum Viable Products (MVPs) but also production-ready web applications. Some of the most notable applications built with Flask include Netflix, Airbnb, or Uber [15] but it’s also widely used by microblogging platforms, content management systems (CMS), RESTful APIs, and data visualization dashboards. Flask's extensibility and modularity make it suitable for projects of any size and complexity, offering developers the flexibility to gradually scale and evolve their applications when needed. Overall, Flask's flexibility, simplicity, and performance make it a popular choice among software developers and enterprises seeking to build modern and efficient web applications.

### 4.1.4 SQLite

SQLite is a self-contained lightweight, and serverless system for relational database management that is widely acclaimed for its reliability and versatility. Particularly, it is favoured for its seamless integration into various applications, demanding minimal configuration and administrative effort. Despite having a lightweight nature, SQLite offers powerful features and capabilities, commonly associated with more extensive database systems. SQLite provides capabilities for transactions management [18], indexes, and triggers.

One notable aspect of SQLite is its support for class modelling, a fundamental concept in object-oriented programming. Developers can map classes to SQLite tables in their applications, creating a seamless interaction between the application's codebase and the database. This creates the opportunity for the implementation of object-relational mapping (ORM) techniques, simplifying database operations and promoting code maintainability [20].

Furthermore, SQLite's smooth compatibility with Python makes it an ideal choice for many developers. Python's built-in support for SQLite offers developers a strong foundation for database interaction. While SQLite integration is provided within Python's standard libraries, leveraging its functionality also requires a deep understanding of SQL queries. A good integration necessitates meticulous attention to detail when creating SQL statements or when managing database connections to ensure data security and integrity. Despite its apparent simplicity, utilizing SQLite in Python effective applications requires a thorough knowledge in both database concepts and programming. Python's extensive standard library includes modules for SQLite integration, enabling developers to perform database integration with ease. Over more, popular third-party libraries such as SQLAlchemy provide advanced ORM features, further enhancing the flexibility and functionality of Python applications using SQLite databases.

Overall, SQLite's combination of simplicity and reliability makes this combination with Python a compelling choice for developers seeking a lightweight yet powerful database solution for their applications. Its support for class modelling, seamless integration with Python, further solidify its position as one of the most preferred database technologies in the software development landscape, being actively used by many enterprises [20].

### 4.1.5 JavaScript

JavaScript is well renowned for its versatility and widespread adoption. This programming language plays a pivotal role in modern web development. JavaScript powers an extremely wide range of online experiences, from dynamic web pages to interactive web applications, making it an indispensable tool for web developers worldwide.

Over the years, a multitude of libraries and frameworks have emerged to further augment JavaScript's capabilities. Among the most popular, React.js [21], maintained by Meta, has gained widespread adoption for its component-based architecture and efficient virtual DOM (Document Object Model) rendering, making it a popular choice for front-end developers. On the other hand, Angular [22], maintained by Google, can be a comprehensive solution for developing smaller, single-page applications, offering features like two-way data binding and dependency injection to facilitate the creation of scalable and maintainable front-end applications.

Nevertheless, “the vanilla” form of the language remains the preferred choice for many developers. Its inherent features and flexibility enable developers to create customized solutions without the overhead of additional dependencies, promoting efficient development practices.

JavaScript's versatility shines through its seamless integration with various APIs, empowering developers to harness a wide range of functionalities within their applications. Whether in accessing third-party services via RESTful APIs or in creating an interface with browser APIs to manipulate the Document Object Model (DOM), JavaScript serves as a robust foundation for building comprehensive web applications enriched with various features.

Asynchronous programming functions [23] represent another standout feature in JavaScript, allowing non-blocking operations and enhancing application responsiveness. Through mechanisms such as “promises” and “async/await”, the programming language can manage asynchronous tasks effectively, ensuring smooth user experiences even during the more complex tasks or more resource-draining sessions.

JavaScript can be used in conjunction with backend frameworks like Flask, therefore becoming essential in enabling bidirectional communication between the frontend and backend components of a web application. Through techniques like AJAX (Asynchronous JavaScript and XML), JavaScript facilitates seamless data transfer, therefore it is considered an important tool for developers to build real-time, interactive applications.

The event-driven architecture of JavaScript empowers developers to properly handle multiple events simultaneously, allowing for the creation of interactive and responsive user interfaces. By creating event listeners and attaching them to DOM elements, developers can handle user interactions in real-time, enhancing user experience and engagement.

Working with the extensive ecosystem of libraries and plugins surrounding JavaScript further amplifies its capabilities, providing developers with a bundle of pre-built solutions for common development tasks. From UI frameworks like React to open-source, run-time environments like Node.Js, JavaScript's ecosystem offers a multitude of tools to streamline development workflows.

The lightweight syntax and dynamic typing contribute to JavaScript’s appeal among developers, facilitating swift prototyping for applications. Its flexible nature allows developers to iterate rapidly, exploring different approaches before refining their code before releasing the product that meet production-ready requirements. The programming language adheres to modern web standards while also ensuring cross-platform compatibility, which makes JavaScript an ideal choice for creating applications for a diverse target-audience. With broad support across browser platforms and different operating systems, JavaScript empowers developers to create powerful experiences that delight users across all devices.

In summary, JavaScript can be suitable in both frontend and backend web development, as it is a foundational technology in the modern web ecosystem. From its elegant syntax and versatility to its cross-platform compatibility and extensive ecosystem, JavaScript helps developers create robust, engaging, and scalable web applications that drive innovation and enhance user experiences.

### 4.1.6 APIs

Application Programming Interfaces (APIs) are sets of programming code or mechanisms which serve as intermediaries that allow different software applications to interact and communicate with each other [25] [26]. They define the protocols and methods through which different software components can interact request services, and exchange data between each other. APIs play a fundamental role in modern software engineering by enabling developers to leverage available functionality offering developers a gateway to connect a vast array of services, tools, and functionalities.

One of the key benefits of an API represents its ability to abstract away underlying and complex processes. This allows software engineers to focus on developing new features and functionalities without needing to have a deep understanding of the intricacies that stand behind those implemented features. The API provides a well-defined interface that shields developers from the unnecessary complexities, making it smoother to integrate third-party libraries, frameworks, and services into their applications.

Furthermore, APIs assure interoperability between different systems and platforms, enabling smooth integration of diverse software components. By conforming to standardized conventions and protocols, APIs ensure that software components communicate effectively regardless of the technologies or programming languages used at the core.

Additionally, APIs can be used to create modular and scalable software architectures by breaking down large and complex systems into smaller manageable components which are easier to manage. Their modular approach allows developers to create and maintain more efficient software, as the focus is centred on building individual components and which are interconnected using well-defined APIs.

A Web API, also known as a Web Service API, serves as a conduit between the web server and the browser, by facilitating the communication and data exchange. In general, all web services are considered APIs, but not all APIs are specifically designed for web services. Among the wide range of API types, the REST API stands out as a specialized form of Web API that adheres to a standard architectural style [25] [27].

The terminology surrounding APIs, such service APIs, stems from their historical development predating the widespread use of the World Wide Web [28]. However, in the context of modern web development, the term "API" often refers to REST APIs. The evolution reflects the shift towards web-based architectures and the dominance of RESTful principles in contemporary web application development.

In general, REST APIs facilitate the communication via HTTP requests, performing standard database operations such as creating, reading, updating, and deleting records within a resource. The well-known HTTP methods: GET, POST, PUT, and DELETE [29] are used for these operations. Responses are delivered and structured in various formats including JSON, HTML, XML, or plain text. Request headers represent a crucial part in REST API interactions, providing important identifier information such as metadata, authorizations, URIs, caching, and cookies.

Overall, APIs represent an essential tool in modern software development, enabling creation of interoperable, robust, and scalable applications by leveraging existing software functionalities and resources while abstracting away complexities.

### 4.1.7 UML

UML stands for Unified Model Language and represents a standardized set of diagrams which are meant to help both software developers and stakeholders with the construction and the visualization of the system. This modelling language provides blueprints for a wide range of infrastructures, which offer different views on the parts or scenarios of the application, from a technical or business perspective. UML includes structural diagrams for representation of static aspects, behavioural diagrams for the representation of system modelling elements and diagrams for the interactions between its elements.

The main advantage of using a standardized modelling language is that all stakeholders can get an understanding of the project, regardless of their technical background. The easy-to-read nature of UML diagrams makes them widely used in project documentations, as a newcomer can quickly get an understanding of the different aspects of the system.

The industry offers plenty of tools for developers to create UML diagrams, which are usually integrated during the phases of the analysis and design of the system. The features of UML are extended by other modelling languages, like SysML (Systems Modelling Language), which was specifically designed to include complex hardware and software elements used in system engineering.

## 4.2 Backend Development

### 4.2.1 Foundations

Backend development represents the backbone of web applications architecture, and it can be considered the engine that powers the entire system. It is responsible for a range of critical activities, like logic implementation, data processing, and handling communication with the user interface. A robust backend is essential for applications intended fulfil a large purpose, empowering them to manage data effectively and to interact with users. Without the complex backend, web applications are essentially limited to static website pages, without the dynamic functionalities available in most modern digital experiences.

In the context of this application, the backend architecture assumes paramount importance, as it is responsible for handling data management, LEZ access validation through algorithms, user account creation and inputs, HTML requests and responses but also dynamic web templating. One of the key aspects of the backend architecture in a web application is the establishment of routes, which essentially serve as pathways for directing incoming web requests to the appropriate resources within the application. Managing a meticulous route configuration, ensures smooth user interaction and navigation within the web application.

### 4.2.2 Route handling in Flask

Within the context of web development using the Flask framework, a route defines a mapping between a URL and a Python function. This mapping enables the execution of specific code sequences when a user accesses a particular endpoint. Route handling represents a fundamental aspect of web backend development, as it is responsible for dictating how URLs are mapped to specific functions within the application. In Flask, routes are generally managed within the views.py file, and takes advantage of decorators which are used to define routes and the corresponding functions.

In Flask, routes are generally defined using Python functions which decorated with a specific syntax, which includes the web application instance [30]. The decorator also takes, as the first parameter, the specific URL path it addresses. The python function decorated by this operator is responsible for handling the request when users navigate to the specified page. Additionally, the decorator accepts the “methods” parameter, which refers to the HTTP requests accepted by the configured route, such as *“GET”* and *“POST”*.

Considering the Code 1 example attached in first Appendix chapter, the code snippet defines the route function implemented within the Flask application, designed to handle the database registration of a new car. The route function is defined for the URL *“/new-car”* and accepts HTTP GET and POST requests. This configuration is made by the use of the “methods” parameter inside the *“@views.route”* decorator, where *views* is an instance of a Flask blueprint object.

The function proceeds to retrieve data submitted via the web form, inside the handling of a POST request. The data is retrieved through the Flask *request* object, specifically using the *form* attribute, which returns a dictionary type interface to access form data submitted inside the HTML POST request.

The algorithm extracts the data using dictionary indexing (request.form.get[‘item’]), assigns it to the corresponding variables and then creates a Car object which is integrated into the session and then immediately appended the object to the database.

At the end, the function makes use of the *render\_template()* function within a Flask application, in order to generate dynamic HTML content in a web browser. This function is integral to Flask's templating system and enables the incorporation of dynamic data into HTML templates.

Within the *render\_template()* function, the first argument "new-car.html" refers to the name of the HTML file to be rendered. This file, stored within the application's templates directory, serves as one of the pages which construct the final web application displayed to the user.

The subsequent argument, *user=current\_user*, provides dynamic data to be incorporated into the HTML template during rendering. In this context, *current\_user* represents the user currently authenticated and interacting with the web application. By passing this object to the template under the variable name *user*, the web page gains access to user-specific information, such as the name, saved cars, or other relevant information stored inside the application’s database. This facilitates personalized content generation tailored to the authenticated user's context.

In summary, the code snippet showcases the integration of Flask's routing mechanism and request handling capabilities to facilitate user registration within a web application, contributing to the overall functionality and user interaction of the system.

In essence, the code snippet creates an illustrative approach of how the web application uses Flask’s routing mechanism and request handling features to enable user interaction. It highlights one fundamental aspect of the backend architecture, showcasing its role in enhancing the overall functionality and user engagement with the system. However, it's essential to acknowledge the fact that the system encompasses a wider range of features and functionalities beyond car registration. These additional components contribute to scope of creating a seamless and comprehensive user experience.

### 4.2.3 Database

At the core of this web application is the relational database model displayed in figure 1. In flask applications often use libraries for Object-Relational Mapping (ORM) to interact with the database. This application uses SQLAlchemy and the database is created and managed in the models.py file. This library allows developers to define database tables using Python classes.

Structurally, the Zone table is responsible for storing the main information for the Low Emission Zones. For each city that has a Low Emission Zone schema, key information for the user, like minimum Euro Standard restrictions, required registrations and their type (physical/digital) are stored. This data is accessed by internal algorithms and usually queried by class functions that are also defined in the models.py file.

ZoneTemporaryData table is related to Zone table through a foreign key. Specifically, ‘zone.id’ column from ZoneTemporaryData references the primary key, ‘id’, from Zone table. This means that each instance from ZoneTemporaryData belongs to a single instance in Zone. The table, as its name suggests, is designed to hold data regarding the temporary characteristics of some Low Emission Zones. For example, this is especially relevant for countries like Italy or Bulgaria which have Winter Low Emission Zone schemas. These are only effective during a specific time of the year. Outside this period, different restrictions apply, therefore the algorithms need to have access to both scenarios in order to validate vehicle access. The backref parameter used in [ANEXACOD] creates a back reference in the Zone model. This allows accessing ZoneTemporaryData instances from a Zone instance using the temporary\_data attribute.

Another essential part of the database model is the User table and the ones related to it. The User table is created to store the essential information for account creation. The amount of information is minimal, as the nature of the application does not need more, but this is also favourable for safety and resources reasons. The main fields, email and password are need for user login. In order to keep the same standards as most modern applications, the database stores a hashed version of the password. The encryption is made using the SHA256 algorithm, which is one of the most secure and used hashing algorithms today [31]. This feature is provided in Python by the import *generate\_password\_hash()* and *check\_password\_hash()* functions from *werkzeug.security* library [32]. The login and register methods are designed as separate routes, as they have their specific web page.

Creating a personal account for each user is important, however this would be pointless without the possibility of storing personal data related to this account. In this application, the user has the possibility to save his vehicles and his routes.

As the name suggests, *Car* table stores the relevant data for the user’s vehicles. It is related to the *User* table through a foreign key, which creates a one-to-many relationship. Meaning, a user can have multiple car instances associated with his *id.* The *Car* table is maintained from the two specific webpages: Add Car or Edit Car. While in reality a vehicle has plenty of details, the application is designed to store only the data that is relevant to validating the access in the already existing LEZs. In this case, details like registration plate number are skipped, but producer brand and model are still considered relevant for easier user management and selection. Over more, the details considered relevant for access validation are build year, fuel type, euro standard and LEZ registrations, which all have separate fields. The completion of this data is the user’s responsibility and he should respect the information registered in the vehicle book.

Routes are saved in a separate table (SavedRoute), which is also related to the User table through the foreign key. Again, the one-to-many relationship implies that a user is able to save multiple routes. This comes as a huge help for frequent travellers and larger enterprises. As for now, the maximum number of destinations in a route is not limited, therefore, creating a database table for this model has one clear solution. The destinations are saved in JSON format and saved in *destinations\_json* field, which is used in the application logic, but they are also saved in *destinations\_text* field which is used to store the human readable version of the data.

Over more, each country that has a Low Emission Zone implements the abstract class *GeneralRegistrations.* This means that for each country a table is created for storing the available registrations. The OOP principle of inheritance allows the creation of one class, superclass, or the parent, which can be extended by other classes called children or subclasses. Through the extension, the subclasses take the parameters and the methods defined in the superclass but can also modify them. In this case, the GeneralRegistration is an abstract class and serves as a blueprint for the other country-based registrations, as it does not create a table in the database. The superclass defines the main relevant attributes and implements the class method find\_best\_registration\_badge, which is used in the Eligibility Check web page. The rest of the classes, extend the GeneralRegistation and some of them redefine the method or may even add some additional attributes. For example, PolandRegistrations class creates some columns necessary for setting the minimum criteria based on Euro Standard but also on the last registration date of the vehicle. Based on the values stored in these fields, the find\_best\_registration\_badge also takes a different form.

A computer screen shot of a computer

Description automatically generated

Figure 1 Database model

### 4.2.4 Access validation algorithms

Considering the scope of this application is to create a user-friendly platform that covers the wide variety of European Low Emission Zones, I have composed the criteria validation algorithms. The plan is to have a different algorithm for each country that has a LEZ schema. This strategy is necessary for covering all possible scenarios while assuring the validation of the relevant criteria.

The structure is designed to ensure an easy upgrading (more countries, more vehicles etc). the scalability of the application. This is ideal because of the evolving nature of the Low Emission Zones. In order for the application to stay relevant in the future, slight adjustments might be needed in the access validation algorithms. Separating the algorithms for each country means the backend can be viewed as a set of interchangeable modules. Each module represents a country that has a Low Emission Zone schema and is structured in three parts: the validation algorithm, registrations and the country objects.

Since each Nation evolves at its own pace, this modular approach means an easier maintenance for both the developer and the stakeholders and therefore, deploying the updates for the new validation methods becomes much simpler. While the responsible stakeholders can monitor the evolution of each country separately, the developer can create small updates only in the specific module. Separating the modules and making the modifications only in one of them ensures minimum down-time of the application and quick bug-fixes.

The validation algorithms are used in the Route Planner page in order to display a short summary of the relevant information. The displayed information includes a concise statement regarding the access in the selected LEZs and other key details, facilitating user compliance to the regulations. This way, the user does not need personally research all the regulatory requirements when planning a trip. The user only needs to fill in all points of interest and select one of the previously saved vehicles.

By the click of the button, the POST request is sent to the application backend and the relevant algorithms are triggered. For each country listed in the request, the corresponding algorithm is triggered. Since most countries have different restrictions from one zone to another, the algorithms are designed to check for the city received in the request. Running a separate algorithm for each country, proves effective in handling all the possible scenarios (LEZ access granted or forbidden, LEZ not active/inactive during selected travel period, city or country does not have LEZ, etc.). This is highly challenging when checking the access for several points of interest at the same time, as this involves verifying different attributes for the selected vehicle.

While viewing the Route Planner page from the front-end it seems simple, the database query, the POST request and the validation algorithms are all processed behind the scenes, in the application backend. As soon as the POST request is received, the “navigation” function handles the data inside the request. Since one request can include several points of interest, they are stored in JSON format and loaded in a list variable. By parsing this list, we can call the correct access validation algorithm for each item.

Based on the values returned by the access validation algorithms, we can create the type and the information of the notifications that will be displayed to the user. The type of the notifications is success or error and creates a suggestive design of the notifications and will be described in the Front-End chapter. The text inside of the notifications is queried from the Zone database through a series of class methods. These methods are created inside models.py file and take the city name parameter. Each of them, is responsible for retrieving key information for the user, like short description of the LEZ, penalties for not complying, official authority’s webpage, etc.

All this data is stored again in the JSON format and is sent as a response to the front-end of the application. There unpacking of the information takes place and it’s displayed to the user. This instant communication between the front-end and the back end create a user-friendly page, using the abstraction principle, meaning the complex logic of the application is hidden from the user, who is only interacting with the system through the graphic interface.

### 4.2.5 Dynamically rendered web pages

Creating a web application with multiple pages can become an exhaustive work. Fortunately, modern technologies allow developers to create dynamically rendered webpages. This means that in the backend, the data is prepared for display by set of predefined functions and methods. Configuring the routes responsible for displaying the pages in this manner, reduces the amount of repetitive code and shifts the focus of the developers on the backend architecture rather than on interface design.

A smart approach for the application is to take advantage of the database model in order to create the web elements instead of manually creating them inside the HTML files. For example, the Zone table stores important data about the Low Emission Zones which is used in the algorithms described above. Furthermore, this data is dynamically rendered as part of the information pages for each country. Another dynamic element inside these info pages is the div that contains the LEZs in the respective country. At core, it is an <ul> element (unordered list) and contains the elements queried by the backend functions.

The info pages are handled by a dynamic method. The function route decorator takes the <country> parameter, based on which decides the query statement and the rendered HTML page. The parameter does not need to be entered manually, as it automatically takes a value when the user accesses one of the county pages from any of the webpages.

## 4.3 Frontend Development

### 4.3.1 General aspects

At core, the web application consists of a multitude of HTML files, structurally tied between them. This flask application stores these HTML files inside the *“templates”* folder. While the HTML allows the creation of simple web pages, without the added features of CSS (Cascading Style Sheets) and other programming languages like JavaScript, these pages are primitive and static. From an architecture perspective, HTML templates can be viewed as the skeleton of the application.

In this application, CSS and JavaScript files are stored inside the *“static”* folder and are linked to the HTML pages inside the *<head>* tag. This file structure is intended to create order and well organization for the developers. Following the protocol or the rules of structuring files is just as important as coding standards when creating a good application. This is especially important in web applications, which used a combination of many different programming languages and file types.

An added benefit of Python is the multitude of templating engines available. This application uses Jinja templating across all web pages, which reduces code redundancy and improves web page and programmer efficiency. In general, the web pages inside one web application have similar look and feel and reuse some elements. This can be easily achieved through the use of Jinja templates. The web pages of this application are created as an extension of *base.html* file. In Jinja, the sentence for this is simple yet reliable. It is enough to create the base template for the file, which is made by *{% block %}* elements. Then, in the rest of the files, the extension is done using the key block *{% extends file.html%}.* This method allows effective reuse of elements across different pages.

The navigation bar represents a key element in any web application because it assures smooth navigation across the website. Since this element is present on all webpages, the navbar was created in the *base.html* file. We need to remember that this base.html is not accessible to the user and it is just a template. In fact, all these other pages are an extension of this *base.html* file. Over more, in Jinja templating, the CSS and Script files linked to the base template are also made available in the files that extend it.

### 4.3.2 Maps API

While extensive reading sessions can be a viable solution, the application uses dynamically rendered maps provided by Maps API, created by Google. While Google provides a wide range of options, Maps JavaScript API fits perfectly to the requirements of this web application. The API allows creation and web integration of custom, dynamic and interactive maps.

First, the application displays all the available Low Emission Zones right on the Home page. A dynamic list is joined with a custom map that displays the LEZs. Both elements were created with the intend of easily highlighting the LEZs. The API allows the creation of custom waypoints, which in this case, are automatically created at runtime, based on the zones queried from the database. This involves the elements to be queried in the backend but the creation of the map and population with the corresponding waypoints happens in the front end of the application.

The user has free mobility over the map, with the possibility of manually scrolling and zooming, to explore the multitude of LEZs.

Further, since the main highlight of the application is the Route Planner page, this feature would not be complete without the navigation elements. The user is guided to fill in the form with his destinations. Afterwards, the map element displays the best route between the selected points. This element is updated in real time, without the need to open a new session or refreshing the page. This is achieved in two steps. The directions are calculated inside a JavaScript function that extracts the points of interest from the page. After that, locations are checked in order to display the waypoints and connect them using the navigation feature of the Google API. This JavaScript function is called inside the asynchronous function responsible for handling the form submission. When the user clicks the *“Calculate directions”* button, the POST request containing the HTML form data is submitted.

### 4.3.3 Asynchronous function for handling form submission

JavaScript programming language allows asynchronous programming methods. This can be achieved by creating asynchronous functions. Inside the body of these functions, the *await* key word structures the asynchronous logic, suspending execution until specified conditions are fulfilled. This behaviour is called promise-based because in fact async functions return Promise objects. Based on the value, conditional programming is executed with structures like try-catch.

Such async function is automatically called by the press of *“Calculate Directions”* button. In order to create a smooth user interface experience, this function allows the page to update in real time, without refresh. First, the function gathers the user-filled information, stores the variable number of destinations in JSON format and creates the FormData element. This latter element composes the body of the POST request sent to the application backend. Then, the front-end interface waits for the backend algorithms to run and send the results. Here, another *await* key word is used to suspend the execution until the response from the backend is received. Once the JSON response can be parsed, the function execution continues to display the newly retrieved information.

### 4.3.4 Notification display

These notifications are designed to offer the key information as fast and accurate as possible. While the access validation algorithms are part of the backend, they are tightly connected to the frontend as well. After the results are calculated and transferred to the frontend, the interface must display the information in a convincing manner.

First, the text of the notifications is created by the algorithms by querying the database, as described in the previous chapter. Based on this text, the notifications are dynamically created using JavaScript. This is made by creating <div> elements and appending them to the page. All of these elements are given a class and an id based on the notification type attribute received from backend. The class and the id are used to create a suggestive look and feel for the notifications. The goal is to not overwhelm the user with information, rather to get the point across quickly. If the algorithms decide the selected vehicle is not eligible for access inside the LEZ, a red alert is raised. Color-coding the alerts or notifications is a common approach as red or green colours already have a signification encoded in the population’s mind.

In order to dynamically create the notifications, the linked CSS files define the style and positioning of the elements. This way, when rendering the notifications, the application uses the pre-defined styles based on class or id. Since the number of notifications is corelated to the number of detitanations, the flex-box approach is used to fit all notifications in a pleasant looking and easy to read manner.

The structure of the content is as follows: sentence stating the result of the verification (vehicle is eligible or not to access the LEZ), short description of the LEZ and the required registration or minimum standard, link for accessing the country info page on the application and link for accessing the official authority website. Over more, the user is directly guided to check if his vehicle is eligible to obtain the required registration.

Each selected point of interest will display a separate notification to summarise the situation.

### 4.3.5 Dynamic elements

This section pertains to the use of input elements which are conditionally displayed based on user interaction. For instance, in the webpages designated for adding or editing a vehicle, the application presents a dynamic and user-friendly form. Rather than overwhelming users with an extensive form containing the selection for all the existing registrations, which will result in numerous blank fields, the interface guides them to fill only the boxes with data relevant to their specific scenario.

For new users with minimal knowledge of Low Emission Zones (LEZs), the multitude of registrations might seem overwhelming. Therefore, they are initially guided to select a specific country before proceeding to choose the actual registrations. For example, a user from Germany may be unfamiliar with the term "Distintivo Ambiental" or Spanish Emission Stickers so the process of selecting the appropriate registration for their vehicle is made more intuitive and user-friendly.

Additionally, the notifications highlighted in the preceding section are also dynamic in nature, appearing only once the user selects the points of interest. As outlined, these notifications shortly present relevant information without suffocating users with long text paragraphs.

Furthermore, when planning trips across some countries, such as Italy, users are prompted to input the period of the trip separately. This information is very significant, particularly because of the different LEZs regulations during the winter period and is factored into the validation algorithms. However, for most countries, this requirement does not apply and therefore is not solicited.

These dynamic fields are typically accompanied by remove or delete buttons to facilitate easy correction in case of user errors or updates over time. This functionality allows users to quickly rectify mistakes or update information as needed, ensuring a smooth and user-friendly experience.

This approach aims to create an interactive form that only necessitates completion of select fields tailored to each user, thereby enhancing efficiency in user interaction while minimizing time consumption. The selective retrieval of relevant data also greatly benefits the efficiency of the algorithms and system, reducing both processing time and resource utilization.

To achieve the described behaviour, the application uses custom JavaScript functions and event-listeners to add and remove HTML elements. These functions utilize the Document Object Model (DOM) to create new elements, set their attributes, and append them to predefined containers within the HTML structure.

Additionally, there are similar functions for removing the elements if necessary. These removal functions traverse the DOM to locate and delete the targeted elements, ensuring efficient management of dynamic content on the page.

### 4.3.6 Conditional selection

While granting users freedom to navigate the application autonomously is commendable, guidance in certain selections ensures prevention of errors. This precaution is crucial as the application's primary objective is to deliver accurate results under all circumstances. Given the extensive scope of Low Emission Zones and the multitude of scenarios they encompass, it is imperative for validation algorithms to receive accurate input to function optimally.

In addition to users receiving accurate responses, thus aiding in regulatory compliance and avoiding potential fines, guiding users through certain selections also enhances their overall experience. By steering users toward valid choices, the application reduces the likelihood of errors and minimizes frustration. This smoother experience fosters greater user satisfaction and confidence in the application's efficacy, ultimately improving user engagement and retention.

Upon saving car to the profile, users are prompted to input the brand and model for identification purposes. Subsequently, after selecting the brand, the model input field is automatically populated with relevant options corresponding to the chosen brand, simplifying the selection process for users.

Similarly, the selection process for fuel type and Euro Standard follows a guided approach. Users are unable to choose a zero-emission Euro standard unless the fuel type of the vehicle is electric. Conversely, an electric vehicle can only be designated as zero-emission. While users are responsible for verifying the car specifications in the car registration documents before entering them accurately, these restrictions help to mitigate potential errors.

Implementing such constraints helps prevent illogical scenarios, such as creating a Euro 2 Electric car, which not only lacks practicality but also risks raising errors during the validation process. By enforcing these restrictions, the system promotes data accuracy and reliability, enhancing overall user experience and regulatory compliance.

Another way the application guides user interaction is on the route planner page, where users utilize autocomplete functionality to select points of interest. This is facilitated by the Places API by Google. The feature stands out as one of the most widely used autocomplete tool, as it is designed for locations prediction, significantly enhancing efficiency by suggesting places and minimizing typing time. Its versatility includes various points of interest, including cities, streets, hotels, and more.

Moreover, the use of autocomplete function ensures the validity of selected places, guaranteeing precise input for the Google Geocoding API to calculate accurate directions and generate waypoints accordingly. For instance, when selecting a hotel, the autocomplete feature automatically includes the country and city, which are then extracted by the geocoder for proper transmission to the backend. As described in the algorithm section, these two elements are considered in the access validation process. This streamlined process significantly reduces user input while optimizing functionality, resulting in a smoother user experience.

In summary, these restrictions are not implemented to hinder user access. Rather, they are designed to ensure system functionality and enhance user experience. By guiding users through selections and imposing constraints, the system ensures accurate data input, minimizes errors, and ultimately facilitates smoother operation, contributing to a more positive user experience overall.

## 4.4 Software Development Moldes and Tools

### 4.4.1 Client-Server Architecture

The Client-Server model represents one of the most used architectures in system development. This model splits the actors that take part in the interaction in two: the server is responsible for hosting and processing resources and clients which make the requests to the server. Generally, the communication takes place via the internet and allows multiple users/clients to connect to the Server at the same time. The client-server model allows system to easily collect user and information and process the data in order to provide fast responses.

The web application presented in this document is designed following the client-server model. In this case, clients are represented by users, specifically by their web browsers, which make requests to the application. The server is basically the web application, which is running on the local machine and started using the built-in Flask development mode, ideal for applications testing during the development phase. Binding the Flask server to the local IP address allows other devices to connect to the application and send requests.

When users navigate across the website and perform actions, the client sends HTTP requests to the server. The Flask backend of the application, described in the previous chapter, receives the requests and routes them to the appropriate handler functions. These functions are responsible for handling user information, whether it be a form that was submitted or a new page that needs to be rendered. Additionally, the server communicates with the database to store, and retrieve information as dictated by the client's request.

This architecture ensures that the application is scalable and maintainable and improves efficiency by centralizing processing tasks on the server. Thus, the client-server model employed in our application, even in its current local deployment, effectively leverages the strengths of both the client and server, resulting in a robust, efficient, and user-friendly web application.

### 4.4.2 Integrated Development Environment (IDE)

Integrated development environments represent one of the most basic tools for a program developer, as they extend the functionality of code writing in the form of plain text, by adding features for compiling and debugging. Their ideal for extensive programming sessions, as they improve programmer efficiency, offering code completion, refactoring or syntax suggestions based on the selected programming language [34]. For big project, like the development of a fully functional web application, where multiple programming languages are used throughout several code files, the IDEs help developers easily store, navigate and organize the program files.

Visual Studio Codes was developed by Microsoft and perfectly integrates the features of IDEs described above. Its lightweight feel and flexibility helped it become one of the most popular IDEs on the market and also make it suitable for the development of this web application. First of all, VSCode ensures smooth and short load times across multiple platforms, usually offering a smoother experience compared to other IDEs. Some the key aspects are the integrated terminal for quick debugging and commands, possibility for extensions like IntelliSense or Git Integration for easy version management, customizable interface or even live server extension for fast web development. For this application, VS Code perfectly integrates Python language, with the possibility to install libraries or frameworks using the pip command directly in the command line.

### 4.4.3 Virtual Environment

Since the application includes a multitude of python libraries which need to be imported and installed on the device that runs the server, the management can be optimised by creating a dedicated virtual environment. One of the very first steps in developing this application is represented by the setup of a new virtual environment. This process was done using a dedicated software.

Named Anaconda, this software application is designed developers in package and environment management, specifically for Python and R programming language [36]. This allows developers to create different virtual environments on the same device, which represents separate and isolated workspaces that encapsulate the prerequisite software required for an application. This isolation allows developers to effectively integrate and manage the dependencies between multiple libraries. Separating the projects to specific virtual environments ensures that the applications benefit of the required version of the libraries and also avoids conflicting packages. This approach is crucial in modern software development, allows easy bug fixing, ensures the global environment does not get corrupt from conflicting libraries but also ensures smooth upgrades.

Anaconda Prompt and Anaconda Navigator are two of the disposed tools which allow the creation and management of virtual environments. The first of them is a command-line interface which allows developers to create, manage or activate virtual environments with short keyboard inputs. On the other hand, the Navigator offers a guided user interface which simplifies the process of managing and understanding of the existing virtual environments. The integration of anaconda with Visual Studio Code allows smooth and optimal setup of the libraries needed in the application development process.

### 4.4.4 Database Management Tools

While the command line integrated in Visual Studio Code offers the possibility of opening the database file and executing SQL commands, this option lacks the versatility and the visualisation aspects offered by other dedicated software applications. For smooth and optimal modelling of the database, the development process used DB Browser, an application dedicated for management of SQLite database files. This open-source tool disposes a guided user interface for visualisation of the database. Additionally, in here developers can edit the tables, use the search function for quick data visualisation but can also use SQL commands and queries in the dedicated terminal.

During the development of this application, the database tables were created in the models.py file, using the Object-Relational Mapping properties of SQLAlchemy library. Additionally, DB Browser was used to populate the fields of the tables using SQL statements. Considering the complexity of the database model, this application was extremely helpful in data visualization, offering the possibility switch between tables but also to make slight adjustments when necessary. The application allowed easy interaction with the database, benefitting the update implementation.

Considering the fact that this application was created in Flask, another aspect that needs to be mentioned in regard to the database management is the migration scripts. This feature is part of the Flask-Migrate extension and integrates SQLAlchemy in order to facilitate database migration within the application. In essence, these migrations represent changes to the database schema, which only become effective after the commit takes place. Such modifications include adding or removing tables or modifying columns in existing tables. The scripts automatically generate and commit the migrations based on the modifications detected in the models.py file. Once generated, migration scripts are applied in a step-by-step manner to update the database schema. Flask-Migrate tracks the current state of the database schema and applies migrations sequentially to bring the database schema in sync with the defined models. One of the biggest advantages of migrations is the possibility of version control, as all migrations are saved in the migrations directory.

## 4.5 Development Techniques

### 4.5.1 REST APIs

Representational State Transfer Application Programming Interface (REST) API are responsible for the communication between the caller and the provider of a resource, using the client-server model. While in general APIs are considered a set of rules and protocols that manage the communication between the two parties, in order for an API to be considered RESTful, it must also follow a number of constraints.

The call of a REST API transfers a representation of the resource’s state to the endpoint. The endpoint refers to a specific URL, accessed by the consumer of the resource, with the scope of performing some action or retrieving some information. Typically, each endpoint has a specific resource associated which is retrieved in specific format type, via HTTP requests managed by the client-server architecture. The communication is considered stateless, as each request is separate and the client information is not stored during the GET requests [36]. REST APIs are not dependent to the programming language, rather to the principles that are behind it. Resources managed by the REST APIs should be assigned to only one endpoint and contain all the necessary information. Important information such as authorizations metadata, uniform resource identifiers (URIs) and are contained in HTTP request headers [27].

This application makes use of the REST APIs to handle POST and GET HTTP requests. The endpoints are defined to handle the retrieval of information with the GET request or to update the resource using the POST request. The transfer of the information between the client and the server uses the JSON format. For each endpoint accessed by the user, the application triggers the designated of algorithms for processing the request. This method ensures the RESTful methods, handling each request separately and serving users with the desired information.

### 4.5.2 JSON

JSON is the short abbreviation for JavaScript Object Notion, and despite the name, this format is independent from the programming language, as it can seamlessly be used in C/C++/C#., Java or Python. The JSON format offers a lightweight method of transferring data, as it is easy to read and interpret by both and humans because its structure and elements are familiar to most developers.

An element in JSON format is structured in two parts: a collection of name and value pairs, similar to dictionaries in Python or in other programming languages, and an ordered list of values, which is similar to how lists or arrays work in most programming languages [37]. In short, JSON formats data in a list of dictionaries. This format built of data structures familiar to most programming languages ensure JSON’s versatility, making it one of the most frequently used data formats for transferring information between systems.

The communication between the backend and frontend of this application enables the data transfer using the JSON format. Python offers the dedicated JSON library, which was imported in the project, together with Flask’s features to handle the creation, manipulation, retrieval and submission of data as JSON format. The frontend of the system used JSON to store and submit via POST or GET requests all the gathered user-input. In the backend, the data is unpacked by the algorithms who parse the JSON format and later process the information to compose a response. This response can contain data which needs to be submitted again to the frontend, in order to be displayed to the user. Again, the data is packed in the JSON format and this time is parsed by the frontend algorithms.

Using JSON format ensured the integrity of the data, compatibility with the two programming languages but also because of its structure, the debugging process was fairly intuitive, as the plain text of this format is readable by humans.

### 4.5.3 CRUD in Web Applications

CRUD operations are fundamental in web applications and are often used in RESTful APIs. Each endpoints provides a method for serving an HTTP request and allows clients to interact with the backend to perform these basic actions. CRUD stands for Create, Read, Update, and Delete. These are the four basic operations that can be performed on a resource of a database. These operations are critical for maintaining the integrity of databases as they ensure that data can be added, retrieved, modified. Such features must be conducted in a controlled and predictable manner, in order to ensure the development and operation of software systems.

In Flask, these operations can be implemented using Flask’s request handling, and SQLAlchemy for ORM (Object-Relational Mapping). The implementation involves defining routes for handling requests to perform database operations. SQLAlchemy makes the process of interacting with the database straightforward, ensuring that data management is efficient and organized. These CRUD operations form the essential building blocks for all web applications, enabling the creation, retrieval, updating, and deletion of data in a structured manner.

### 4.5.4 AJAX

AJAX represents a set of techniques used in web development and is short for Asynchronous JavaScript and XML. This ensures the client-server communication takes place without the need to reload the page after every HTTP requests. It is possible because the client and the server only exchange small packages of data at a time. This concept improves the overall feel of the application, minimizing interruptions as it allows only partial update of the HTML page. AJAX composes techniques like XHMTL and CSS, Document Object Model, JavaScript or XMLHttpRequests, which make it heavily dependent to the browser used by the client, as the JavaScript interpreted is integrated inside the browser and some do not offer support these techniques.

Traditionally, in web applications the user interaction is locked during the transport of process of the HTTP request. AJAX sends and receives data from the server asynchronously creating the possibility of handling the server communication without intervention form the client. This means that now users do not need to wait for the HTTP request to be completed and the interface can be updated dynamically without the need to fully reload the web page. This creates the possibility of user interaction in real-time, like click a button or complete an input field and receive and immediate response or a visible update.

The implementation of AJAX in the application presented by this document improves user experience, offering a smooth interaction with the pages, allowing users to test different scenarios of navigation without the need to refresh the page.

### 4.5.5 Document Object Model

The structure of documents can be represented as logical trees, consisting of nodes and objects using the Document Object Model (DOM). In this technique nodes contain objects and can have event handlers attached. The nodes are interconnected by the branches of the tree. Using the DOM methods the tree elements can be accessed by programmers in order to change the content or the structure of the document [39].

DOM offers different types of nodes for documents, elements, text, attributes, or comments. Objects, on the other hand are instances of classes that define the structure and behaviour of the document. Each node is represented by a JavaScript object. In JavaScript, these objects can be accessed by methods such as getElementByID() and manipulated by methods such as appendChild(), setAttribute().

By integrating the DOM and AJAX using JavaScript programming language, the application offers a seamless user experience. The combination of the two techniques ensures the dynamic and responsive nature of the application. The friendly interface allows users to directly input data in the DOM elements, which are sent to the backend of the application in an asynchronous communication. Based on the response from the backend, the JavaScript algorithms from the frontend of the system, manipulate the DOM to offer real time update without the need to refresh the page.

### 4.5.6 ORM

While the ORM was first introduced in Java in the early 2000s, the technique is now widely used across several programming languages. Python offers the possibility to integrate Object-Relational Mapping (ORM) techniques in the management of the database systems [40]. One of the most popular ORM libraries is SQL Alchemy. The ORM techniques create an abstraction layer that maps Object-Oriented elements of the programming language to relational database elements [41]. More specifically, this procedure maps classes to database tables, and objects to rows or entries in those tables. Once the fields of the tables are populated, developers can access them using object-oriented paradigm code instead of executing SQL queries.

This approach shifts the focus on application or business logic, concentrating the code required to interact with the database in the backend language. Classes mapped to tables can be easily adapted to fit future updates, resulting in an easy maintenance of the database schema.

SQL Alchemy is a very popular ORM library and is often used in the management of SQLite databases in Flask applications. The application described in this document uses this combination as the database is created by classes configured in the models.py file. The CRUD operations are handled in the endpoint handler functions, which interact with the database using Python operations like “db.session.add()” or “Zone.query.filter\_by()”.

### 4.5.7 Authentication and Authorization

Creating the account allows user to personalize his experience and interaction with the application as it enables the access to gathering information from the database. The authorization is a security critical aspect of the web application. Authentication refers to the verification of the user’s identity, while the authorization determines the actions available to an user, usually based by his role.

The authentication process typically involves verifying user’s credentials against the ones already existing in the database. Obviously, the user needs to have an account already created and use the credentials saved in the database to authenticate. As explained previously, the database stores SHA256 hashed versions of the passwords, in order to ensure the integrity and security of the application and its users. Flask provides a handful of tools for managing client authentication process. User sign-up, login and logout, all have dedicated route handlers for their endpoints.

Considering the nature of the application, the authorization process takes into account three types of users. First are the regular users, possibly first timers, as these represent clients that do not have an account created or signed in. Their access is limited by the *@login\_required* decorator associated to the route handler functions.

Secondly, are those who have already signed in with account in order to gain access to the features dedicated to the general public. This requires they would first complete the signup process on the application registration page. Each user session is handled by application backend and authorization is verified by the *current\_user* offered in Flask [42].

Lastly, the application authorizes access to a special admin user. This one has been created with the purpose of accessing and manipulating the database from the dedicated application page. The admin user role has been created for stakeholders with little technical background, as they benefit of the user-friendly page to edit the table storing data about Low Emission Zones. This role offers authorized users to make database adjustments, when necessary, without the involvement of the developer. The feature is considered to be essential in the industry, as some stakeholders should have the possibility to manage modern application. However, for security and data integrity reasons, this role must not be assigned to any person. Accessing the endpoint for the admin dashboard is only possible for users who have this authorization set as true in the database. The *User* table has a specific Boolean field that is checked by the algorithms before rendering the admin dashboard.

### 4.5.8 Templating Engine

Template engines represent tools used to generate dynamic content files based on a static template. In the case of web development, this refers to HTML files. This method allows developers to create a static HTML file, which contains placeholders for data and serves as template for other files. On runtime, or when the necessary, the placeholders are filled in with dynamic data, offered by the templating engine. This approach separates the HTML files from the backend of the application, promoting cleaner and maintainable code [43].

Jinja is a powerful templating tool for Python, and it is possible to integrate with several web frameworks but is best used in combination with Flask, as it uses Jinja as the default templating engine. The Jinja web template is an HTML file that also contains specific variables, functions and tags which help the implementation of programming logic. The syntax is very familiar to Python developers, as it’s been derived from it. Jinja offers the possibility of inheritance, which was also used in the development of the application. This means that the base template, usually named accordingly as *base.html* is extended by other HTML files, called child templates. The base file acts as the skeleton for its children files, as through extension, just like in Object Oriented Programming, all the variables and functions are passed to the children. In Jinja this includes CSS or JavaScript files linked in the head of the file or even entire HTML elements, which heavily reduces the amount of code.

As explained in chapter 4.3.1 this application makes use of this technique, in order to promote code reusability and create a website theme. The application also uses render\_template in order to display the HTML page to the user. This means the files are rendered on server side, as they HTML files are fully generated before being sent to the client. In some pages, the application combines this rendering with REST APIs for the CRUD operations, as this involves the fact that the client consumes the API, and based on the received data, updates or generates the user interface.

# 

# PART II – PROJECT CONTRIBUTIONS

# *5* Development Rationale and Methodology

## *5.1* Addressing the Issue

(existenta multor LEZ care aduc beneficia dar pot crea confuzii pentru participantii la traffic + lipsa unei resurse eficiente si personalizabile pentru a obtine informatii de interes)

## 5.2 Proposed Solution

*(Dezvoltarea unei aplicatii web )*

# 6 Development Methodology

## 6.1 Analysing Requirements and Specification

The development of this web application was structured in different stages, following the principles explained in [33].

Analysing the system requirements represents the first step in the software development lifecycle. It involves gathering, documenting and understanding the needs and expectations of the system to be developed. This step is crucial for ensuring that the final product meets the users' and stakeholders’ needs. The first step was to identify the precise subject of this application, which involved finding an opportunity for research and development. Afterwards, defining the scope and the functionality of the application was fairly intuitive: the multitude of vehicle restrictions create chaos for regular individuals, which means that the restrictions are not followed. The development of this application, meant to simplify the urban access restrictions, would solve the issue, helping users with easy compliance. Considering the fact that Low Emission Zones represent the majority of vehicle restrictions, we can reduce the requirements of the project to a software application meant to provide quick access to information regarding LEZ access. Nevertheless, the application must provide intelligible and correct information.

Creating the overall design of the application represents the next logical step towards achieving a fully functional application. This involves specifying precise function and requirements to be met, minimal database model creation and planning the next steps for the actual developments.

When it comes to requirements, the application needs to provide an interactive experience which is simple and intuitive for any user. However, in order to bring a valuable solution, the application encapsulates validation algorithms and the database model. All of these were created through extensive research for the existing Low Emission zones, in order to ensure the correctness of the data. In order to ensure quick access to information, the application lets users request data specifically based on their situation, more specifically based on their owned vehicles. That means, the application must allow users to create accounts and save vehicle specifications. In order to simplify the process, but also improve the memory and effectiveness of the application, only necessary information is stored. The database model is structured to fit these conditions. Its tables must contain information about the existing Low Emission zones or registrations but must also include tables designated for handling user information. Creating a good model from the analysis stage, improves easy implementation or updates in the future, but also creates smooth data query.

So far, only the skeleton of the application has been designed. Further, based on what has been composed, we can create a detailed design of the application. This includes deep details of the requirements, like deciding on what technologies are used, technical configuration of the application routing, delimiting the countries and cities covered by the application database, selecting information relevant for user account creation. Also, in this part, the overall look of the application and the content of the web pages was set. While the server-client model was an easy pick for the architecture, due to its nature being perfectly suitable for the web applications, this part also involved decisions regarding the basic functionality of the validation and the eligibility algorithms. For more complex pages, like the route planner, the planification also includes the sequence of events, starting from user, all the way to the response generated from backend.

Further, based on the analysis of the requirements and technical aspects described above, the development process began. The main topics composed the setup of the application backend routes, algorithm implementation and refinement and database model implementation. All of these topics were described in chapter X. The frontend elements of the website were created a bit earlier, after the API routes were already functional at a basic level. Taking this approach, allowed focus on application functionality and later on design, as this was dynamic, based on user interaction and backend response.

### 6.1.1 Information Sources

The research process behind creating this application includes different types of resources and references, all of which are mentioned in the Bibliography chapter. Considering the fact that the scope of the application covers the relatively new and innovative topic of Low Emission Zones, traditional books are generally scarce or non-existent. Nevertheless, online resources represent other credible repositories providing valuable information on this topic. The analysis and development of the application took into account the following categories of resources.

Gathering information for Low Emission Zones was primarily focused on governmental websites, like European Parliament official websites under the .eu domain, and other authoritative institutions, like official websites for National frameworks. This ensures that the application provides the newest and most accurate information. While other online resources exist, this sources generally provide the best explanation of the framework restrictions and requirements, compliance procedures, affected vehicle, often clarifying the contradictions created by the information gathered from other sources.

On the other hand, academic research or conference papers often offer information regarding the newest case studies and their findings. While the focus of this application at the moment is not to analyse and decide on the more effective Low Emission Zone framework, having an overall understanding of the topic, helps in structuring and delivering the information to the user. Other studies may reflect the future of the topic, for example the study by Statista.com, which highlights the abundance of Low Emissions Zones, based on which, the decision to create a database model that is easy to scale is obvious.

As part of the analysis process, reviewing already existing applications provides practical insights into real-world examples of how the system can be helpful for the user. From a research perspective, we can see how the information is delivered and compare this information across other sources, achieving often contradictory results. While the clarification for this issue was discussed in the second paragraph of this chapter (to opt for official websites that are up to date and accurate), the analysis of existing applications, offer additional benefits. From a technical point of view, these examples can reflect challenges faced by these applications and their solutions, but also highlight technical topics where other applications could improve on. Overall user experience, application time/resource efficiency or available features can also be analysed from a technical point of view, as they are mainly part of the application backend, as well as from a business perspective, highlighting the places where the new application can improve on.

The resources described until here in this chapter represent the foundation during the analysis part of the development, based on which the choice of the used technologies was made. While the choice, their features and how they were encompassed in the project were described in detail in chapter X, the development of the application also involves technical research. This relies on a range of resources, from traditional books on programming, to online documentation for frameworks and libraries or community support offered through trusted websites.

For the frameworks, libraries and APIs used in the web development process, like Flask or Google API, the main resource is the official documentation page. Here, developers can find the latest updates, code examples, features compatibility and work patterns but also frequent bug fixes or questions to FAQ (frequently asked questions). The database structure was modelled based on the principles and practices learned through the university, while also using the available documentation and development tools. Community support, in general offers reasonable solutions to frequent problems, development, innovation or refinement idea but is also generally more exposed to possible bugs and necessitate a good understanding of the overall application in order to include adopted futures.

## 6.2 Design elements

### 6.2.1 Application Use Cases

Once the analysis of the project requirements is complete, the next step is to design the overall schema of the system. This process is effectively structured and visualized by the Use Case Diagram. As part of the UML, the Use Case Diagram provides a visual representation of the interactions between users and the system.

In UML, the use cases are represented by an oval and should be named suggestively for those who read the diagram. The use cases represent possible scenarios of the communication between the system and external users. As part of the diagrams, these external users interacting with the system are represented by a human figure and they are referenced as actors. The possible scenarios of interaction between actors and system are represented by use cases, which represent a sequence of actions. The purpose of this sequence is to provide the user the intended action. This interaction between users and use cases can only be of association type, represented through a flat line. Use cases can be related through inclusion or extension. Inclusion means that the execution of one-use case implies the execution of another case. Exclusion means that one use case is conditionally triggered by another.

The interaction between the users and the web system descried by this document was represented in the Use Case Diagram, Figure 2. Part of the UML suite, the Use case diagram represents the interaction between users and the system. This diagram was created in the early stages of the system analysis, in order to define the functionalities of the system and how it should interact with the end-user.

As a short explanation of the diagram, the actor on the left represents regular users, who can initially access the home page and the pages designated for each country’s LEZ schema. Here, they can find general information about the application and how it works but also text-based information about the different Low Emission Zones. In order to gain full access to the features of the application, users must create accounts and save their vehicle. This is represented by the inclusion relationship, as in order to login, users must create an account. The same logic applies to the other inclusion relationships. Vehicle registration is also offered only for users with accounts created and logged in. Users are able to access the edit vehicle use case, but of course that is only possible for already saved vehicles. The second actor in this diagram, from the right side, represents the administrator of the application, who has a special account type, which allows him to edit the database of the application from the web application. Of course, the admin user is also able to use the other functions of the applications, but the visual of representation for that would be unnecessary.

A diagram of a network

Description automatically generated

Figure 2 Use Case Diagram

### 6.2.2 Activity diagram

The activity diagrams are also part of UML standard and represent the flow of control or data within the system. They are used in the modelling of dynamic aspects of a system and generally illustrate the sequence of activities triggered automatically. Activities generally represent steps in the execution of an algorithm and only have entry and exit transition associated. These transitions, are generally called flow controls, represented by arrows and are triggered automatically by the end of an activity. Additionally, in the activity diagram, developers use elements to display conditional statements. The “if” node is graphically represented by a diamond-shaped figure, with one input and multiple output transitions, each representing a different condition. Another way to add complexity in the diagram is to use concurrency elements when representing synchronization between parallel activities.

A diagram of a company

Description automatically generated Figure 3 represents the activity diagram of the Route Planner page and depicts the procedural sequence triggered by the user. After the form is submitted via the HTTP Post request, the diagram represents the asynchronous behaviour of the application, as the frontend interfaces interrupts the flow and waits for backend response. Afterwards, the notifications are displayed, which means user has received all the information he wanted, and the sequence is finished.

Figure 3 Activity Diagram

## 6.3 Non-functional Requirements

The functional requirements refer to what the application can do and in what way it interacts with the users. These requirements were described above, specifically in the Use-Case diagram and represent the utility aspects of the system. Non-functional requirements refer to application performance aspects, which in some cases may not be directly visible to the user but are essential in the functionality and stability of the system but also improve user-experience. Below, is the description of the non-functional requirements of the application as they were defined in the system analysis process.

The system needs to be scalable in order to cover future traffic regulations, a larger number of vehicle types and maintain wide traffic. The database model of the application must easily adopt to changes. Scalability includes easy maintenance of all application elements, including the adjustments needed to the validation algorithms.

In order to ensure the application is reachable to a wide audience, the application must be available across different platforms, devices or operating systems.

Real time response to user interactions should be ensured by the system, maintaining the performance even during high load conditions.

Security and integrity of the data is important. Use data should be encrypted and the system should be protected against unauthorized access to avoid data corruption.

## 6.4 Application Constraints

For the moment, this application has been developed to cover a specific niche in the market, therefore its utility is limited to meet the requirements of those individuals. Nevertheless, the effort put in the analysis of the urban regulations subject and in the development of the web application, serves as a fundamental step in creating a large-scale application, meant to become a central hub for traffic restrictions and regulations throughout Europe.

First of all, the application is limited to a single type of vehicle, specifically passenger cars. The choice to fit in these vehicles was made because they are the most widely used for personal transportation and are the ones most affected by the newly European regulations, because they are typically powered by combustion engines. Since the Low Emission Zones but also other traffic regulations impose different access criteria for each vehicle type, a viable validation system needs to also consider this aspect.

The second constraint of the web application is represented by the inclusion of only Low Emission Zones. While the urban regulations impose different strategies for reducing pollution, these ones are affecting the majority of population [1]. This results from the nature of the restrictions as the target represents the central areas in large cities but also from the wide spread of the zones. The application encompasses the Low Emission Zones, because they are the most effective form of pollution reduction in central areas and because the modelling of the different regulations and the development of a universal access validation system is a challenging aspect, which at the moment is still to be discussed.

# 7 Description of the Practical Application

## 7.1 User Guide

## 7.2 Application Verification and Validation

## 7.3 Results Obtained

# 8 Conclusions

# Bibliography

*[1] https://www.europarl.europa.eu/topics/en/article/20190313STO31218/co2-emissions-from-cars-facts-and-figures-infographics*. (n.d.).

*[2]* [*https://transport.ec.europa.eu/transport-themes/urban-transport/urban-vehicle-access-regulations\_en*](https://transport.ec.europa.eu/transport-themes/urban-transport/urban-vehicle-access-regulations_en)

*[3]* [*https://dieselnet.com/standards/se/zones.php*](https://dieselnet.com/standards/se/zones.php)

*[4]* [*https://www.mdpi.com/2071-1050/15/23/16260*](https://www.mdpi.com/2071-1050/15/23/16260)

*[5]* [*https://dieselnet.com/standards/eu/lez.php*](https://dieselnet.com/standards/eu/lez.php)

*[6]* [*https://www.germanemissionssticker.com/the-german-diesel-ban/*](https://www.germanemissionssticker.com/the-german-diesel-ban/)

*[7]* [*https://www.france.fr/en/holiday-prep/crit-air-anti-pollution-vehicle-sticker*](https://www.france.fr/en/holiday-prep/crit-air-anti-pollution-vehicle-sticker)

*[8]* [*https://urbanaccessregulations.eu/countries-mainmenu-147/france/paris*](https://urbanaccessregulations.eu/countries-mainmenu-147/france/paris)

*[9]* [*https://www.statista.com/statistics/1321264/low-emission-zones-europe-by-country/*](https://www.statista.com/statistics/1321264/low-emission-zones-europe-by-country/)

*[10]*[*https://www.london.gov.uk/who-we-are/what-london-assembly-does/questions-mayor/find-an-answer/usage-ultra-low-emission-zone-ulez-automatic-plate-recognition-cameras-crime-prevention-and*](https://www.london.gov.uk/who-we-are/what-london-assembly-does/questions-mayor/find-an-answer/usage-ultra-low-emission-zone-ulez-automatic-plate-recognition-cameras-crime-prevention-and)

*[11] Miguel Grinberg, Flask Web Development: Developing Web Applications with Python, O'Reilly Media, 2018*

*[12] Feng J, Hou H. Review of Research on Urban Social Space and Sustainable Development. Sustainability. 2023; 15(22):16130.* [*https://doi.org/10.3390/su152216130*](https://doi.org/10.3390/su152216130)

*[13]* [*https://www.w3schools.com/html/html5\_geolocation.asp*](https://www.w3schools.com/html/html5_geolocation.asp)

*[14]*[*https://www.statista.com/statistics/1241923/worldwide-software-developer-programming-language-communities/*](https://www.statista.com/statistics/1241923/worldwide-software-developer-programming-language-communities/)

*[15]* *https://survey.stackoverflow.co/2023/#overview*

*[16]* [*https://flask.palletsprojects.com/en/3.0.x/*](https://flask.palletsprojects.com/en/3.0.x/)

*[17]https://www.redswitches.com/blog/flask-vs-django/#:~:text=Netflix%3A%20Netflix%20is%20among%20the,Flask%20to%20build%20internal%20tools.*

*[18] https://www.sqlite.org/lang\_transaction.html*

*[19]* [*https://realpython.com/python-sqlite-sqlalchemy/*](https://realpython.com/python-sqlite-sqlalchemy/)

*[20]* [*https://discovery.hgdata.com/product/sqlite*](https://discovery.hgdata.com/product/sqlite)

*[21]* [*https://react.dev/*](https://react.dev/)

*[22]* [*https://angular.io/*](https://angular.io/)

*[23]*[*https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/async\_function*](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/async_function)

*[24]* [*https://nodejs.org/en*](https://nodejs.org/en)

*[25]*[*https://aws.amazon.com/what-is/api/#:~:text=APIs%20are%20mechanisms%20that%20enable,weather%20updates%20on%20your%20phone*](https://aws.amazon.com/what-is/api/#:~:text=APIs%20are%20mechanisms%20that%20enable,weather%20updates%20on%20your%20phone)*.*

*[26]* [*https://www.altexsoft.com/blog/what-is-api-definition-types-specifications-documentation/*](https://www.altexsoft.com/blog/what-is-api-definition-types-specifications-documentation/)

*[27]*[*https://www.ibm.com/topics/rest-apis#:~:text=A%20REST%20API%20(also%20called,transfer%20(REST)%20architectural%20style*](https://www.ibm.com/topics/rest-apis#:~:text=A%20REST%20API%20(also%20called,transfer%20(REST)%20architectural%20style)*.*

*[28]* [*https://api.first.org/*](https://api.first.org/)

*[29]* [*https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods*](https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods)

*[30]* [*https://flask.palletsprojects.com/en/2.3.x/tutorial/views/*](https://flask.palletsprojects.com/en/2.3.x/tutorial/views/)

[31] Permana, Iman, Mardi Hardjianto, and Kiki Ahmad Baihaqi. "Securing the website login system with the SHA256 generating method and time-based one-time password (TOTP)." Systematics 2.2 (2020): 65-71.

[32] <https://werkzeug.palletsprojects.com/en/2.3.x/utils/>

[33] Analiza si proiectarea sistemelor informatice

[34] <https://aws.amazon.com/what-is/ide/#:~:text=An%20integrated%20development%20environment%20(IDE,easy%2Dto%2Duse%20application>.

[35] <https://conda.io/projects/conda/en/latest/index.html>

[36] <https://www.redhat.com/en/topics/api/what-is-a-rest-api>

[37] <https://www.json.org/json-en.html>

[38] <https://www.ibm.com/docs/en/rational-soft-arch/9.6.1?topic=page-asynchronous-javascript-xml-ajax-overview>

[39] <https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model>

[40] Xia, Chuanlong, Guangcan Yu, and Meng Tang. "Efficient implement of orm (object/relational mapping) use in j2ee framework: Hibernate." *2009 International Conference on Computational Intelligence and Software Engineering*. IEEE, 2009.

*[41]* Chen, Tse-Hsun, et al. "An empirical study on the practice of maintaining object-relational mapping code in java systems." *Proceedings of the 13th International Conference on Mining Software Repositories*. 2016.

[42] Gaspar, Daniel, and Jack Stouffer. Mastering Flask Web Development: Build Enterprise-grade, Scalable Python Web Applications. Packt Publishing Ltd, 2018.

[43] https://jinja.palletsprojects.com/en/3.1.x/intro/

# Appendix 1. Code Snippets

@views.route('/new-car',methods=['POST','GET'])

@login\_required

def new\_car():

if request.method== 'POST':

owner\_id=current\_user.id

…

#registrations

france\_reg=request.form.get('France\_registration')

germany\_reg=request.form.get('Germany\_registration')

…

new\_car Car(owner\_id, … ,france\_reg,germany\_reg, …)

db.session.add(new\_car)

db.session.commit()

flash('Car added. ', category='Success')

…

return render\_template(“new\_car.html”,user=current\_user, …)

Ex. Code 1

@views.route('/info/<country>')

def country\_info(country):

country=country.upper()

# Here you can retrieve additional information about the selected country

# For example, you can query the database based on the country name

# Then render the corresponding HTML template for that country

from .models import Zone

list\_cities = Zone.query.filter\_by(country=country).with\_entities(Zone.city).distinct().all()

cities = [city[0].strip("()''") for city in list\_cities]

countries\_with\_lez =Zone.get\_countries()

cities\_by\_country=Zone.get\_countries\_and\_cities()

print(f"cities of {country}: ", cities)

return render\_template(f"/countries\_templates/info-{country.lower()}.html", user=current\_user,country=country, cities=cities, countries\_with\_lez=countries\_with\_lez, cities\_by\_country=cities\_by\_country)

To be considered

4.5.1 Integration of Technologies

4.5.2 Frontend and Backend Interaction

4.5.3 Responsive Design and Cross-Browser Compatibility

c