Master Thesis

Czech
Technical
University
in Prague

F3

Faculty of Electrical Engineering
Department of Computer Science

Crowdsourcing Platform for Finding Vacant Parking Spaces

Kirill Kazakov

Supervisor: Ing. Pavel Náplava, Ph.D.
Field of study: Computer Science
May 2024

MASTER‘S THESIS ASSIGNMENT

I. Personal and study details

Student's name:	**Kazakov Kirill**	Personal ID number:	**495590**
Faculty / Institute:	**Faculty of Electrical Engineering**		
Department / Institute:	**Department of Computer Science**		
Study program:	**Open Informatics**		
Specialisation:	**Software Engineering**		

II. Master’s thesis details

Master’s thesis title in English:

Crowdsourcing platform for finding free parking spaces

Master’s thesis title in Czech:

Platforma pro vyhledávání volných parkovacích míst na bázi crowdsourcingu

Guidelines:

Analyze the possibilities and methods of obtaining data on free parking spaces in Prague, which can be used for the purpose of creating a mobile application helping find a free parking space. Proceed as follows:
1) Describe parking options and methods in Prague
2) Analyze the possibilities of identifying available parking spaces
3) Search and evaluate available parking applications and systems
4) Evaluate the usability of the existing applications/systems and assess the possibilities of their modification/extension by users with application development skills
5) Design your own crowdsourcing-based method of identifying available parking spaces
6) Design a mobile application that will implement your proposed method.
7) Develop the application in the form of an MVP and verify its functions by user tests.
8) Evaluate application usability and suggest how to extend/add more functionality.

Bibliography / sources:

[1] TSK: Prague Transportation Yearbook 2022 [online]. [visited on 2024-01-20]. Available from: https://www.tsk- praha.cz/static/udi-rocenka-2022-cz.pdf
[2] R-CNN, Fast R-CNN, Faster R-CNN, YOLO — Object Detection Algorithms | by Rohith Gandhi | Towards Data Science [online]. [visited on 2024-01-20]. Available from: https://towardsdatascience.com/r-cnn-fast-r-cnn-faster-r-cnn-yolo-object-detection-algorithms-36d53571365e
[3] Crowdsourcing: Definition, How It Works, Types, and Examples [online]. [visited on 2024-02-20]. Available from: https://www.investopedia.com/terms/c/crowdsourcing.asp

Name and workplace of master’s thesis supervisor:

Ing. Pavel Náplava, Ph.D. Centre for Knowledge Management FEE

Name and workplace of second master’s thesis supervisor or consultant:

Date of master’s thesis assignment: **15.02.2024** Deadline for master's thesis submission: **24.05.2024**

Assignment valid until: **21.09.2025**

Ing. Pavel Náplava, Ph.D. Supervisor’s signature	Head of department’s signature	prof. Mgr. Petr Páta, Ph.D. Dean’s signature

III. Assignment receipt

The student acknowledges that the master’s thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the master’s thesis, the author must state the names of consultants and include a list of references.

.	
Date of assignment receipt	Student’s signature

Acknowledgements

I want to thank my supervisor, Ing. Pavel Naplava Ph.D., for his valuable advice, support, and time spent throughout the process of writing the paper.

Declaration

I declare that I have prepared the submitted work independently and referenced all the information sources used.

Prague, May , 2024

Abstract

This thesis aims to implement a crowdsourcing-based application for the identification of vacant parking spaces in Prague, focusing on enhancing urban parking management in densely populated areas. The paper analyzes existing parking solutions, human behavior, and technological advancements in the city infrastructure. It reviews current parking applications and collects user data through surveys to understand users' needs. One of the core ideas throughout this paper is the crowdsourcing integration into the smart parking context, focusing on low cost, expansive implementation, and real-time data management. The proposed approach combines user-generated content with data analysis to improve drivers' experience while searching for a vacant parking space. This community-driven strategy lays the foundation for developing a Minimum Viable Product (MVP). The effectiveness of the MVP is then evaluated through user acceptance testing involving a group of seven participants.

Keywords: smart city, parking, crowdsourcing, app development

Supervisor: Ing. Pavel Náplava, Ph.D.
Praha, Technická 2, B2-39d

Abstrakt

Cílem této práce je implementace aplikace založené na crowdsourcingu pro identifikaci volných parkovacích míst v Praze se zaměřením na zlepšení správy parkování ve městech v hustě obydlených oblastech. Studie důkladně analyzuje stávající řešení parkování, lidské chování a technologický pokrok městské infrastruktury. Analyzuje současné parkovací aplikace a sbírá preference a požadavky uživatelů prostřednictvím průzkumů. Centrální roli v této studii hraje využití principů crowdsourcingu v rámci chytrých parkovacích řešení s důrazem na nákladovou efektivitu, škálovatelnost a efektivní využití dat v reálném čase. Navržený přístup kombinuje obsah generovaný uživateli s analýzou dat s cílem zjednodušit hledání volného parkovacího místa. Tato komunitou řízená strategie tvoří základ pro vývoj minimálního životaschopného produktu (MVP). Účinnost MVP je dále vyhodnocena prostřednictvím uživatelského akceptačního testování, do kterého je zapojena skupina o sedmi účastnících.

Klíčová slova: smart city, parkování, crowdsourcing, vývoj aplikací

Překlad názvu: Systém pro vyhledávání volných parkovacích míst na bázi crowdsourcingu

Contents

Figures

Tables

Chapter 1
Introduction

In the age of rapid urbanization and technological advancement, the concept of smart cities has emerged as a visionary approach to urban development. Smart cities leverage information and communication technologies (ICT) to enhance various parts of urban services. The aim is to optimize city functions, promote economic growth, and improve the quality of life for its citizens through smart technology and data analysis. The integration of ICT solutions in urban management facilitates more efficient use of resources, resulting in a cleaner, safer, and more sustainable urban environment [1].

1.1 Problem Overview

One of the critical challenges in modern urban areas, including Prague, is the management of parking spaces. With the growing number of vehicles in cities, finding a vacant parking space has become a time-consuming and often stressful task for drivers. This issue significantly contributes to traffic congestion, as drivers circling for parking add to the volume of vehicles on the road. According to the European Parliamentary Research Service (EPRS), an estimated 30% of city traffic in Europe is caused by drivers searching for parking [2].

In Prague, like in many European cities, the problem is compounded by the historical layout of the city, which was not designed to accommodate the current volume of vehicular traffic. The limited availability of parking spaces, coupled with a growing number of vehicles, makes effective parking management a critical aspect of urban planning. The technical administration of communications reports that as of year 2022, there were approximately 1.250.000 registered vehicles in Prague with around 920 vehicles per 1000 inhabitants, highlighting the magnitude of the challenge [3].

1.2 Paper Objective

The primary objective of the Master thesis is to develop a platform in the form of the MVP that will help people identify vacant parking spots by utilizing a crowdsourcing-based approach. The final application is then tested with User Acceptance Testing (UAT).

1.3 Paper Outline

Initially, the paper will focus on a detailed description of the parking options and methods currently available in Prague. This will involve an in-depth study of the city's parking infrastructure, including street parking, public and private parking spaces, and any innovative parking solutions in place. Understanding the existing parking landscape is crucial for identifying the gaps and opportunities for improvement.

Following this, the paper will analyze the possibilities of identifying available parking spaces. This step will involve researching and evaluating various technologies and methods used globally for parking space detection, such as sensors, GPS data, and camera-based systems. The aim is to assess their applicability and effectiveness in the context of Prague's urban environment.

The next phase will thoroughly search and evaluate existing parking applications and systems. This will not only cover those currently used in Prague but also include a broader review of global best practices and innovative solutions in smart parking. The evaluation will focus on the features, accuracy, user experience, and overall effectiveness of these systems in managing parking space availability.

Subsequently, the paper will propose a unique crowdsourcing-based method for identifying available parking spaces. This method will leverage the power of community input and real-time data sharing to provide accurate and up-to-date information about parking space availability, specifically tailored to the unique characteristics of Prague.

The Master's thesis will then conduct a software analysis of the proposed solution, specifying system requirements, visualizing them using diagrams, designing user interfaces, and outlining the technology stack for implementation.

The development process of the application will be described, from setting up the database to implementing the user interface.

Finally, the application will be tested through User Acceptance Testing to validate its functionality and effectiveness.

1.4 Possible Limitations

The pursuit of this paper, while ambitious and potentially impactful, is subject to several limitations that must be acknowledged to maintain a realistic scope and expectations. These constraints range from data availability and currency to dependencies on third parties and the scale of user testing.

One of the primary limitations is the availability and currency of important data. The paper heavily relies on access to up-to-date and accurate information about parking spaces, traffic patterns, and user behavior in Prague. However, there is a possibility that some of this data may not be readily available or may be outdated. This limitation could significantly impact the accuracy of the analysis and the effectiveness of the proposed application. For instance, changes in urban infrastructure, new parking regulations, or shifts in traffic flow that are not reflected in the available data could lead to erroneous conclusions or ineffective solutions.

Another significant limitation is the dependency on third parties, particularly when it comes to implementing certain features of the application. The project's success partly hinges on cooperation with entities like the technical administration of communications companies (TSK), city authorities and open data providers (Golemio), or private parking lot operators. These organizations control critical infrastructure and data that are essential for the comprehensive functioning of the proposed application. However, gaining access to these resources or integrating with their systems may pose challenges due to bureaucratic hurdles, privacy concerns, or technical compatibility issues.

Additionally, the development and testing phase of the application is constrained by the scale of user participation. The plan to develop a Minimum Viable Product (MVP) and conduct user testing is crucial for evaluating the application's usability and effectiveness. However, the testing phase might involve a relatively small number of people, which could limit the diversity of feedback and the representativeness of the findings.

Chapter 2

Parking Options in Prague

As the city of Prague grows and modernizes, the management of parking spaces has become an increasingly pressing issue. This chapter aims to explore the diverse range of parking options available in Prague, providing a comprehensive overview that forms the foundation for understanding and addressing the city's parking challenges.

2.1 Types of Parking

Understanding the types of parking available in Prague is essential for addressing the city's parking management challenges. Parking in Prague can be categorized into two types: on-street parking and off-street parking. This section provides an overview of these two primary types of parking and the role of Prague's zoning system in managing them.

2.1.1 On-street Parking

On-street parking refers to spaces designated for vehicle parking along the city streets. These spaces are directly integrated into the urban landscape and are often the most accessible form of parking for short-term use. On-street parking in Prague is commonly found in commercial areas, near tourist attractions, and in residential neighborhoods.

The availability of on-street parking varies depending on the location and time. During peak hours and in high-demand areas, finding an on-street parking spot can be challenging. The cost for on-street parking also varies, with higher fees typically charged in central and high-demand areas. Regulations for on-street parking often include time limits to encourage turnover and availability [4].

Prague Zoning System

Prague zoning system is designed to regulate parking in various parts of the city. This system categorizes areas into different zones, each with its specific regulations and pricing. The zoning system primarily serves to prioritize residents and regulate the number of non-residential and commercial vehicles, thereby balancing the needs of locals with the city's overall traffic flow. Information regarding individual zones is publicly available on the official website of the administrator of Prague Parking Zones [5]. As shown in the table 2.1, there are 174.083 parking spaces across blue, purple, orange, and other on-street parking zones.

Table 2.1: Number of parking spaces and parking meters in ZPS areas (as of December 2022) [3]

District	Blue	Purple	Orange	Other	Payment stations
Prague 1	6 126	1 798	29	1 264	89
Prague 2	7 897	3 428	2 909	12	112
Prague 3	11 196	3 215	0	689	103
Prague 4	13 107	8 484	176	1 051	185
Prague 5	10 004	9 320	119	885	104
Prague 6	18 842	8 406	5	1 332	195
Prague 7	6 572	2 104	559	543	69
Prague 8	9 636	5 058	91	780	85
Prague 9	0	10 273	0	438	53
Prague 10	17 727	6 931	104	868	91
Prague 13	0	0	56	5	2
Prague 16	0	0	18	2	1
Prague 18	0	3 489	0	133	6
Prague 22	0	0	120	11	3
Total	**101 107**	**62 506**	**1 279**	**9 191**	**1 098**

- **Blue zone** is mainly for residential parking. Residents, business, and property owners with long-term permits can park without time limits. Besides, electric vehicles also have unlimited access to parking spaces. For other motorists, short-term paid parking is limited to one hour in the city center and three hours outside. Outside of designated operating hours, parking is free and unlimited for all. [6]

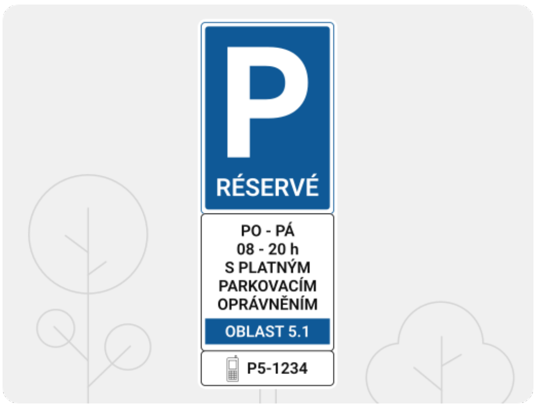


Figure 2.1: Typical blue parking sign [6]

- **Purple zone** offers similar privileges as the Blue Zone for residents and permit holders, including unlimited parking for electric vehicles. For others, it allows paid parking up to 24 hours. Operating hours are marked on signs, with free parking outside these times. [7]

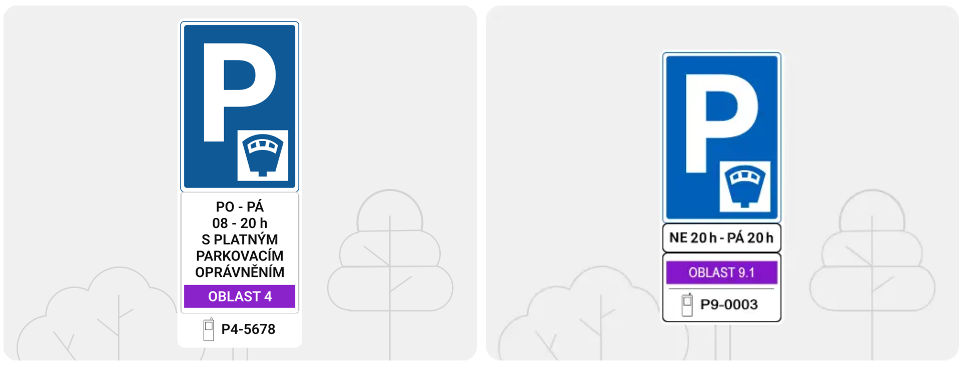


Figure 2.2: Variants of purple parking signs [7]

- **Orange zone** caters to visitors, particularly near public facilities and is not intended for residents. Long-term permits are not valid here, and parking time is limited as per traffic signs. Free parking is available outside of indicated operating hours, which vary across different districts. [8]

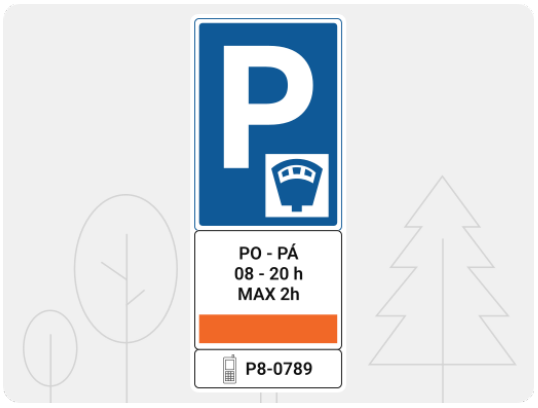


Figure 2.3: Orange parking sign [8]

Kiss & Ride

The Kiss & Ride (K&R) system is typically located near airports, ports, and public transport stops. Vehicles are allowed to stop there only for a few minutes [9].

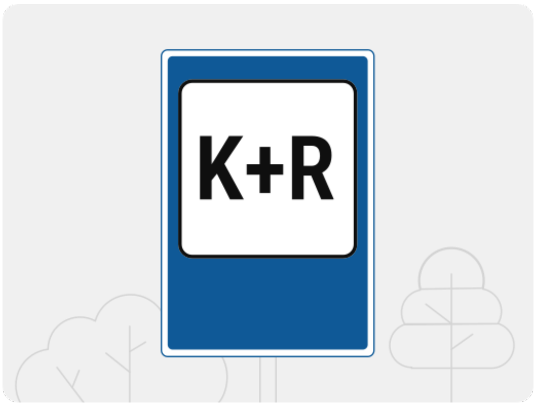


Figure 2.4: Kiss & Ride parking sign [9]

2.1.2 Off-street Parking

Off-street parking in Prague represents parking facilities that are distinct from public roads and streets, offering a structured and secure environment for vehicle parking. Off-street parking can be found near shopping centers, residential complexes, airports, and business centers [4]. Off-street parking spaces are operated by either private companies or the city of Prague. Hence, operating hours, prices, and parking rules may vary.

Park & Ride

Park & Ride (P&R) spaces in Prague provide convenient parking solutions for commuters and visitors, integrating parking with easy access to public transport. These lots are suitable for people seeking to avoid the busy city center, offering a cost-effective way to park and move around Prague using public transport. Selected parking lots may be free for up to 12 hours with an option for all-day parking for a flat rate of 50 CZK or 100 CZK [10].

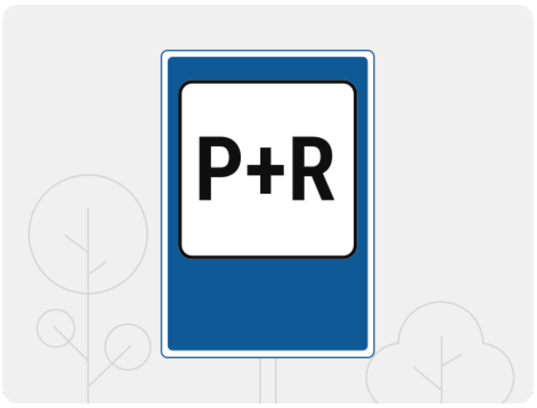


Figure 2.5: Park & Ride parking sign [10]

2.2 Conclusion

The city's on-street parking is organized into three distinct zones: residential blue zones for residents, mixed purple zones, and visitor orange zones. Additionally, Prague features the Kiss & Ride zones, designed for quick stops, mainly for passenger pick-up and drop-off. Off-street parking spaces, known as Park & Ride, are located near public transport hubs, shopping centers, residential areas, airports, and business districts. The following chapter analyzes technologies and tools used for parking occupancy detection.

Chapter 3

Detection of Parking Space Occupancy

The growing urbanization and increasing number of vehicles on the roads are key drivers of traffic congestion [11]. Traditional parking management systems need to be improved to address this challenge. This chapter delves into smart parking technologies, exploring various sensors and advanced technologies contributing to efficient parking solutions.

3.1 Smart Parking Sensors

Sensors are essential for smart parking systems, as they provide the basic data for parking availability and occupancy. By using different physical phenomena, such as sound waves, light waves, magnetic fields, or radio waves, these sensors can detect changes in the environment caused by the vehicles and convert them into electrical signals. Such signals are then transmitted to a central server or a cloud network, where they are processed and analyzed to provide parking information and services. The choice of the sensor type depends on various factors, such as accuracy, reliability, installation and maintenance cost, and environmental impact. Each sensor type has advantages and disadvantages, and no single sensor can meet all the requirements of different parking scenarios. Therefore, a combination of different sensors may be needed to achieve the optimal performance of smart parking systems. The current chapter describes popular types of sensors and evaluates their accuracy, installation complexity, and maintenance cost.

3.1.1 Ultrasonic Sensors

Ultrasonic Sensor sensors use sound waves to detect the presence of a vehicle in a parking space. They emit ultrasonic waves, and when these waves bounce back from an object (like a car), the sensor calculates the time taken for the echo to return, thereby determining the presence of a vehicle [12]. Ultrasonic sensors are known for their high accuracy and relatively easy installation. However, their performance can be affected by environmental factors like extreme temperatures and obstructions [13]. Therefore, these sensors are best suited for indoor parking areas, where they are less affected by any of the mentioned factors. Maintenance costs are moderate, typically involving sensor cleaning and calibration checks [14]. Figure 3.1 illustrates the working principle of ultrasonic sensors in the context of parking.

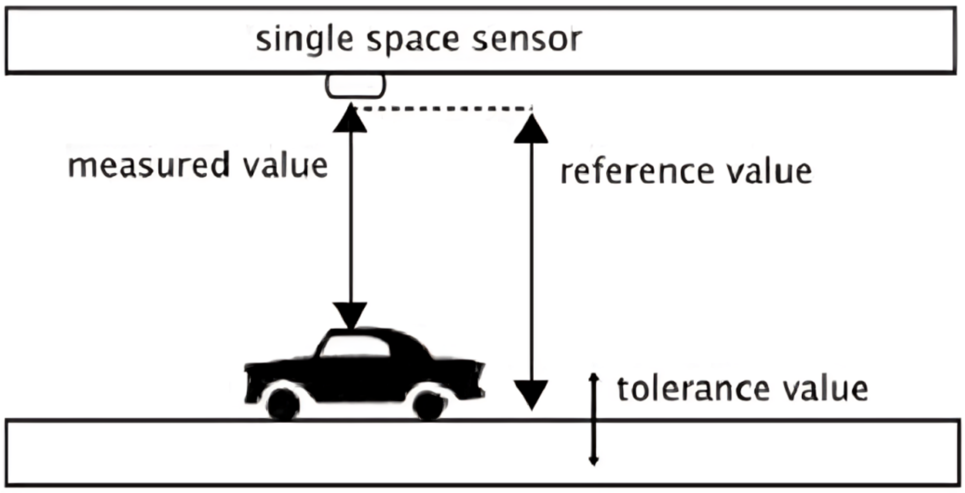


Figure 3.1: Working principle of ultrasonic sensors [12].

3.1.2 Infrared Sensors

Infrared (IR) sensors detect vehicles based on the infrared radiation or heat emitted by objects. They are typically installed in overhead structures or on pavements. IR sensors offer good accuracy and are less affected by environmental conditions than ultrasonic sensors. However, their installation can be more complex as they often require a direct line of sight to the parking space. Maintenance involves periodic checks and recalibration, and the costs are generally high [15]. Infrared sensors can operate in two modes: active or passive.

- Active IR sensors generate infrared radiation and emit signals to detect the presence of vehicles. These sensors consist of an IR emitter and a receiver. The emitter releases infrared light, and the receiver detects the reflected signal. When a vehicle is present in the parking space, it obstructs the infrared beam, causing a change in the received signal [16].
- Passive IR sensors, on the other hand, do not emit infrared radiation themselves. Instead, they rely on detecting the natural infrared radiation emitted by objects, including vehicles. These sensors consist of an infrared detector that senses changes in the thermal radiation in its field of view. When a vehicle reaches the parking lot, a sensor observes the changes and notifies about parking lot occupancy [16].

3.1.3 Magnetometers

Magnetometers work based on the principle of detecting changes in magnetic fields caused by metallic objects, such as cars [17]. This technology is employed in parking management systems to monitor and optimize parking space utilization [18]. Implementing magnetic sensors for parking occupancy detection involves embedding or installing these sensors in designated parking spaces. When a vehicle parks over a sensor, it alters the magnetic field, triggering the sensor to register the presence of a car. This data can then be collected and analyzed to determine parking occupancy. Magnetic sensors are often considered cost-effective compared to alternative technologies, such as ultrasonic or infrared sensors [19]. However, it is essential to note that magnetic sensors may be affected by adverse weather conditions, such as heavy snowfall or flooding, as these conditions can interfere with the accuracy of the sensor readings.

3.1.4 Inductive Loop Detector Sensors

These sensors are coils of wire embedded in the pavement at parking spaces or the entrance and exit points of parking areas. When a vehicle passes over or parks on the loop, the metal mass of the vehicle alters the inductance of the loop. This change is detected by the sensor, signaling the presence of a vehicle [20]. Inductive loop sensors are reliable in terms of accuracy, achieving close to 100% detection accuracy under optimal conditions. However, their installation is relatively complex and intrusive, requiring cutting into the pavement, which can be a significant drawback in certain environments [21]. The initial installation cost can be high due to the labor and materials required. Maintenance can also be challenging; the sensor can fail if the pavement is damaged or the loop wire breaks [20]. Regular inspections are necessary to ensure operational integrity.

3.1.5 Light Detection And Ranging (LIDAR)

LIDAR sensors work by emitting laser pulses and measuring the time it takes for the reflected light to return, thereby determining the distance between the sensor and objects in its environment [22]. These sensors are known for their accuracy in detecting the presence of vehicles in parking spaces. The accuracy of LIDAR sensors can be influenced by environmental factors such as weather conditions (i.e. fog), phantom reflections, or obstructions in the sensor's field of view [23]. The installation of LIDAR sensors can be more complex compared to some other types of sensors, as it requires careful placement and alignment to ensure accurate functioning. Maintenance can also be challenging, particularly if the sensors are exposed to harsh environmental conditions or physical obstructions that might affect their performance. Regular checks and calibration may be necessary to maintain their accuracy and operational efficiency.

3.2 Summary

The evaluation of various smart parking sensors indicates that while each sensor type—Ultrasonic, Infrared, Magnetometers, Inductive Loop Detectors, and LIDAR—has its specific advantages, there are also significant drawbacks that make them less suitable for universal application across all parking spaces in a city like Prague.

3.2.1 Challenges in Implementing Sensors in Prague

1. **Environmental Sensitivity**: Ultrasonic, infrared, or magnetic sensors are sensitive to environmental conditions. Prague's varied weather, ranging from hot summers to snowy winters, could significantly affect their reliability and accuracy.

2. **Installation Complexity**: For a historic city like Prague, with its protected architectural sites, the intrusive installation process of inductive loop detectors, which requires cutting into pavement, might not be feasible or desirable in many areas.

3. **Cost Considerations**: The high initial installation costs of radars and the maintenance expenses of ultrasonic sensors could be economically burdensome for a large-scale deployment across the city.

Given these challenges, integrating sensors in over 170,000 parking lots in Prague as shown in the table 2.1 might not be the most practical or cost-effective solution. The varied landscape, weather conditions, and the historical significance of the city's infrastructure require a more adaptable and less intrusive approach.

An alternative to traditional sensors is the use of Computer Vision technology coupled with object detection algorithms. This approach involves using cameras installed at many locations to monitor parking spaces. The cameras are connected to a central system that uses advanced algorithms to detect the presence of vehicles in real-time.

3.3 Computer Vision

An alternative option for detecting empty parking spaces is using data from photo and video cameras. In the pursuit of optimizing urban mobility, this approach leverages advanced computer vision technologies to detect and monitor free parking spots in real-time. This method not only enhances the overall efficiency of parking infrastructure but also contributes to reduced traffic congestion, improved user experience, and more sustainable urban environments.

3.3.1 Deep Learning in Parking Space Detection

The current section depicts commonly used algorithms for object detection that can be applied to detect free parking spots. Such algorithms apply deep learning techniques, specifically convolutional neural networks (CNNs), for various computer vision tasks. *Deep learning* is a branch of artificial intelligence (AI) that mimics the human brain's neural networks, enabling computers to learn and make decisions independently. It involves training computer models to recognize patterns and features in data, allowing them to perform tasks that traditionally require human intelligence. Deep learning is especially useful for handling complex data types like images, text, and sound [24]. In the context of parking space occupancy detection, deep learning is like teaching a computer to "see" and understand whether a parking space is vacant or taken by training it with a large number of examples. Once trained, the system can quickly analyze real-time data and help people find available parking spaces. It is like having a smart assistant for parking that can make the process more convenient and efficient.

In order to make systems work correctly and precisely, cameras are placed in a parking lot, continuously capturing images or data related to each parking spot. The collected data is then processed by various algorithms and utilized for training a deep learning model to recognize patterns associated with empty and occupied parking spaces. Once the initial training phase is complete, the system is deployed to work in real-time, analyzing incoming image data from a parking lot. The deep learning model may continually refine its understanding of different parking scenarios as it processes more data over time to increase accuracy in parking occupancy detection.

3.3.2 Modern Object Detection Algorithms

Region-Based Convolutional Neural Networks

While Faster R-CNN and Mask R-CNN are indeed more suitable for parking occupancy determination due to their improved speed and accuracy, it is crucial to mention their predecessors - R-CNN and Fast R-CNN to provide a better understanding of work of algorithms within the R-CNN family.

- R-CNN was one of the pioneering algorithms for object detection. It employs a selective search algorithm to propose regions of interest (RoIs) within an image and subsequently utilizes a Convolutional Neural Network (CNN) to classify and refine these proposed regions [25]. Although R-CNN is a simple and effective algorithm for object detection, it is still computationally expensive and time-consuming due to its sequential processing of each region.

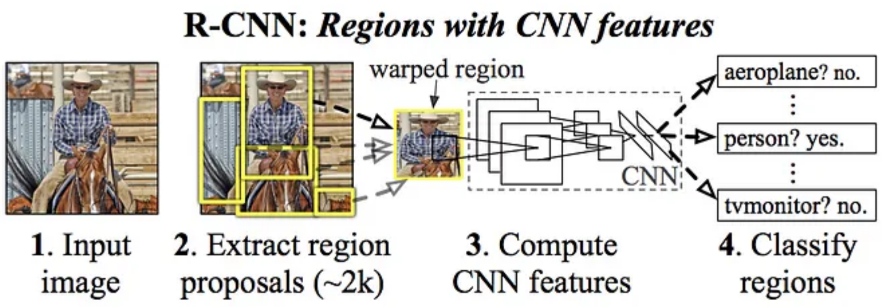


Figure 3.2: RCNN [25]

- Fast R-CNN introduces improvements on R-CNN by sharing the convolutional layers for feature extraction across all proposed regions. This modification allows for the elimination of redundant computations and accelerates the object detection process. Instead of processing each region independently, Fast R-CNN processes the entire image simultaneously, improving speed and computational efficiency. Similarly to its predecessor, it uses a selective search algorithm to generate region proposals [26].

- Faster R-CNN takes efficiency further by introducing a Region Proposal Network (RPN) that generates region proposals within the neural network, eliminating the need for an external algorithm. This enhancement significantly speeds up the detection process and enhances accuracy. The algorithm is much faster than its predecessors (Figure 3.3) and can be used for real-time object detection [27].

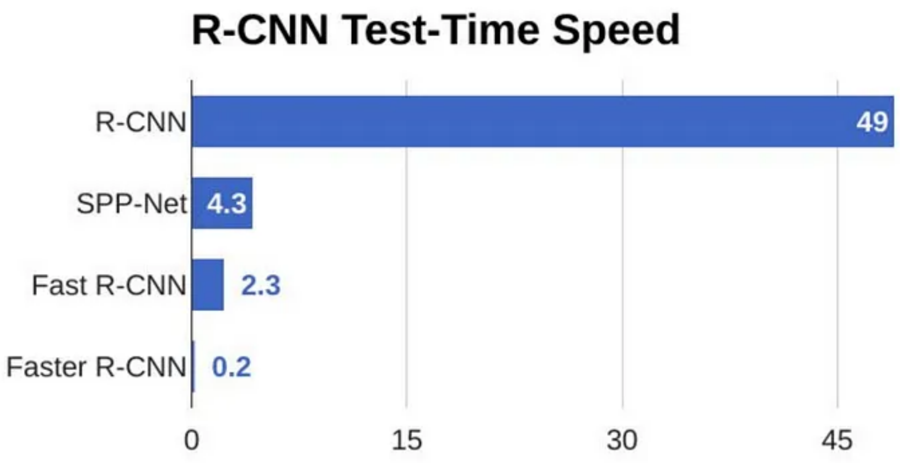


Figure 3.3: RCNN Speed Comparison [25]

- Mask R-CNN, an extension of Faster R-CNN, introduces an additional task: instance segmentation. In addition to predicting bounding boxes and class labels, Mask R-CNN also predicts pixel-wise segmentation masks for each object [28].

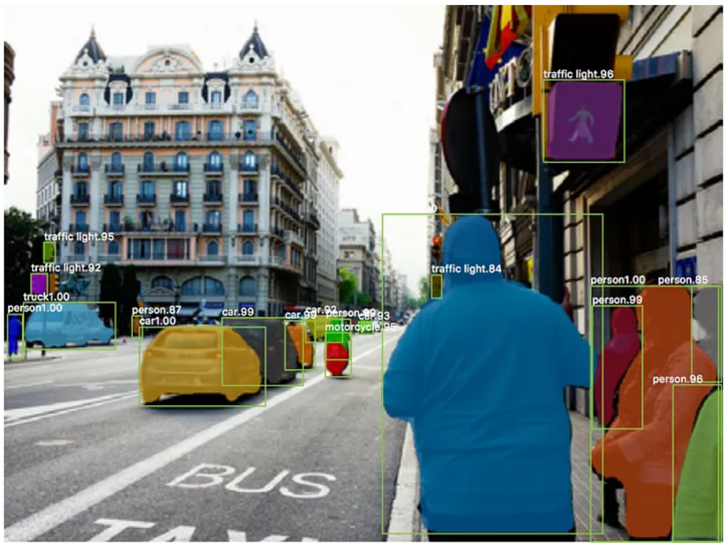


Figure 3.4: Mask R-CNN [25]

Feature Pyramid Network (FPN)

Feature Pyramid Network is not an object detection model but a feature extraction method. It enhances the feature pyramid of an image to improve the performance of object detection models [29]. Faster RCNN with FPN is a common combination in modern object detection systems.

Single Shot Detection (SSD)

Single Shot Detection is an object detection methodology designed for real-time applications, offering a substantial speed improvement over traditional approaches like Faster R-CNN. Unlike Faster R-CNN, which relies on a time-consuming region proposal network (RPN), SSD achieves its efficiency by integrating multi-scale features and default boxes directly into its architecture. By eliminating the need for a separate proposal step, SSD significantly boosts frames per second, achieving nearly five times the speed of Faster R-CNN while maintaining competitive accuracy. Multi-scale features allow the detector to adapt to objects of varying sizes, while default boxes serve as references for accurate localization across different scales. SSD stands out as a powerful solution for real-time object detection tasks, balancing speed and precision effectively [30].

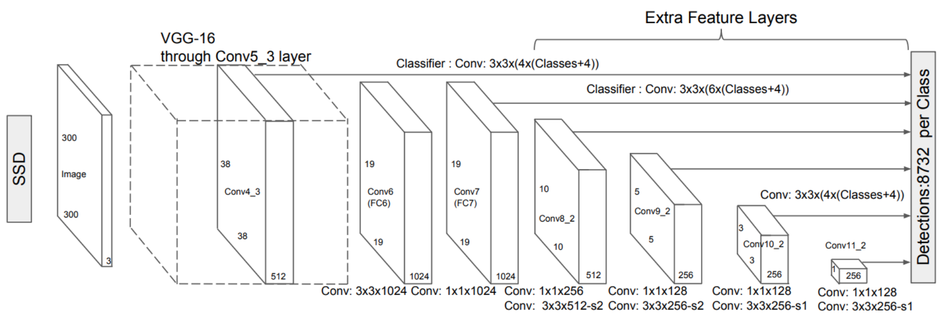


Figure 3.5: Working process of SSD [30]

You Only Look Once (YOLO)

You Only Look Once (YOLO) is one of the most popular object detection models for object detection in images (including real-time). Unlike other object detection solutions that use region proposal networks and multiple stages to identify objects, YOLO divides the input image into a grid and predicts bounding boxes and class probabilities directly for each grid cell. The convolutional network simultaneously predicts multiple bounding boxes and their associated class probabilities, efficiently detecting multiple objects in a single pass. With this approach, YOLO achieves high processing and computation speed and maintains high accuracy [31].

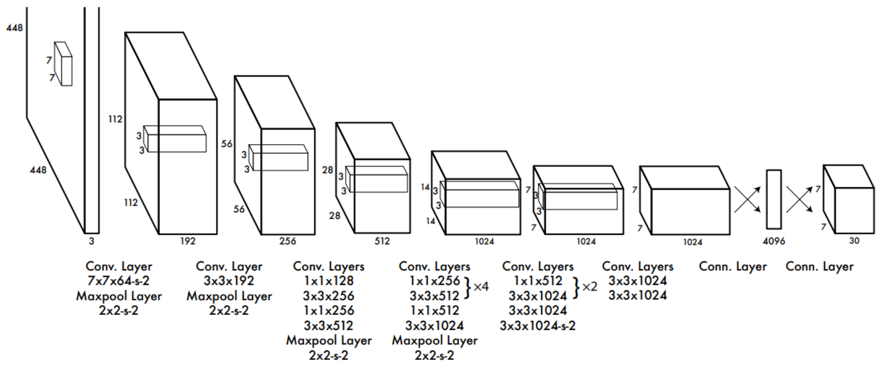

Conv. Layer 7x7x64-s-2 Maxpool Layer 2x2-s-2
Conv. Layer 3x3x192 Maxpool Layer 2x2-s-2
Conv. Layers 1x1x128 3x3x256 1x1x256 3x3x512 Maxpool Layer 2x2-s-2
Conv. Layers 1x1x256 3x3x512 ×4 1x1x512 3x3x1024 Maxpool Layer 2x2-s-2
Conv. Layers 1x1x512 3x3x1024 ×2 3x3x1024 3x3x1024-s-2
Conv. Layers 3x3x1024 3x3x1024
Conn. Layer
Conn. Layer

Figure 3.6: Working process of YOLO [31]

RetinaNet

RetinaNet is a highly efficient and accurate object detection framework known for addressing the challenge of class imbalance, a common issue in object detection tasks. This is achieved through its novel component, the Focal Loss function, which significantly improves the accuracy in scenarios where the presence of objects is sparse compared to the background. RetinaNet combines the benefits of two main types of architectures: the speed and efficiency of one-stage detectors like SSD and YOLO and the accuracy of two-stage detectors like Faster R-CNN. It utilizes a Feature Pyramid Network (FPN) alongside a ResNet-based backbone, enabling it to detect objects at various scales efficiently. It is particularly suitable for diverse and dynamic environments like those in smart parking systems. The architecture of RetinaNet ensures that it not only performs detection tasks swiftly but also with a high degree of precision, especially in detecting small or hard-to-notice objects [32].

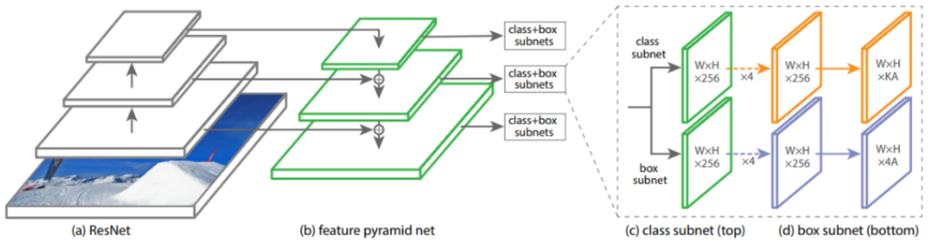


Figure 3.7: Working process of RetinaNet [32]

MobileNet

MobileNet is a streamlined architecture for mobile and edge devices, emphasizing efficiency to enable deployment in applications with limited computational resources. Its core innovation lies in using depthwise separable convolutions, which significantly reduce the number of parameters and the computational burden compared to standard convolutions used in larger models. This approach involves splitting the convolution process into two layers: a depthwise convolution and a pointwise convolution. The depthwise convolution applies a single filter per input channel, and the pointwise convolution, a simple 1x1 convolution, is used to create a linear combination of the output of the depthwise layer. This design choice makes MobileNet lightweight and maintains a reasonable level of accuracy, making it suitable for real-time applications [33].

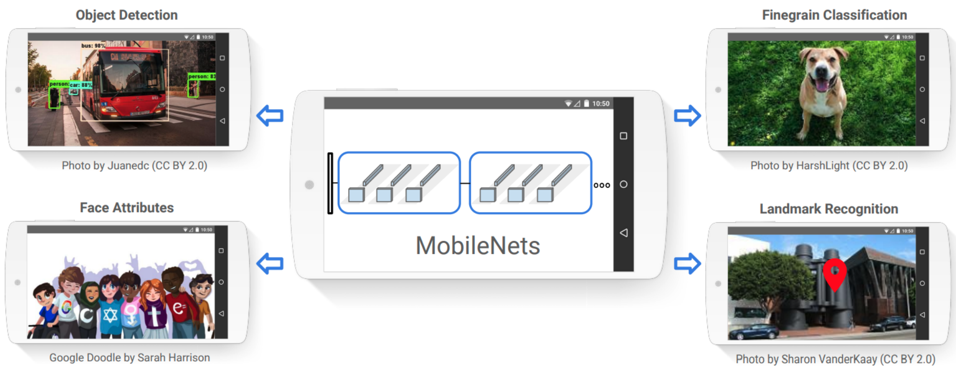

Object Detection
Photo by Juanedc (CC BY 2.0)
Face Attributes
Google Doodle by Sarah Harrison
MobileNets
Finegrain Classification
Photo by HarshLight (CC BY 2.0)
Landmark Recognition
Photo by Sharon VanderKaay (CC BY 2.0)

Figure 3.8: MobileNet models applied to different tasks [33]

3.3.3 Conclusion

To summarize, the most appropriate algorithms for parking occupancy detection are Faster R-CNN or Mask R-CNN, SSD, and YOLO due to their balance of speed and accuracy. Faster R-CNN, with its Region Proposal Network, offers a robust solution for accurately detecting parking spaces. However, its computational intensity might be better for real-time applications. SSD and YOLO, on the other hand, provide the necessary speed for real-time detection, making them more suitable for applications where immediate occupancy information is essential. They achieve this efficiency by eliminating the need for a separate proposal step and predicting multiple bounding boxes and class probabilities directly from the image. RetinaNet and more advanced algorithms, while efficient and accurate, might be unnecessary for the relatively straightforward task of parking space detection.

While computer vision and object detection algorithms present a sophisticated approach for detecting empty parking spaces, using them exclusively for every parking in Prague may not be the most feasible solution. One of the primary limitations is that not every parking space in Prague is equipped with cameras. Installing and maintaining cameras throughout the city would involve significant investment in infrastructure and resources. Additionally, continuous monitoring of public spaces may raise privacy concerns.

Moreover, the complexity and variability of an outdoor environment pose challenges for computer vision systems. Factors like varying light conditions, weather changes, and different vehicle sizes and colors can affect the accuracy of these systems. While algorithms like CNNs, SSD, YOLO, and others are powerful, they require extensive training with diverse datasets to handle such variability effectively. This training is resource-intensive and requires ongoing adjustments to maintain high accuracy.

3.4 Crowdsourcing

An alternative and complementary solution to smart parking systems is crowdsourcing. Crowdsourcing, in the context of urban mobility and smart parking, involves leveraging the collective input from a large number of people, typically through mobile apps or web platforms, to gather and share information about parking availability [34]. This approach is based on the principle that individuals, especially drivers, can contribute real-time data about the occupancy status of parking spaces.

A crowdsourcing solution could work as follows: drivers use a mobile app to indicate when they occupy or leave a parking spot. This information is then aggregated and analyzed to provide real-time data on parking availability to other users of the app. Such a system could also integrate reports from pedestrians or shop owners in busy areas, who can provide additional data points.

3.4.1 Core Principles of Crowdsourcing in Smart Parking

- **Community Participation**: Crowdsourcing relies heavily on active participation from a community of users. In the case of smart parking, drivers and pedestrians can report the availability of parking spaces.
- **Real-Time Data Collection**: It allows for the accumulation of real-time data, which is particularly useful for dynamic environments like city streets where parking availability can change rapidly.
- **Scalability and Cost-Effectiveness**: This method can be more scalable and cost-effective compared to installing physical sensors or camera systems, as it primarily depends on user-generated data.
- **User Engagement and Incentivization**: Successful crowdsourcing platforms often incentivize users to contribute data, ensuring a steady stream of accurate and timely information.

In conclusion, while computer vision and object detection algorithms offer advanced solutions for parking occupancy detection, their universal application in Prague is limited by the lack of necessary infrastructure and potential privacy concerns. On the other hand, a crowdsourcing-based approach can enhance smart parking systems by leveraging community participation and real-time data, offering a scalable and cost-effective solution for urban parking challenges.

3.5 Parking Occupancy Predictions

While the primary objective of smart parking systems is to assist drivers in locating available parking spots [35], there is an additional layer of utility that can be integrated into these systems: parking occupancy predictions. Providing information about parking occupancy from sensors and computer vision may be applied to only those people who are relatively close to a parking lot. Predictions extend the utility of smart parking solutions beyond immediate, real-time information, offering foresight and planning capabilities for drivers. Predictive features in smart parking systems are especially beneficial for drivers planning their trips in advance. For those who are not in the immediate vicinity of a parking lot, predictions provide valuable insights into the likelihood of finding a parking spot upon arrival. This is particularly useful in dense urban areas like Prague, where parking availability can vary significantly throughout the day [36].

3.5.1 Utilization of Mobile Network Data

Mobile network data can enhance the prediction of parking space occupancy by leveraging the movement and presence of mobile devices within a specific area. By analyzing anonymized signaling data from these devices, it is possible to monitor patterns and behaviors related to vehicular movement and parking.

This data-driven approach allows for real-time assessment of street-level occupancy without physical sensors or cameras. The predictive algorithms can then identify trends and provide accurate predictions on parking availability. This method ensures compliance with privacy regulations like GDPR while offering a scalable and efficient solution for urban parking management [37]. Moreover, standardized signaling data enables the deployment of such systems across various regions and countries. Such an approach was implemented in the eParkomat mobile application [1]. At the time of writing the thesis, the application was not available for Android or iOS platforms.

3.5.2 Advantages

- **Enhancing User Experience and Urban Mobility**: Accurate predictions can significantly improve the user experience by reducing the time and stress associated with finding a vacant parking space. This efficiency can also contribute to reduced traffic congestion, as drivers spend less time circling around searching for parking.

- **Supporting Smart City Initiatives**: Parking occupancy predictions align well with broader smart city objectives, such as reducing carbon emissions and improving urban planning. By optimizing parking space utilization, cities can reduce the environmental impact of vehicles and plan urban spaces more effectively.

3.5.3 Implementation Considerations

- **Accuracy and Reliability**: Ensuring the accuracy of predictions is important, as inaccurate information can lead to frustration and reduced trust in the system [38].

  The accuracy of parking predictions can be enhanced by considering various temporal factors. Differentiating between workdays, weekends, and holidays is crucial, as parking patterns vary greatly. For example, a business district might have high occupancy during weekdays but low demand on weekends. Conversely, entertainment or tourist areas might experience the opposite trend.

  Another way of increasing predictions accuracy is by integrating data from multiple sources, including historical occupancy data, real-time information from sensors or computer vision systems, and even crowd-sourced data. Advanced algorithms can analyze these data points to identify patterns and predict future parking space availability [36].

- **Data Privacy and Security**: Handling large volumes of data, especially when integrating crowdsourced information, raises concerns about user privacy and data security.

[1]eParkomat Website: https://www.eparkomat.com/

- **Technological Integration**: Seamlessly integrating predictive features with real-time occupancy data requires robust technological infrastructure and advanced analytical capabilities.
- **Cost**: Developing and maintaining predictive systems involves significant financial investment, particularly in advanced analytics and infrastructure.

In conclusion, incorporating parking occupancy predictions into smart parking systems significantly benefits urban mobility and aligns with smart city goals. However, this integration demands attention to accuracy, privacy, technology, and costs. While predictive analytics improve planning and reduce parking search times, successful implementation requires balancing the financial investment against the efficiency and sustainability gains in urban transportation.

3.6 Conclusion

In conclusion, this chapter has thoroughly explored the various technologies and methodologies for detecting parking lot occupancy, highlighting their potential applications and limitations within the context of Prague's urban landscape. From the detailed examination of smart parking sensors like ultrasonic, infrared, magnetometers, inductive loop detectors, and LIDAR, to the innovative use of computer vision and crowdsourcing, this chapter underscores the diversity and complexity of solutions available for smart parking systems.

Each approach, whether sensor-based, reliant on advanced object detection algorithms, or leveraging the power of community through crowdsourcing, presents its own set of advantages and disadvantages. The integration of these technologies into a cohesive system is not just a matter of technical feasibility but also involves considerations of cost, environmental impact, urban infrastructure, and user privacy.

Crowdsourcing was chosen for this paper due to its cost-effectiveness, scalability, and applicability for managing a large number of parking spaces in Prague. Unlike sensor-based or computer vision technologies that require significant changes to city infrastructure and initial high investment, crowdsourcing leverages the collective input of individuals through technology they already use, like mobile apps. This method not only reduces the financial and logistical burdens associated with implementing a smart parking system but also encourages community participation and real-time data sharing.

Moreover, the scalability of crowdsourcing makes it particularly appealing for a city like Prague, which has a complex urban layout and a significant number of parking spaces. It allows for gradual expansion and adaptation, accommodating the diverse needs of different areas within the city.

In summary, crowdsourcing stands out as the most practical and feasible approach for Prague. It aligns well with the city's requirements and constraints, offering an efficient community-driven solution to the challenges of urban parking management.

The next chapter analyzes the functionalities that need to be incorporated into the future MVP based on the INRIX parking survey and examination of already existing solutions.

Chapter 4

Requirements Elicitation

This chapter focuses on eliciting requirements for the development of a Minimum Viable Product (MVP) for a smart parking application based on crowdsourcing. The primary aim is to identify core features that will address the specific needs of users in urban parking environments, particularly in Prague. The process involves analyzing existing solutions, understanding user preferences, and exploring the potential of crowdsourcing in smart parking.

4.1 Survey

To gain insights into user preferences and pain points in parking, a survey from INRIX was studied [39]. The INRIX parking survey provides insightful data on parking challenges faced by drivers in major cities, including those in the United States, United Kingdom, and Germany. The survey highlights the significant amount of time and resources wasted in search of parking spaces, the economic and non-economic costs associated with parking, and the general parking experience of drivers.

4.1.1 Most Desired Functionalities

The survey findings offer valuable insights into the functionalities desired by more than 18,000 respondents in a smart parking application. The key functionalities include:

1. **Real-Time Parking Availability**: A substantial majority of respondents (around 90%) expressed a strong preference for having ability to see available parking spots in real-time.

2. **Finding and Comparing Closest and Cheapest Parking**: Almost 90% of respondents showed interest in being able to locate and compare parking options based on proximity and cost.

3. **Navigation Directly to an Available Space**: Nearly 90% of drivers preferred a navigation feature that directs them straight to an available parking space, further decreasing the time needed to find a parking spot.

4. **Reservation Capability**: A significant 75-85% of respondents were interested in the ability to reserve parking spots through an application or service, a feature that ensures parking spot availability upon arrival.

5. **Advanced Payment**: More than 70% of respondents want to be able to pay in advance.

4.1.2 Other Findings

Along with the most desired application features, the survey provides information regarding parking payment preferences, technology delivery method, and payment for the use of parking application services.

Payment Preferences

Results on parking payment preferences can be observed in the table 4.1. Overall, across all countries, there is a nearly even split between traditional methods (cash and card at a machine) and digital methods (integrated into navigation and mobile apps), highlighting a shift towards more technologically integrated payment options in parking. This even split suggests that incorporating an in-app payment feature in the parking application would be beneficial for a significant portion of users. However, with a substantial number of people still preferring to pay through payment machines, it is essential for the parking application to also provide information on the locations of such payment machines.

Table 4.1: Parking Payment Preferences [39]

Country	Cash at a Machine	Card at a Machine	Payment via Navigation App	Separate Mobile App
U.S.	15%	29%	28%	28%
U.K.	28%	27%	20%	25%
Germany	39%	22%	22%	17%
Average	27%	26%	23%	24%

Technology Delivery Method

As shown in the table 4.2, in-car system integration emerges as the most preferred method at 48% across all surveyed countries, followed by navigation apps (28%) and standalone apps (24%). Due to the potential lack of partnerships and technical feasibility, integrating the smart parking solution into car navigation systems or popular navigation apps may not be possible at the moment of writing the thesis. However, the preference for "In Standalone App" is still notable, with around a quarter of respondents across all countries showing interest. Thus, developing a standalone mobile app for smart parking remains a viable solution.

Table 4.2: Parking Technology Delivery Method Preferences [39]

Country	In Navigation App	In Standalone App	In Car's Navigation System
U.S.	34%	23%	43%
U.K.	25%	27%	49%
Germany	27%	22%	52%
Average	28%	24%	48%

Payment for the Application

While a mobile application is a cost-effective solution compared to installing parking sensors or integrating vehicle detection technologies, it might still require investment sources. Across all the countries, a significant portion of respondents (43%) are not willing to pay for parking apps. However, a notable 28% are open to a one-time fee, suggesting an opportunity for monetization. Subscription services are less popular, with only 6% acceptance. Interestingly, alternative payment models like a percentage of the parking fee (11%) and advertising-supported apps (14%) also show potential. This diversity in payment preferences suggests that while free services are preferred, there are multiple avenues for monetization, including one-time fees and advertising, that could be viable for a portion of the user base.

Table 4.3: Preferences for Payment for the Parking Application

Country	One-Time Fee	Subscription Service	Free	% of Parking Fee	Ads
U.S.	27%	9%	40%	11%	14%
U.K.	27%	5%	46%	11%	14%
Germany	30%	4%	42%	10%	14%
All Countries	28%	6%	43%	11%	14%

4.1.3 Summary

The INRIX survey provides vital insights for developing a smart parking application, highlighting user preferences for features like real-time parking availability, cost comparison, in-app navigation, reservation capabilities, and advanced payment options. Additionally, it emphasizes the need to include locations of traditional parking payment machines, catering to users preferring conventional payment methods. Despite a general preference for free apps, the survey suggests potential revenue streams through in-app payment systems, advertisements, and one-time or periodic fees. This combination of essential features and strategic monetization approaches is crucial for creating a comprehensive and sustainable smart parking solution.

4.2 Smart Parking Mobile Applications

An analysis of existing smart parking mobile applications, both worldwide and specifically in Prague, will be conducted to identify common features and best practices.

4.2.1 Worldwide

This subsection explores popular smart parking applications in countries other than the Czech Republic. It aims to identify innovative features and successful strategies that have been implemented in these applications, which could be adapted or improved upon for the Prague context. After comparing a total of three worldwide smart parking applications, an analysis was conducted to identify and compare their main functionalities. Based on this analysis, a table has been created that depicts the key features of each application. This table serves as a useful tool to understand how these applications address various parking needs and preferences, aligning with the functionalities most desired by users as identified in the INRIX survey.

Table 4.4: Comparison of Main Functionalities in ParkMobile [40], SpotHero [41], and ParkWhiz [42]

Functionality	ParkMobile	SpotHero	ParkWhiz
Displaying Parking on the Map	✓	✓	✓
Displaying Parking in the List	✓	✓	✓
Displaying Parking Details	✓	✓	✓
Reservation in Advance	✓	✓	✓
Mobile Payment Options	✓	✓	✓
Zone Parking Payments	✓	✓	✓
Parking Time Extensions	✓	✗	✓
Parking History and Receipts	✓	✓	✓
Parking Reviews	✗	✓	✗
Current Parking Indication	✓	✓	✓
Parking Searching	✓	✓	✓
In-App Navigation	Through 3rd party app	✗	Through 3rd party app
Parking Filters	✗	✓	✓
Real-time Parking Capacity	✗	✗	✗
Parking Photos	Only off-street parking spaces	Only off-street parking spaces	Only off-street parking spaces
Adding Parking to Favorites	✗	✗	✗

In summary, while ParkMobile [40], SpotHero [41], and ParkWhiz [42] offer a robust set of functionalities that cater to most of the primary parking-related needs, there is room for enhancement. Particularly in terms of providing real-time parking capacity data and leveraging user-generated content for parking reviews. Addressing these gaps could lead to a more comprehensive and user-centric smart parking experience.

4.2.2 Prague

Following the same analytical approach used for evaluating worldwide applications, a thorough research on parking apps available specifically in Prague was conducted. This research aimed to understand and compare the functionalities of popular local parking applications, considering the unique parking needs and urban dynamics of Prague. In total, 4 applications were used for analysis: CityMove [43], EasyPark [44], PidLitacka [45], and MPLA.io [46].

Table 4.5: Comparison of Main Functionalities in CityMove [43], EasyPark [44], PidLitacka [45], and MPLA.io [46]

Functionality	**CityMove**	**EasyPark**	**PidLitacka**	**MPLA.io**
Displaying Parking on the Map	✓	✓	✓	✓
Displaying Parking in the List	✓	✓	✗	✓
Displaying Parking Details	✓	✓	✓	✓
Reservation in Advance	Only P&R	✗	✗	✗
Mobile Payment Options	✓	✓	✓	✓
Zone Parking Payments	✓	✓	✓	✓
Parking Time Extensions	✓	✓	✓	✓
Parking History and Receipts	✓	✓	✓	✓
Parking Reviews	✗	✗	✗	✗
Current Parking Indication	✓	✓	✓	✓
Parking Searching	✓	✓	✓	✓
In-App Navigation	Through 3rd party app	✗	Through 3rd party app	✗
Parking Filters	✓	✗	✓	✓
Real-time Parking Capacity	✗	✗	✗	✗
Parking Photos	Only off-street parking spaces	✗	✗	✗
Adding Parking to Favorites	✓	✗	✓	✓

4.3 Crowdsourcing Applications

Studying current platforms that rely on user-generated content is essential to create a smart parking application that utilizes crowdsourcing. This analysis will help understand effective ways to encourage users to review parking and maintain current information about prices, opening hours, and other relevant parking details. The Waze application was chosen for the analysis for this paper as it contains a rich spectrum of crowdsourcing-based features.

Waze

Waze [47], a popular navigation app, employs crowdsourcing as a core part of its functionality to enhance the driving experience for its users. This approach involves leveraging real-time data from users to provide up-to-date information on various aspects of driving and navigation. Crowdsourcing is used in the following features:

- **Traffic Updates**: Users contribute real-time information on traffic conditions, such as traffic jams, road closures, or accidents. This information helps other users to avoid delays and choose the most efficient routes.
- **Police Presence**: Drivers can report the location of police checkpoints or speed cameras, allowing others to be aware and drive more cautiously in those areas.
- **Accurate Mapping**: Through user contributions, Waze continuously updates its maps, making them more precise and reflective of current road layouts and conditions.

Waze motivates its users to contribute through various engaging and interactive methods. These strategies are designed to create a sense of community and make the process of contributing both rewarding and enjoyable:

- **Gamification**: Waze uses gamification techniques to encourage user participation. Users earn points for reporting traffic incidents, hazards, and other road conditions. These points can lead to higher user levels, badges, and recognition within the Waze community.
- **Customizable Avatars**: Users can personalize their experience with customizable avatars, which become more elaborate as they earn more points. This personalization adds a fun element to using the app.
- **User Rankings**: Waze includes a ranking system where users can see how they stack up against others in their area or globally. This competitive aspect motivates users to contribute more to improve their rank.
- **Direct Communication**: Allowing users to send messages to other drivers can create a more interactive and engaged community. Users could share traffic updates, parking tips, or even safety warnings directly with others nearby.

4.4 Conclusion

The conducted research lays a strong foundation for developing a new crowdsourcing-based smart parking application. Based on the INRIX survey findings and the analysis of the existing applications, the proposed solution will leverage user-generated content to provide up-to-date information on parking space availability, pricing, and conditions. Detailed implementation, including technical specifications and development strategies, is discussed in the subsequent sections of the Master thesis.

Chapter 5
Software Analysis

The previous chapter depicted the features of existing smart parking and crowdsourcing applications. Based on the findings, a new mobile application will be developed to reduce drivers' time spent searching for vacant parking spaces. By offering real-time information on the availability and status of nearby parking spaces, this application will save time and money and contribute to environmental conservation.

5.1 Overview

Initially, the application will be launched as a Minimum Viable Product (MVP), including only the essential features needed to attract the target users and meet their basic needs [48]. This approach ensures that the application is practical and useful before spending time on more complex functionality. Following the User Acceptance Testing phase and validation of the platform's viability, the focus will be shifted towards enhancing stability and expanding functionality.

The software analysis starts with the definition of the software requirements followed by the visualization of the system design using several UML diagrams based on the book "Destilované UML" by Martin Fowler [49]. The use case diagram will illustrate system interactions within the application. The class diagram will define system entities and their relationships.

5.2 System Requirements

The following section introduces a list of system requirements outlining what features and properties the application should have. The requirements can be divided into two types:

- **Functional requirements (FR)** provide an overview of how the system should behave.
- **Non-functional requirements (NFR)** specify the system's properties and constraints. NFR may include information regarding system performance, look, and responsiveness.

Each requirement may include additional properties to help better organize the implementation process. The author incorporated a priority attribute for each requirement for better project planning. This approach ensures that the most critical tasks are resolved first while also maintaining a backlog of less essential features that might improve the application in the future. The requirements prioritization was done according to the "MoSCoW" criteria [50].

- **Must have (M)** - requirements with the highest priority. They are critical for the system to work and must be delivered.
- **Should have (S)** - essential requirements, but not vital. Those requirements have high priority and should be included in the solution if possible.
- **Could have (C)** - desirable items that may improve the overall user experience. They are delivered only when there are available resources and time.
- **Want to have (W)** - least critical requirements that can be postponed until the subsequent releases.

Functional Requirements

The functional requirements are based on the results from analyzing existing smart parking applications and the INRIX survey.

User

FR-1 Local Registration (M)
The application will enable users to sign up with an email and a password.

FR-2 Registration with SSO (C)
The application will enable users to sign up with SSO.

FR-3 Local sign in (M)
The application will enable users to sign in with an email and a password.

FR-4 Sign in with SSO (C)
The application will enable users to sign in with SSO.

FR-5 Anonymous Sign in (C)
The application will enable users to sign in without specifying their personal information.

FR-6 Sign out (M)
The application will enable users to sign out.

FR-7 View account information (C)
The application will enable users to view their account information.

FR-8 Update account information (C)
The application will enable users to update their account information. The MVP version should include a password reset.

FR-9 View profile (C)
The application will enable users to view their profiles. In the MVP version of the application, the profile section will include information about the vehicle license plate number and the district of residence in Prague.

FR-10 Update profile information (C)
The application will enable users to update their profile information.

FR-11 User levels (W)
The application will enable users to achieve various levels based on the number of reports created, liked, and disliked.

FR-12 Achievements (W)
The application will enable users to achieve prizes for reports.

Parking Space

FR-13 Display all parking spaces on a map (M)
The application will display all parking spaces on a map.

FR-14 Display all parking spaces as a list (C)
The application will display all parking spaces as a list.

FR-15 Detailed information of a parking space (M)
The application will display detailed information about a selected parking space. Information will include parking ID, zone color, price per hour, total number of parking spots, current number of occupied parking spots, recent reports, pricing, and reviews.

FR-16 Parking reviews (C)
The application will enable users to write reviews on a selected parking space.

FR-17 Search parking spaces (S)
The application will enable users to search for a parking space by street name or parking ID.

FR-18 Parking in a residence district (M)
The application will enable each user to park as a resident. Parking end time will not be known until a user drives away.

FR-19 Parking outside a residence district (M)
The application will enable users to park for a specific time outside their residence district. Parking end time will be known right away.

FR-20 Payment for parking (W)
The application will enable users to pay for parking outside their residence district.

FR-21 Navigation to a parking space using third-party applications (S)
The application will enable users to open directions to a specific parking space using third-party applications such as Apple Maps, Google Maps, and Waze.

FR-22 Navigation to a parking space using in-app navigation (C)
The application will enable users to navigate to a parking space.

FR-23 Current parking relation (M)
The application will enable users to view current parking relation details. The details will include parking ID, zone color, parking street, parking time duration, and a parking location map.

FR-24 History of parking relations (C)
The application will enable users to view a list of previous parking relations.

FR-25 Cancel current parking relation (M)
The application will enable users to cancel their current parking.

FR-26 Save parking spaces to favorites (C)
The application will enable users to save parking spaces to favorites.

FR-27 Suggestion to park (W)
The application will automatically detect if users are driving a car and suggest to park when they stop for a longer period.

FR-28 Automated end of a parking relation (C)
The application will detect when users leave a parking space and automatically end parking.

FR-29 Background tracking (W)
The application will track user location and speed in the background, suggesting parking spaces and ending parking relations when drivers move away.

Map

FR-30 Parking space appearance (M)
The application will show parking space area bounds, zone colors, and price per hour at the current time.

FR-31 Map settings (C)
The application will enable users to configure the map. The MVP version will include a switcher between a "streets" and a "satellite" mode.

FR-32 User current position (S)
The application will display the user's current position on the map.

Parking Report

FR-33 Display relevant parking reports (M)
The application will display relevant reports for a selected parking space. Relevance is determined by the time the report is created. Initially, reports will be grouped by their type, and the user will be able to expand the group to see all reports of a group.

FR-34 Create report (M)
The application will enable users to create reports for a specific parking space. The MVP version will have four types of reports:
Occupancy reports will indicate how hard it is to find a vacant spot on a selected parking space.
Closure reports will indicate that a selected parking space is unavailable for a specified reason. Users may choose a time interval when a parking space will be unavailable.
Charging stations reports will indicate how many vacant charging stations are available in a specific parking space.
"Other" reports will enable users to describe the issue in their own words. This type of report will not be visible in the parking details.

FR-35 Like report (W)
The application will enable users to like other reports. Reports within a group will be sorted accordingly.

FR-36 Dislike report (W)
The application will enable users to dislike other reports. If a report gets a specific number of dislikes, it gets deleted.

Non-functional Requirements

NFR-1 Security (M)
The application will provide confidential information only to authorized users. The application will not store user geolocation in a database nor share it with other users.

NFR-2	Platform support (M) The application will be available for Android and iOS.
NFR-3	Usability (M) The application interface must be easy to use for a wide range of people.
NFR-4	Profanity (W) The application will automatically detect profanity and remove inappropriate content.
NFR-5	Internationalization (C) The application will be available in Czech and English languages.
NFR-6	Archive parking relations (W) The application will archive old parking records every day.
NFR-7	Performance (C) The application must run at least 40 frames per second on all supported devices.
NFR-8	Platform-agnostic Design (C) The design of the mobile application must not look like a native Android or iOS application.

5.3 User Roles

To ensure a great user experience, the application will not require users to authenticate when they open the application for the first time. Instead, each user will be automatically signed in anonymously. If people need to synchronize their progress with other devices, they will be required to create a new account or sign in to an existing one. Below is a list of user roles that will be available in the MVP.

- Anonymous user
  A person who is logged into the system without providing credentials.
- Authenticated user
  A person who is logged into the system.
- Report author
  An anonymous or authenticated user who created a report.

5.4 Use Case Diagram

The current section introduces several use case diagrams based on the system requirements described in section 4.2 and user roles from section 4.3. Use case diagrams aim to visualize relations between actors and depict the use cases of each actor.

5.4.1 Actors

Figure 5.1 demonstrates the main actors in the system: the first three actors from the left represent user roles described in the previous section, and the last actor represents time. The arrow pointing from the "Report Author" to "Authenticated User" symbolizes inheritance. The report author will have all the functionality available to the authenticated user and may have additional features not available to the authenticated user. Actors are colored differently to demonstrate what use cases are specific to each actor. White use cases are common for every actor.

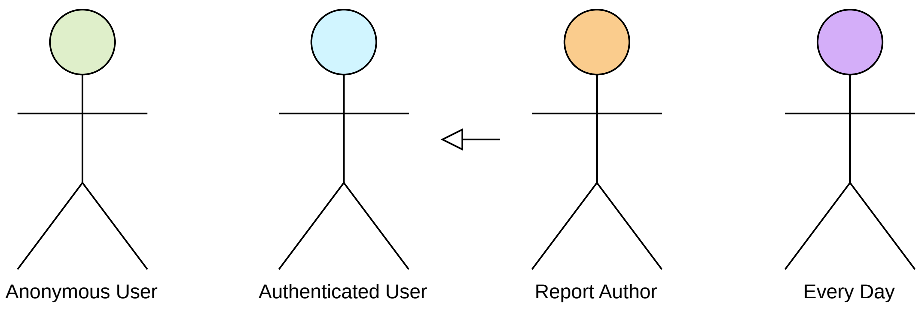


Figure 5.1: Actors UC diagram [author]

5.4.2 Anonymous User

Figure 5.2 shows a use case diagram for the "Anonymous User" role. The "Anonymous User" is the first role a user gets when opening the application for the first time. Anonymous users have access to almost all features available for authenticated users (except for viewing and updating their account information). In addition, anonymous users have access to login and registration screens. The use cases available only for anonymous users (UC1, UC2, UC3, UC4) are colored in green.

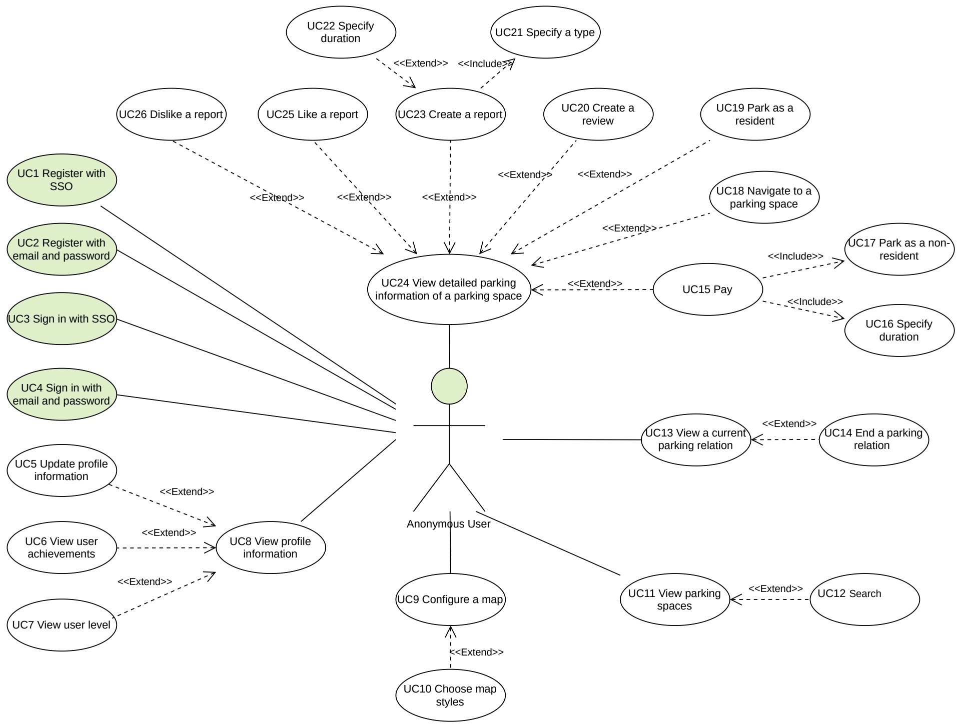


Figure 5.2: Anonymous user UC diagram [author]

5.4.3 Authenticated User

Figure 5.3 illustrates a use case diagram for the "Authenticated user" role. In comparison to anonymous users, authenticated users will not have options for signing in or creating a new account. On the other hand, they will have access to the account information screen. The use cases available only for authenticated users (UC28, UC27) are colored in blue.

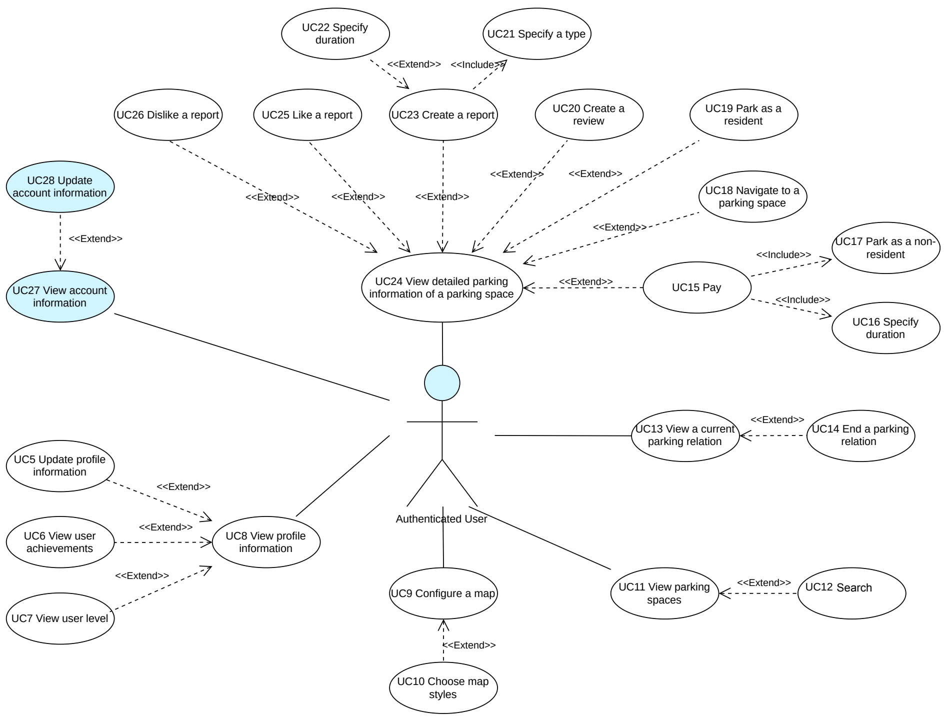


Figure 5.3: Authenticated user UC diagram [author]

5.4.4 Report Author

As shown in figure 5.4, the "Report author" user role inherits all the features available for authenticated users. Besides the base functionality, report authors may update their report information. The use cases available only for report authors (UC29) are colored in orange. The use cases available for authenticated users (UC28, UC27) are colored in blue.

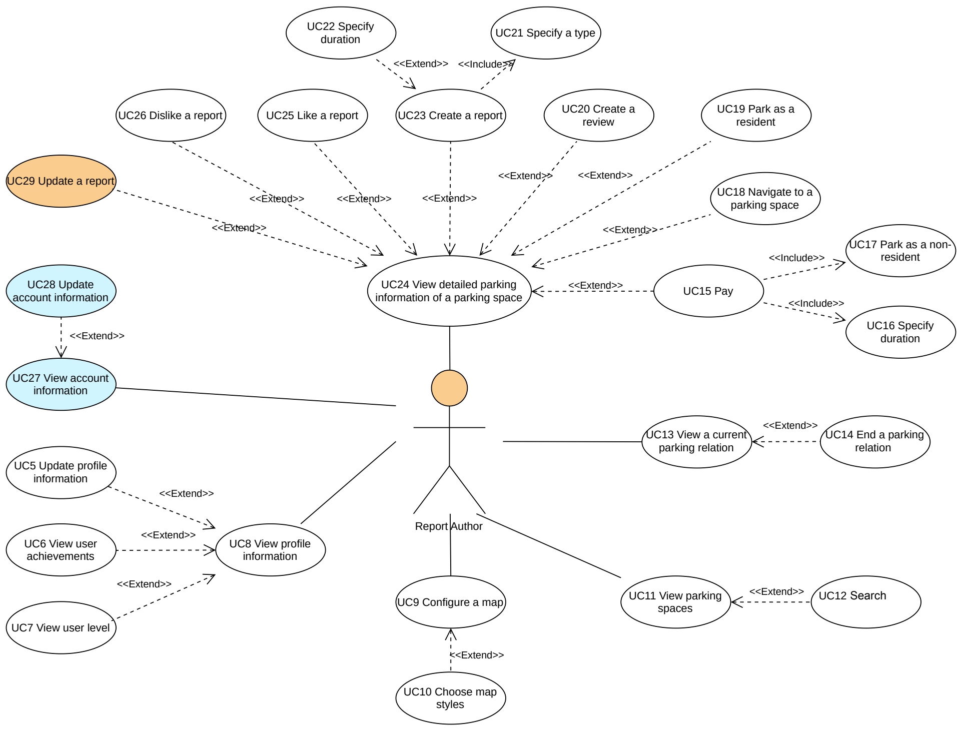


Figure 5.4: Report author UC diagram [author]

5.4.5 Time Actor

The system will archive old parking records daily, as stated in the NFR6. Figure 5.5 demonstrates a single actor connected to the related use case.

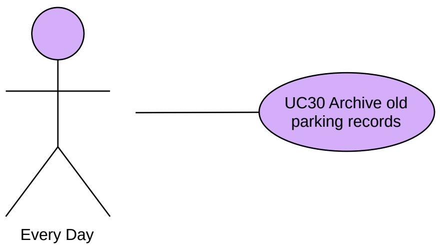


Figure 5.5: Time actor UC diagram [author]

5.5 Class Diagram

Class diagrams visualize entities in the system, their attributes, and relationships. Figure 5.6 shows the class diagram for the smart parking mobile application.

- ParkingSpace
  This class represents a parking space and includes attributes such as longitude (float8), latitude (float8), place (text), zone_id (text), and price per hour (int2). Parking space is located in one of the parking zones represented by attribute zone_id and related enumeration type.

- CurrentParking
  This class captures active or recent parking sessions. Attributes include whether the session is cancelled (bool), the end time of the parking (timestamp), and the total price charged (float4).

- Profile
  Represents a user profile with attributes including email (text), residence district (int2), and a list of licence plates (text[])

- Report classes
  Each report is represented by its own class to ensure that data is properly structured. *ReportOther* includes a description (text) of various reports that do not fit into the other specified categories. *ReportChargingStations* contains a count (int2) of charging stations. *ReportOccupancy* maintains the occupancy level categorized by the OccupancyLevel enumeration. *ReportClosure* reports closures with reasons defined by the *ClosureReason* enumeration and includes attributes for when the space was closed from and to (timestamp).

The relationships between these classes are mostly one-to-many:

- Each *ParkingSpace* can be associated with zero to many *CurrentParking* records.

- *Profiles* can have zero to many *CurrentParking* records linked to them, reflecting multiple parking sessions per user.

- Each report has a reference to its creator and a reference to a parking space that contains a report.

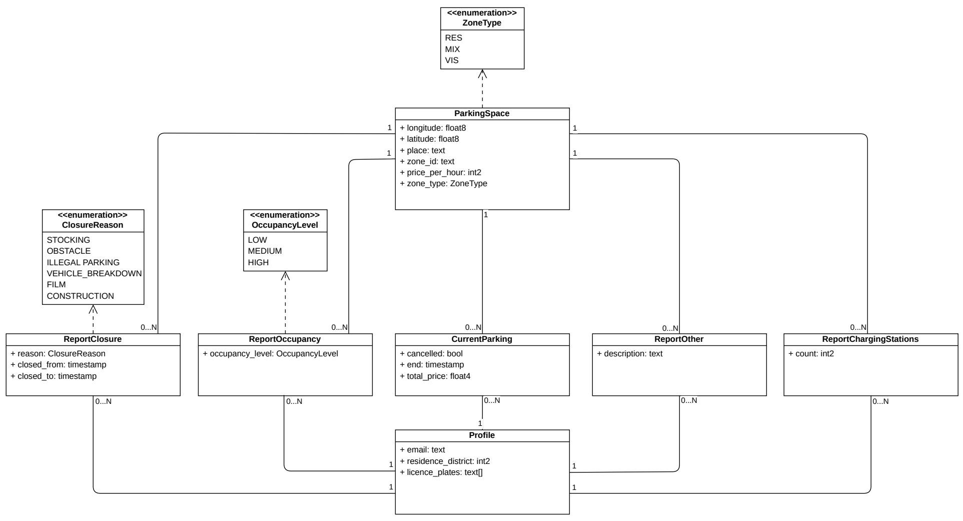


Figure 5.6: Class diagram [author]

5.6 User Interface Design

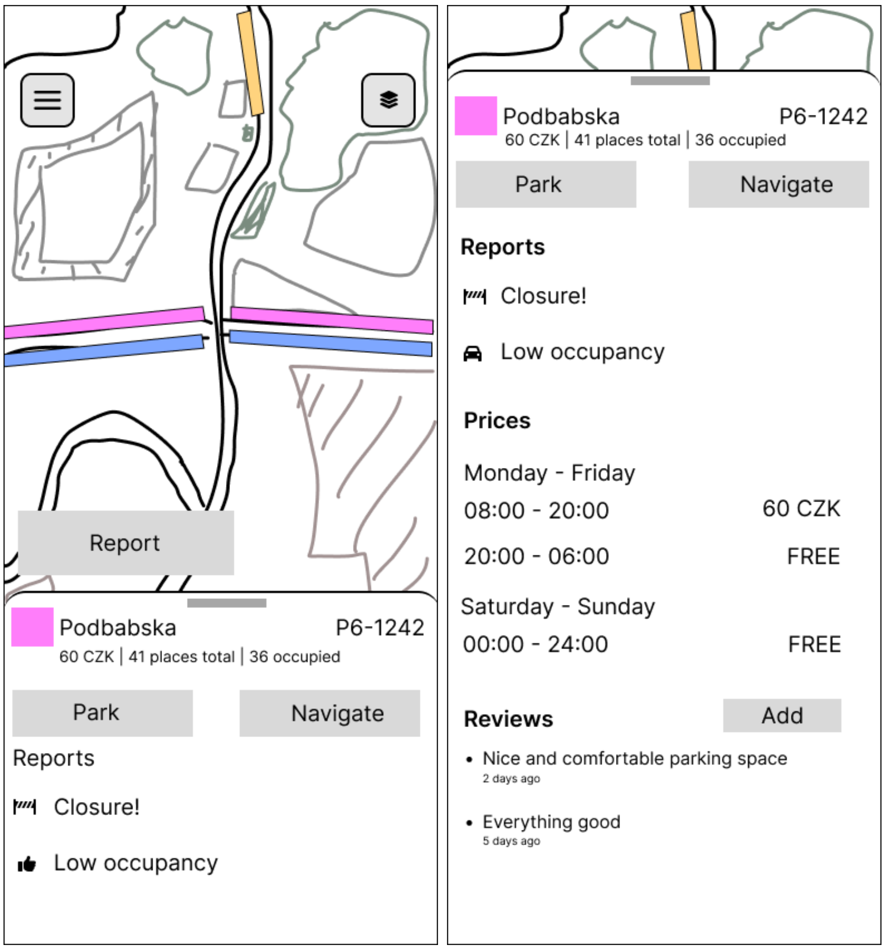
The following section outlines the design process for developing the user interface of the mobile application. The approach involves initial low-fidelity (Lo-Fi) prototyping, followed by usability testing, and the creation of a high-fidelity (Hi-Fi) prototype based on the test findings.

5.6.1 Lo-Fi Prototype

Lo-Fi prototype is an early-stage tool that helps design teams visualize ideas on how the future application might look like. It depicts a base page layout, content arrangement, and potential user interactions without focusing on aesthetics or functionality. Developing a Lo-Fi prototype is crucial for usability testing. By demonstrating a simplified version of the product, teams can gather valuable feedback from the target users and identify potential issues with layout and flow. [51]

The created Lo-Fi prototype covers the core functionalities of the application, including finding the most suitable parking space, searching for a parking space, and creating a report. Figure 5.7 demonstrates parking details in a bottom sheet. Initially, the bottom sheet presents the general information regarding a parking space, action buttons for parking, creating a report, and navigation to a parking space. Additionally, the parking space has closure and occupancy reports at the bottom of the sheet. When the bottom sheet is expanded, users may also observe pricing and reviews.

The whole Lo-Fi prototype for the developed application can be found in the attached CD.



Podbabska P6-1242
60 CZK | 41 places total | 36 occupied

Reports

Closure!

Low occupancy

Prices

Monday - Friday	
08:00 - 20:00	60 CZK
20:00 - 06:00	FREE
Saturday - Sunday	
00:00 - 24:00	FREE

Reviews

- Nice and comfortable parking space
  2 days ago
- Everything good
  5 days ago

Figure 5.7: Lo-Fi prototype [author]

5.6.2 Usability Testing

In total, five people agreed to participate in the testing session for the parking app prototype, employing the Wizard of Oz method. This approach involved the participants interacting with what they believed to be a fully functioning app, while behind the scenes, the moderator (the author) manually controlled the application responses to simulate various functionalities [52]. Before the start, participants were given a questionnaire to give the moderator a better understanding of the individual. Each participant completed a questionnaire designed to provide the moderator with insights into their previous experience with smart parking applications. The questions were centered around their usage habits, the challenges they frequently encounter, and the features they value the most in such apps.

The main objective of the usability testing was to evaluate the application functionality, ease of use, and overall user experience across various realistic scenarios. To achieve this, each participant was asked to perform tasks that mirror typical actions while using the app. These tasks were structured to assess how intuitively the app meets the needs of its users and how effectively it resolves common parking issues. The tasks included:

1. *"You are planning to visit a major tourist attraction in the heart of Prague during a busy weekend. Use the prototype to find the nearest available parking space and park there."*

2. *"Upon arriving at a parking lot, you find a parked car blocking multiple parking spaces. Report this issue using the app's reporting feature."*

3. *"You are coming to a party in a location you have never been to and want to park your car. Use a search feature on the prototype to find a parking space near the party place."*

4. *"You live in the busy downtown, and your house is being renovated. The parking space near your house is unavailable due to the construction. Use the reporting feature to notify other users about parking space closure."*

5. *"You have parked in a parking space and noticed several other vacant parking spots. Report the parking occupancy using the prototype."*

After completing these tasks, the participants were interviewed to collect qualitative feedback on their experiences.

Results

During the post-test interviews, the participants provided mostly positive feedback. They appreciated the overall concept, particularly the ability to notify others about parking space occupancy and any issues encountered at a parking space. The navigation between screens felt intuitive mainly because of prior experience with other smart parking applications.

During the testing session, participants encountered various challenges that highlighted areas for improvement.

Map Usability Issues

- **Small Parking Space Areas**
  Two participants noted that the parking spaces on the map were too small, making them difficult to select accurately, especially while walking or driving. This could lead to user frustration and errors in selecting the desired parking space, detracting from the app's overall usability.

- **Map Navigation and Location Tracking**
  Participants asked if there was an option to move the map to the user's current location automatically. Unfortunately, this feature was not available at that moment. The absence of such features could lower the application effectiveness in navigation scenarios where real-time tracking is essential.

Parking Related Issues

- **Finding the Cheapest Parking**
  The participants found it time-consuming to locate the cheapest parking option as it required selecting each parking space individually and expanding the bottom sheet to view pricing details.

- **Real Photos of a Parking Space**
  One participant expressed a desire to see photos of the parking spaces. Images could enhance user trust and decision-making, providing a clearer expectation of the parking environment before arrival.

- **Accessibility of Controls**
  The control buttons were displayed too far from the thumb's reach when the information sheet was expanded, making it difficult to press on bigger mobile devices. This ergonomic issue could affect the app's accessibility, particularly for users operating the device with one hand.

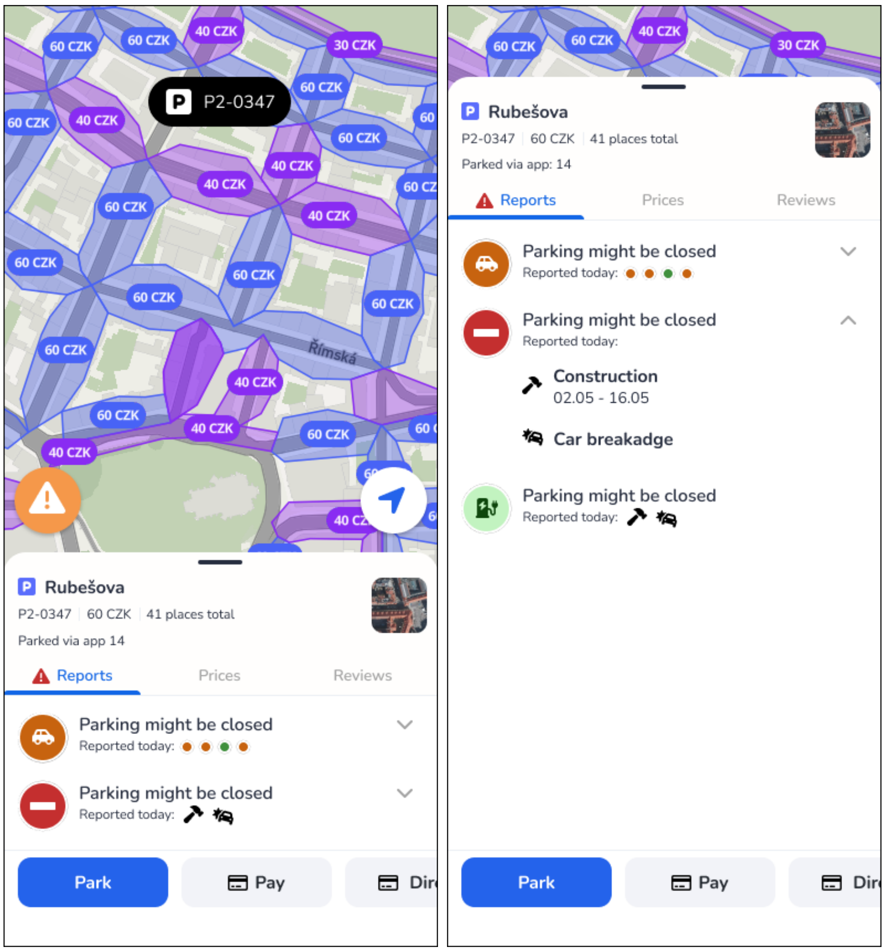
- **Missing Current Parking Information**
  After parking through the application, a participant did not find a way to see the current parking details.

- **Inconvenient Parking Occupancy Selection**
  The Lo-Fi prototype required the exact number of available vacant parking spots. Some parking spaces contain tens of parking spots, raising concerns about potential inconvenience and the possibility of displaying inaccurate information regarding space availability.

5.6.3 Hi-Fi Prototype

Based on insights from usability testing and several design iterations, a high-fidelity (Hi-Fi) prototype was developed. This Hi-Fi prototype showcases the close-to-final polished look of the application, with refined visuals and functionality that closely mimic the end product [53]. Figure 5.8 displays the screen with parking space details. Compared to the Lo-Fi prototype, the parking space information is now divided into multiple sections: reports, prices, and reviews. This change enables users to quickly find the information they need. The most frequently used actions are relocated to the scrollable bottom sheet footer. In contrast, secondary actions, such as creating a report and moving to the current position, are placed above the sheet. The Hi-Fi prototype was then presented to participants from usability testing for the additional validation.

The whole high-fidelity prototype of the smart parking application can be observed in the attached CD.



Rubešova
P2-0347 | 60 CZK | 41 places total
Parked via app 14

Parking might be closed
Reported today: ● ● ● ●

Parking might be closed
Reported today:

Rubešova
P2-0347 | 60 CZK | 41 places total
Parked via app: 14

Parking might be closed
Reported today: ● ● ● ●

Parking might be closed
Reported today:

Construction
02.05 - 16.05

Car breakadge

Parking might be closed
Reported today:

Figure 5.8: Hi-Fi prototype [author]

Chapter 6

Technology Stack Specification

This chapter focuses on selecting technologies to be used during the development of the mobile application. It examines various libraries and services that could be useful for both backend and frontend implementation.

6.1 Backend

The following section is devoted to choosing backend technologies that will be used for the project. The two possible options for integrating the backend into the project are implementing a custom backend service or using an existing Backend as a Service (BaaS) from third-party vendors.

6.1.1 Custom Backend Service

Implementation of a custom backend service provides big flexibility and control. This approach involves setting up servers, databases, and APIs designed specifically for the project's needs. Before implementing the custom backend service, it is essential to consider several pros and cons of this approach.

Pros

- **Full control of the Server Infrastructure**
  Significant performance gains come when the written code is optimized for the hardware it runs on. Having complete control over the backend environment allows teams to optimize every layer of the stack, including hardware specifications, operating systems, and server configurations. Such tailored optimizations ensure that the system operates at peak efficiency, reduces latency, and handles high-load scenarios better than a generic setup might.

- **Database for the Project Needs**
  Choosing a suitable database is critical for project success, as different types cater to specific needs. Key-value stores like Redis are excellent for quick data retrieval and storage and are suitable for caching and sessions. Wide-column databases such as Cassandra or ScyllaDB handle

large volumes of data efficiently, ideal for messaging apps. Document-based databases like MongoDB are designed for managing unstructured data and are useful in content management systems. Graph databases are utilized in networking applications, while relational databases like PostgreSQL ensure data integrity for transaction-heavy applications like financial systems. Selecting an appropriate database enhances project performance, scalability, and reliability. A custom backend service also allows teams to modify, add, or remove databases as a new requirement comes in.

- **Security**
  Some projects require advanced security mechanisms that third-party vendors might not be able to handle. For example, bank applications may require data encryption and an advanced authentication process for accessing confidential data.

- **Long-term Cost-Efficiency**
  Setting up a custom backend requires significant investment in time and money, and it can ultimately reduce operational expenses. By tailoring the system to the project's specific needs, teams can use resources more efficiently, thus lowering the costs associated with server resources and external services over time.

Cons

- **Time-Consuming**
  In the early stages, building all the server infrastructure will require a significant amount of time. Not having experience in some areas may slow down the implementation even more.

- **Higher Initial Costs**
  To handle larger volumes of data, projects require investment in appropriate hardware, software licenses, development resources, and initial testing.

- **Resource Intensive**
  A custom backend requires constant attention to ensure its smooth operation. This includes regular updates, security patches, and potentially troubleshooting and resolving hardware issues.

6.1.2 Backend as a Service (BaaS)

Backend as a Service solutions try to mitigate disadvantages of implementing a custom backend. Opting for BaaS provides a streamlined path to backend development, allowing teams to focus on frontend development and user experience while offloading backend complexities to a third-party service. BaaS offers a set of ready-to-use backend functionalities, including database management, server-side logic, and cloud storage, accelerating the development process and reducing the need for backend expertise.

Multiple items must be considered before integrating a BaaS.

Pros

- **Development Speed**
  BaaS significantly speeds up the development and deployment process, enabling faster market entry. This benefits startups that need to validate product-market fit before spending much money on infrastructure and development.

- **Opt-in Scalability**
  BaaS platforms are designed to automatically adjust their processes to handle increased loads, making them particularly well-suited for projects with variable demands. This feature is especially beneficial for startups and companies in the early stages of development, as it allows them to save money initially by not paying for resources they do not yet use.

- **Cost-Effective Maintanace**
  The service provider maintains all the processes within the backend, reducing the internal workload and associated costs.

Cons

- **Limited Control**
  In contrast to custom backend solutions, BaaS solutions offer limited customization regarding hardware specifications and database selection. BaaS provides services for a broad range of applications rather than specific use cases; therefore, it might not be suitable for some projects requiring unique features.

- **Vendor Lock-In**
  While using a Backend as a Service (BaaS) platform, transitioning away from it can often be impractical due to high associated costs. These costs may include designing a custom backend solution from scratch and refactoring the existing frontend codebase to accommodate the new backend architecture. This process involves significant financial investment and requires extensive time and resource allocation. The complexity of adapting the frontend to interface smoothly with a different backend system can lead to substantial development challenges, potentially disrupting service continuity and impacting user experience during the transition period.

- **Potential Cost Increases**
  While initially cost-effective, BaaS can become expensive as the project scales, mainly if the pricing model depends on resource usage or user count.

Summary

Given the objectives of the Master thesis to test the feasibility of an idea rapidly, the choice to use Backend as a Service (BaaS) is the most appropriate. BaaS provides a swift and cost-effective route to deployment, which aligns with the thesis focus on validating the concept without significant upfront investment. Moreover, the time savings from using BaaS are crucial in keeping the project timeline focused on the core objectives rather than backend complexities.

If the application proves to be feasible, a transition to a custom backend solution is needed. Communication between the frontend and backend should be carefully encapsulated to mitigate the impact on the frontend codebase during this transition. This means designing the system so that backend-specific details are abstracted away from the frontend components. By doing so, changes to the backend infrastructure can be implemented with minimal adjustments required on the frontend, thereby reducing the complexity and potential disruptions during the transition.

6.1.3 Supabase

Based on the author's prior positive experience with Supabase, it was decided to integrate it as the Backend-as-a-Service for the smart parking application. Supabase is an extensive set of open-source technologies that build up a ready-to-use backend server. By the time of writing the thesis, Supabase had more than 65 thousand stars and nearly 200 issues on GitHub. The main features include:

- PostgreSQL database is a highly scalable relational database with extensions (such as PostGIS for storing geospatial data), row level security (RLS) for multi-tenant applications, and performance tuning [54].
- Supabase Studio is a web-based dashboard platform for managing the backend service.
- GoTrue is a ready-to-use authentication service with the support of many SSO providers, including Apple, Google, and GitHub.
- PostgREST is a tool for auto-generating APIs directly from the PostgreSQL database.
- Realtime subscribtions
- File storage
- Database and Edge functions
- Language-specific SDK

Despite the features, Supabase offers a free plan for starting projects with the following conditions [55]:

- Unlimited API requests
- 50,000 monthly active users
- 500 MB database space
- 2 Core shared CPU, 1 GB RAM
- 5 GB bandwidth
- 1 GB file storage

Such limitations are acceptable for the current project as it is expected to handle a small portion of requests and store small amounts of data.

6.2 Frontend

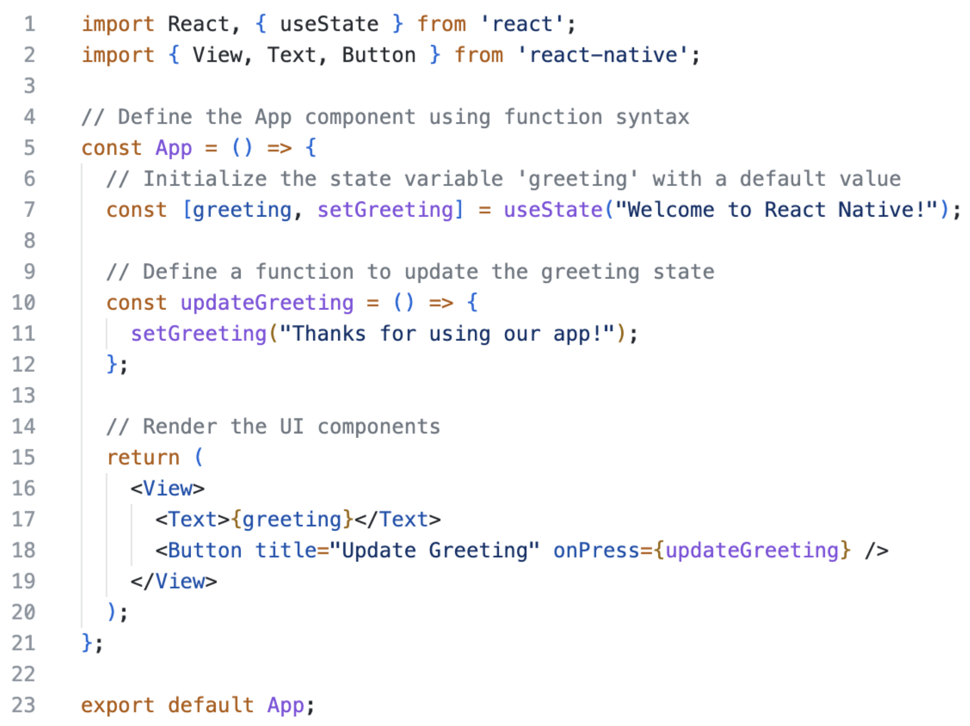
According to the non-functional requirements stated in Chapter 5, the mobile application must be available for Android and iOS. There are two ways of fulfilling this requirement: native development using Kotlin for Android and Swift or Objective-C for iOS or cross-platform development using Cordova, Ioniq, Flutter, or React Native.

6.2.1 React Native

The author's prior experience with React in web development has led to the choice of React Native for this project. This framework extends the capabilities of React, enabling the development of mobile applications that are nearly indistinguishable from native apps in terms of performance and user experience. Such performance is achieved by providing platform-agnostic native components like *View*, *Text*, and *Button* that map directly to the platform's native UI building blocks [56].

The syntax of React Native is quite similar to React's, utilizing JSX to build application views. JSX is a syntax extension for JavaScript that allows developers to write UI components that resemble HTML in structure within their JavaScript code. This blend of JavaScript and HTML-like elements enables a more intuitive design process for building application interfaces. The familiarity of this syntax to those experienced with React simplifies the transition to mobile development, as the same principles of component-based architecture are applied and the same design patterns are used. The example of JSX syntax in React Native is provided in figure 6.1.

There are, however, some differences between the two libraries. The most noticeable is styling. React Native renders native components that look different on Android and iOS. Hence, styles must be applied to unify the look of the application. In contrast to React, React Native does not support CSS. Instead, it uses JavaScript bindings to set fonts, colors, spacing, and other properties. This way ensures all styles are set and displayed correctly on both platforms.



```
import React, { useState } from 'react';
import { View, Text, Button } from 'react-native';

// Define the App component using function syntax
const App = () => {
  // Initialize the state variable 'greeting' with a default value
  const [greeting, setGreeting] = useState("Welcome to React Native!");

  // Define a function to update the greeting state
  const updateGreeting = () => {
    setGreeting("Thanks for using our app!");
  };

  // Render the UI components
  return (
    <View>
      <Text>{greeting}</Text>
      <Button title="Update Greeting" onPress={updateGreeting} />
    </View>
  );
};

export default App;
```

Figure 6.1: React Native JSX Example [author]

6.2.2 TypeScript

JavaScript is a dynamically typed language, which means that it infers the types of variables at runtime. This characteristic offers certain advantages, such as saving time in writing explicit type definitions. However, it can also lead to unpredictable behavior or runtime exceptions due to incorrect type assignments. Hence, the project will leverage the capabilities of TypeScript. TypeScript is a programming language developed on top of JavaScript that introduces static typing and optional type annotations. By integrating TypeScript, the codebase will have the flexibility of JavaScript while significantly enhancing code safety and predictability, reducing the likelihood of type-related errors in the application. Moreover, TypeScript brings zero overhead to the final code as all the types are deleted during compilation.

6.2.3 Expo

Expo[1] is a robust framework and platform that enhances React Native development by providing a wide range of built-in functionalities and streamlined processes.

The core benefits include:

[1]Available on: https://docs.expo.dev/get-started/introduction/

- Variety of modules and APIs that extend the capabilities of React Native applications. These modules cover a range of functionalities, from accessing the device's hardware, like cameras and sensors, to integrating third-party services. This reduces the need for custom native development, streamlining the development process and allowing developers to implement complex functionalities easily.
- In case a mobile application requires a custom logic, developers can utilize Expo Modules. The Expo modules package enables the integration of custom native code into the React Native codebase.
- The Development stage often requires testing the developed mobile application on real devices. For that, developers may utilize either ExpoGo for simple applications or install development builds when full control over the native runtime is required.
- Expo also simplifies releasing the final application to Google Play and AppStore.

6.2.4 Summary

Given the prior experience with React, the author opted for React Native to create apps that emulate native performance and user experience while maintaining a single codebase for Android and iOS. The project will utilize JSX for its HTML-like syntax within JavaScript to ease development. The TypeScript programming language will be used throughout the project to ensure the type safety. Finally, the project will utilize Expo for its robust set of packages, real-device testing, and custom native code integration.

6.3 Client-Server Communication

As mentioned earlier, the project will use Supabase as a backend service. Supabase provides the official JavaScript SDK with typed annotations. The functions within the SDK help connect a Supabase with the frontend and perform database operations. Figure 6.2 demonstrates a simple Supabase query to retrieve first 10 rows from the profiles table.

```
let { data: profiles, error } = await supabase
  .from('profiles')
  .select('*')
  .range(0, 9)
```

Figure 6.2: Supabase Query Example [author]

Developers may utilize database functions when a project requires more complex queries or mutations. Figure 6.3 shows how to call a database function using the Supabase JavaScript client.

```
const { data, error } = await supabase.rpc("db_function_name", {
  param1: value1,
  param2: value2,
});
```

Figure 6.3: Supabase database function example [author]

6.4 Deployment

The deployment process for the mobile application involves utilizing Expo Application Services (EAS) to streamline the publication on Google Play and the App Store. Expo facilitates this by providing a cloud-based tool that generates application bundles. Once these bundles are created, they are submitted to the respective app stores for review. If the application meets all the necessary criteria and passes the verification process successfully, it is then published, making it available to users on both Android and iOS platforms. Figure 6.4 illustrates the deployment diagram after the application was published to the app stores. Mobile devices communicate with the Supabase Cloud through the API Gateway provided by Kong. Sequentially, requests are being redirected to the respective service. Most services communicate with the PostgreSQL database, and if the request is related to the file storage, it is redirected to Amazon Web Services (AWS) S3 storage provider. The deployment diagram is based on the Supabase architecture[2].

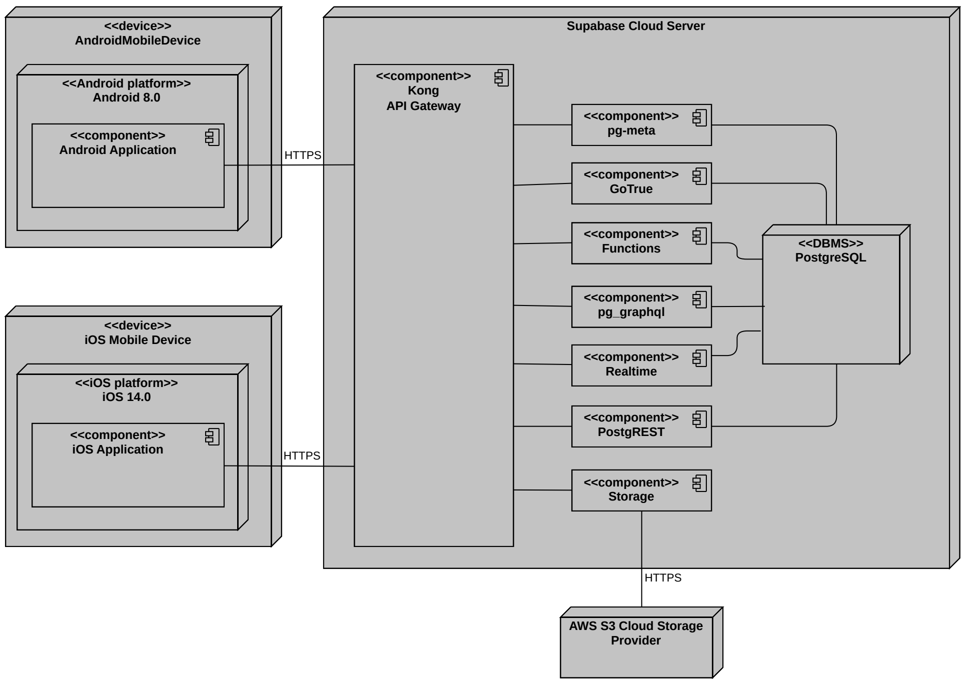


Figure 6.4: Deployment diagram [author]

[2]Available on: https://supabase.com/docs/guides/getting-started/architecture

6.5 Chapter Summary

This chapter provided an analysis of technologies used during the project implementation. Supabase will power the mobile application as the backend, providing a JavaScript SDK for client-server communication. The frontend will be written in TypeScript using the React Native framework. This choice ensures cross-platform compatibility. Finally, the Expo framework will accelerate the development and release processes. When the application is ready, it might be deployed to Google Play and AppStore to make it publicly available.

Chapter 7
Development Process

This chapter outlines the implementation steps of the proposed application.

7.1 Data Preparation

The mobile application must access comprehensive data on Prague City's infrastructure to meet the system requirements. This data should cover a wide range of details, including geolocation, pricing lists, zone identifiers and colors, the total number of parking spaces, real photos, the number of charging stations, locations of parkometers, and other relevant information. Fortunately, Prague operates an official geoportal that houses nearly all the data required for the smart parking application. This resource is invaluable as it provides accurate and up-to-date information that can be directly integrated into the app, ensuring that users receive reliable and current data on parking options within the city. Using the geoportal the author got access to the following set of data:

- **Payment machines**
  Data will be used for the visual representation on the map. GeoJSON features were reduced to only contain geolocation.
  ID: 150f549136904a429c26ed140f1184f8_0

- **Charging stations for electrocars**
  Data will be used for the visual representation on the map as well as parking reports.
  ID: d5931c5868f94961b8da43d26998d752_0

- **Paid parking zones**
  Data will be used for the visual representation on the map, displaying parking details and parking reports.
  ID: 1f31afc674ac4c1496a9a2da7d96e588_0

- **Public transport lines**
  Data will be used to detect whether a user drives a car or uses public transport.
  ID: e2e6a68b508049119978795aa44d18f5_0

The data are available on: *https://geoportalpraha.cz/data-a-sluzby/<ID>*

7.2 Map Setup

Considering the requirement to implement in-app navigation, Mapbox was chosen as the map provider. Mapbox offers a versatile range of mapping services, including static map images, interactive maps, and turn-by-turn navigation. Besides, Mapbox provides a great developer experience by allowing developers to configure map styles and display data in a dedicated web application - Mapbox Studio.

The map is the central component of the application, necessitating an appealing color scheme that not only enhances the visual appeal but also provides clear contrast with parking areas and other user interface elements for optimal user comfort. To achieve this, I designed a custom color scheme that fulfills these requirements and distinctly sets the application apart from Apple Maps, Waze, and Google Maps. Additionally, the chosen color scheme plays a strategic role in branding, as distinctive colors help identify and differentiate the product, making it more recognizable to users. Figure 7.1 illustrates the map used in the application. Colors of parking spaces are configured dynamically based on the GeoJSON feature property. Residential zones are colored in blue, mixed - in purple, and visitor - in orange. Usability testing also discovered that searching for the cheapest parking space may take much time. This issue is mitigated by displaying parking space prices dynamically according to the current hour.

In the context of the developed application, the map can be seen as an additional database. When loaded, users immediately get access to all the parking space information without needing to send requests to the server. Instant feedback enhances user experience and productivity.

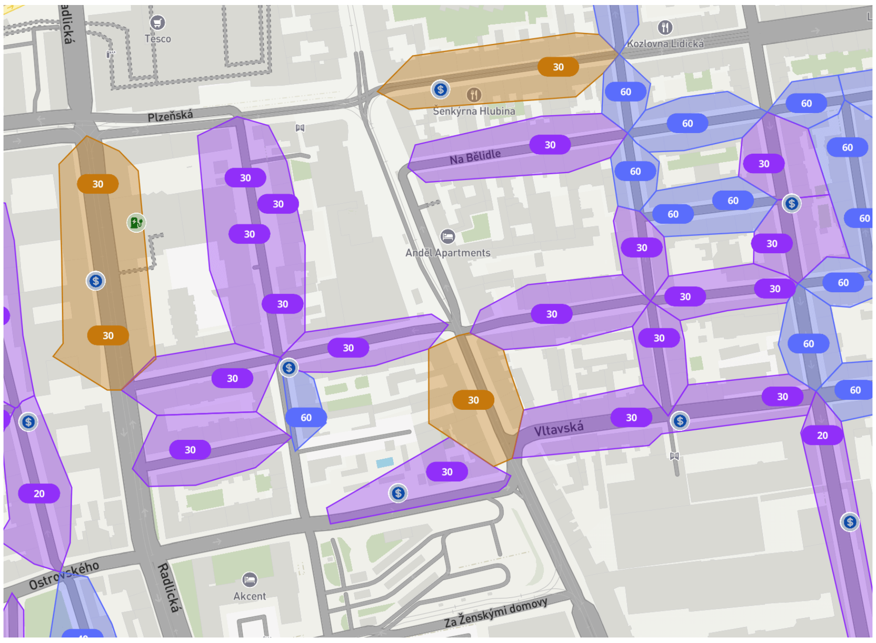


Figure 7.1: Configured Mapbox map [author]

7.3 Database Setup

The next step in the project implementation is dedicated to the Supabase database setup. The process involves creating several tables, functions, triggers, and enabling authentication.

7.3.1 User Profiles

Initially, all user information in Supabase is stored within the *users* table under the *auth* schema. The MVP version of the application requires storing licence plates and residence districts of users. The *profiles* table was created to accommodate this information. Figure 7.2 demonstrates the SQL script and the process for creating this table. Firstly, it creates the table itself. Then, it enables row-level security (RLS) to ensure that data can be viewed or modified only by authorized users. Finally, it creates a function to insert a new row into the table. The function is triggered once a new user is created.

```sql
CREATE TABLE if not exists
  public.profiles (
    id uuid not null references auth.users on delete cascade,
    email text,
    licence_plate text,
    residence_district text,
    primary key (id)
  );

ALTER TABLE public.profiles ENABLE ROW LEVEL SECURITY;

CREATE POLICY "Can only view own profile data." ON public.profiles FOR
SELECT
  USING (auth.uid () = id);

CREATE POLICY "Can only update own profile data." ON public.profiles
FOR UPDATE
  USING (auth.uid () = id);

CREATE
OR REPLACE FUNCTION public.create_profile () RETURNS TRIGGER AS $$ BEGIN
INSERT INTO public.profiles (id, email)
VALUES
  (
    NEW.id,
    NEW.email
  );
RETURN NEW;
END;
$$ LANGUAGE plpgsql SECURITY DEFINER;

CREATE TRIGGER create_profile_trigger
AFTER
  INSERT ON auth.users FOR EACH ROW WHEN (
    NEW.raw_user_meta_data IS NOT NULL
  ) EXECUTE FUNCTION public.create_profile();
```

Figure 7.2: Creation of user profiles [author]

7.3.2 Parking Spaces

The following subsection details the configuration related to the parking spaces functionality. It outlines the specific tables that need to be created and provides examples of database functions designed for operating with these tables.

Table definitions

Setting up database tables related to parking spaces required the creation of several tables.

- parking_spaces
  A read-only table that stores general information about parking spaces. To ensure a fast lookup, the index is created on the id column.
- current_parking
  Stores information about parking records.
- public_transport_routes
  Stores geometry data of public transport routes.
- reports_closure
  Stores records of closure parking reports.
- reports_occupancy
  Stores records of occupancy parking reports.
- reports_charging_stations
  Stores records of parking reports with a count of available charging stations.
- reports_other
  Stores records with parking reports of type "other".

Database Functions

To accommodate users who may wish to update their parking space reports after noticing an error in their initial submission, the author included a set of database functions designed to manage such updates effectively. These functions first verify the existence of the record and check whether it falls within a specified time interval since its creation. If the record exists and is not older than this defined interval, the function updates the existing report with the revised values, avoiding the creation of a duplicate entry. A new record is created to capture the updated information if no corresponding record is found or the existing record is beyond the allowed time interval. An example of such a function is demonstrated in Figure 7.3.

```sql
create
or replace function update_or_create_report_closure (
  in p_parking_id varchar,
  in p_reason REASON,
  in p_interval interval,
  in p_from timestamp,
  in p_to timestamp,
  out is_updated boolean
) returns boolean as $$
BEGIN
  UPDATE reports_closure AS ro
  SET reason = p_reason,
      closed_from = p_from,
      closed_to = p_to
  WHERE ro.parking_id = p_parking_id
    AND ro.created_by = auth.uid()
    AND ro.created_at >= NOW() - p_interval;

  IF FOUND THEN
    is_updated := TRUE;
  ELSE
    is_updated := FALSE;
    INSERT INTO reports_closure (reason, parking_id, created_by,
closed_from, closed_to)
    VALUES (p_reason, p_parking_id, auth.uid(), p_from, p_to);
  END IF;
END;
$$ language plpgsql;
```

Figure 7.3: Creation of a closure report [author]

Figure 7.4 represents a function that retrieves nearby public transport lines. This functionality is a part of heuristics for the detection of transportation modes. Based on the provided longitude and latitude, the function returns all transport lines within the specified distance. Additionally, it orders results by the distance.

```sql
CREATE OR REPLACE FUNCTION public.find_nearby_lines(lat double
precision, long double precision, max_distance double precision)
RETURNS TABLE(id integer, dist_meters double precision) AS
$$
BEGIN
    RETURN QUERY
    SELECT l.id,
           ST_Distance(l.coordinates::geography,
ST_SetSRID(ST_Point(long, lat), 4326)::geography) AS dist_meters
    FROM public.public_transport_routes l
    WHERE ST_DWithin(l.coordinates::geography, ST_SetSRID(ST_Point(long,
lat), 4326)::geography, max_distance)
    ORDER BY dist_meters;
END;
$$ LANGUAGE plpgsql;
```

Figure 7.4: Find nearby transport lines query [author]

7.3.3 Authentication

As per system requirements, application users are initially signed in anonymously and must be able to authenticate later using email and password or using SSO providers. For the scope of the MVP project, the author decided to configure only Google provider. Enabling email and password authentication and anonymous sign-ins was a straightforward process requiring turning on a switch in the Supabase admin panel. Setting up authentication via the Google provider was slightly more involved; it required the creation of a new project on the Google Cloud Platform. Once the project was initialized, the author obtained the necessary Client ID and Client Secret from the Google Cloud Console. These credentials were then entered into the Supabase admin panel to complete the integration.

7.4 Frontend Implementation

The following section provides an overview of functionalities that have been implemented into the mobile application.

7.4.1 Navigation

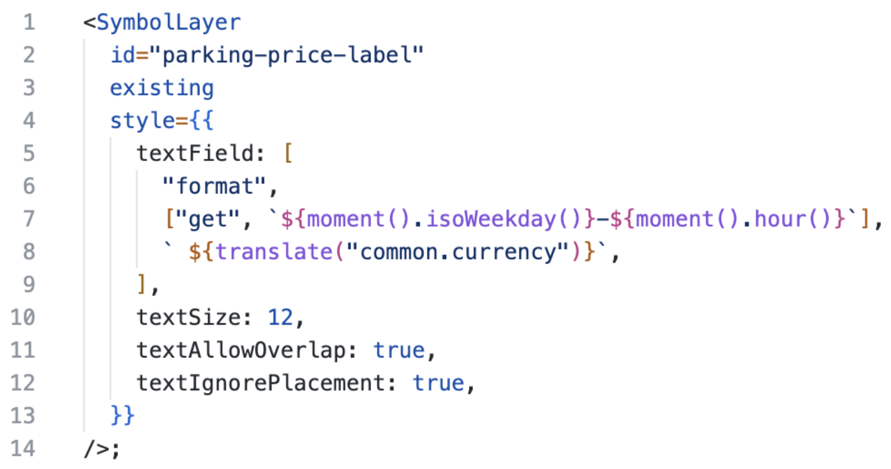
Expo projects have a built-in solution for navigating between screens, extending the React Native Navigation package with file-based navigation. This foundational functionality was further customized to accommodate specific app requirements. The two additional navigators were created using the bottom sheet and top tabs. Bottom sheets are used to display parking space details and report creation forms. Top tabs are utilized for multi-step forms, providing smooth transition animations without writing much code. Both navigators reduce application state maintenance and provide great flexibility.

Used packages: `expo-router`, `@react-navigation/native`, `@th3rdwave/react-navigation-bottom-sheet`.

7.4.2 Map Integration

Mapbox does not officially support direct integration of their maps into a React Native codebase. Fortunately, its community maintains an open-source module that fully satisfies the project's needs. Using its components, the application renders a map with adjusted existing layers to display parking space prices and highlight selected parking spaces dynamically. The code in Figure 7.5 demonstrates using Mapbox expression to query for a relevant price in the GeoJSON feature and append a currency to the string.

Used packages: `@rnmapbox/maps`.



```
<SymbolLayer
  id="parking-price-label"
  existing
  style={{
    textField: [
      "format",
      ["get", `${moment().isoWeekday()}-${moment().hour()}`],
      ` ${translate("common.currency")}`,
    ],
    textSize: 12,
    textAllowOverlap: true,
    textIgnorePlacement: true,
  }}
/>;
```

Figure 7.5: Dynamic parking space price [author]

7.4.3 Authentication

The authentication process within the application is managed using the JavaScript SDK from Supabase, which offers a suite of functions designed to handle user authentication states. When a user successfully signs in, the SDK facilitates the creation of a session that encapsulates the user's authentication details. This session is then securely saved to the device's internal memory. This mechanism ensures that the user remains authenticated on subsequent app uses without needing to sign in repeatedly. When a user signs out, the session is deleted from the memory.

Used packages: `@supabase/supabase-js` , `@react-native-async-storage/async-storage`

7.4.4 Parking Space Details

Information regarding parking spaces is retrieved from the database and GeoJSON Feature properties. The GeoJSON features carry static data such as parking ID, zone colors, and geolocation. Dynamic, user-generated content such as current parking space occupancy and reports is stored in the database. To optimize performance and reduce server load, fetched results are cached. The cache is invalidated when the relevant data is mutated or when the cache is old enough. Websockets are utilized to display real-time updates on the number of available parking spots. This setup allows a direct connection between the user's device and the server whenever a parking space is selected. If there are any changes to the data, the user interface is updated to reflect the new information.

Used packages: `@supabase/supabase-js` , `@tanstack/react-query`

7.4.5 Internationalization

Applications must support internationalization (i18n) to be accessible to a broader range of users. This feature is integrated into the application on the system level. The application interface will be translated into Czech if the system language is set to Czech. Otherwise, the application interface will be in English.

Used packages: `i18n-js`, `expo-localization`

7.4.6 Navigation to a Parking Space

The application incorporates navigation functionalities that integrate both external and internal navigation services to provide accurate directions to designated parking spaces. For external navigation, the application employs deep links to well-established navigation platforms such as Google Maps, Waze, and Apple Maps. These deep links provide a transition from the application to the user's preferred navigation app, automatically opening turn-by-turn navigation with the selected destination.

Simultaneously, the application features an internal navigation system developed using custom native code in Swift. This in-app navigation capability not only provides directions but also may display additional information about parking occupancy and reports, enriching the user experience with valuable real-time data. Besides, in-app navigation enables the system to run other services, including the automatic ending of parking relations or sending suggestions to park when users stop driving.

Used packages: `expo-modules-core`, `mapbox/mapbox-navigation-ios`

7.4.7 Transport Mode Detection

The codebase includes a custom React hook that detects whether a user drives a car. The functionality is based on the heuristic that there is a high probability that users will be at least on one road without public transport lines while driving a car. If users move faster than 30 km/h, the application queries a server whether there is a public transport line within 10 meters every 6 minutes. If there are no public transport lines, the application will assume a person drives a car; otherwise, when the application finds six public transport lines in a row, it will assume that a person uses public transport. Such functionality heavily relies on GPS data due to its higher precision compared to other sensors like accelerometers. Therefore, it may consume a lot of energy. Future releases of the application will focus on function optimizations, including the transport mode detection heuristics. This algorithm enables the application to enhance user experience further by automatically detecting the start and end of parking relation.

Used packages: `expo-location`

7.5 Summary

The chapter described the development process of the smart parking mobile application, detailing the backend and the frontend parts. The chosen technologies significantly sped up the implementation process and enabled the author to deliver the application for both Android and iOS platforms.

The application source code is available in the attached CD. Additionally, the source code contains a guide with instructions for setting up the development environment and what needs to be configured beforehand.

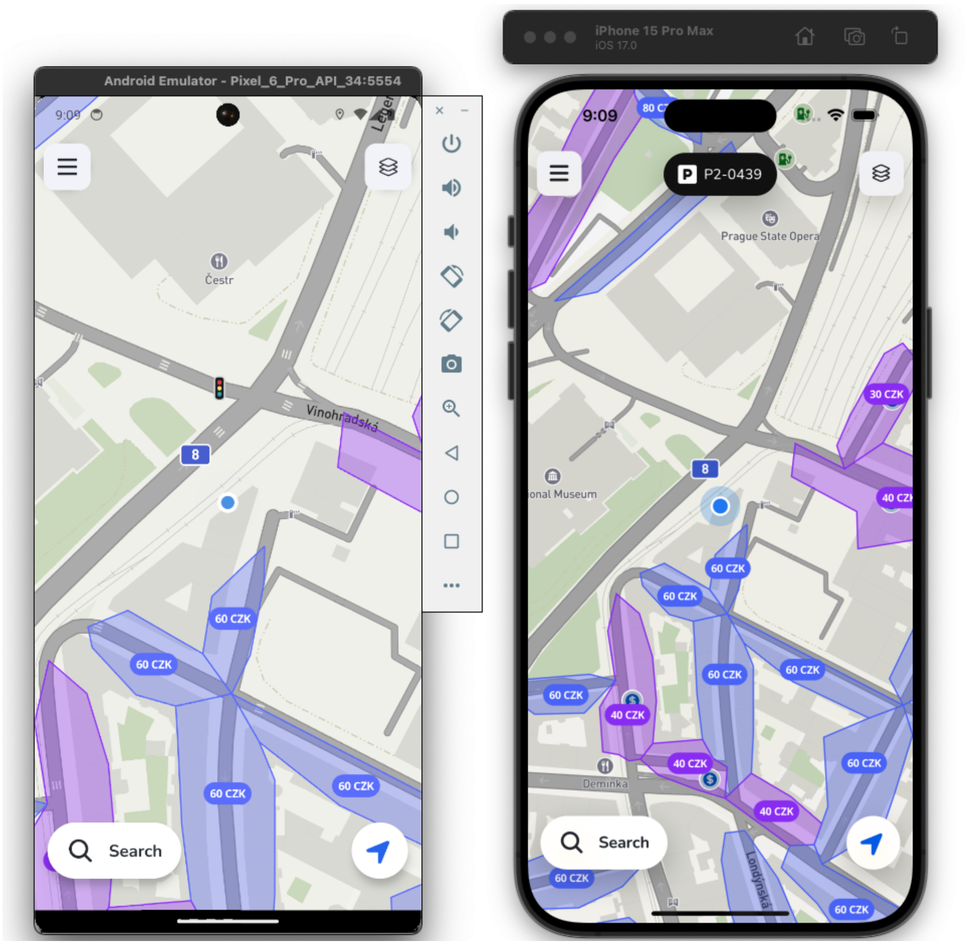


Figure 7.6: The final application on Android and iOS [author]

Chapter 8
User Acceptance Testing

To that point, the MVP version of the mobile application is ready for User Acceptance Testing. UAT ensures that the developed software works as intended in real-world situations. The main goal of the testing process is to validate if the smart parking application covers the specified requirements and has no significant issues in the system logic and UI.

8.1 Participants

The testing session involved a group of 7 people aged 19 to 37 years who drive a car on a daily basis. The participants were chosen through self-selection sampling, where people voluntarily chose to participate in testing. Four individuals also took part in usability testing. The other participants were new to the project, and it was necessary to describe the project goals before the testing session.

8.2 Testing Format

The testing session required additional configuration of the participants' mobile devices to install the developed application. Using participants' devices during testing was essential to ensure the application works correctly on different processors and screen sizes. The testing session started once the configuration was complete and the application was successfully installed.

Testing the application on real devices provided the advantage of capturing all sessions using screen recorders. These recordings are invaluable for reproducing any bugs discovered during testing.

At the end of the session, participants shared their overall experience using the application, mentioning what they liked and disliked, whether they would use the application in the current state, or what stops them from using it in the current state.

8.3 Testing Scenarios

At the beginning of the testing session, participants received a list of tasks they should complete using the application. The tasks covered all the core functionalities, including:

1. Registration
2. Authentication
3. Locating a parking space using map and search
4. Reporting an issue
5. Managing a parking relation

8.3.1 Test Scenario 1 - Registration

Use Case ID: UC1, UC2
Actors: system, person.
Goal: to verify that the person can create a new account with email and password or using Google provider.
Preconditions: the person is not authenticated.

Main scenario:

1. The user requests to open a menu.
2. The system displays the fullscreen modal with the application menu (WF - Menu - UC8)
3. The user navigates to User Settings.
4. The system displays a screen with User Settings (WF - User Settings - UC1, UC2, UC3, UC4)
5. The user chooses to register with email and password and presses on the related button.
6. The system displays the registration screen (WF - Registration - UC2).
7. The user fills in the email and password fields.
8. IF the data is valid AND the database does not have a user with the same email THEN the system displays the email verification screen (WF - Registration - OTP - UC2) ELSE the system warns the user of the error and asks to enter the credentials again.

Alternative scenario: The user chooses to register using Google (Step 5)

1. The system redirects the user to the Google SSO screen.
2. The user chooses Google account and confirms the registration in the system.
3. IF the database does not have a user with the same email THEN the system creates a new account ELSE the system warns the user of the error.

8.3.2 Test Scenario 2 - Authentication

Use Case ID: UC3, UC4
Actors: system, person.
Goal: to verify that the person can sign in to the existing account using email and password or using Google provider.
Preconditions: the person is not authenticated.

Main scenario:

1. The user requests to open a menu.
2. The system displays the fullscreen modal with the application menu (WF - Menu - UC8)
3. The user requests to open user settings.
4. The system displays a screen with user settings (WF - User Settings - UC1, UC2, UC3, UC4)
5. The user chooses to authenticate with email and password and presses on the related button.
6. The system displays the authentication screen (WF - Sign in local - UC2).
7. The user fills in the email and password fields.
8. IF the data are valid AND the database has a user with the same email THEN the system authenticates the user ELSE the system warns the user of the error.

Alternative scenario: The user chooses to login using Google (Step 5)

1. The system redirects the user to the Google SSO screen.
2. The user chooses Google account and confirms the authentication in the system.
3. IF the database has a user with the same email THEN the system authenticates the user ELSE the system warns the user of the error.

8.3.3 Test Scenario 3 - Locating parking spaces

Use Case ID: UC11
Actors: system, person.
Goal: to verify that the person can find a desired parking space.

Main scenario:

1. The user requests to show all parking spaces.
2. The system displays a screen with parking spaces on the map (WF - Map - UC11).

Alternative scenario:

1. The user requests to search for a parking space.
2. The system opens a search screen (WF - Search - UC12).
3. The user starts typing a street name.
4. The system suggests parking spaces starting with the typed term (Search Results - UC12).

8.3.4 Test Scenario 4 - Creating a report

Use Case ID: UC23
Actors: system, person.
Goal: to verify that the person can create and update parking space reports.

Main scenario:

1. The user requests to show all parking spaces.
2. The system displays a screen with parking spaces on the map (WF - Map - UC11).
3. The user selects a desired parking space.
4. The system displays a bottom sheet with parking space details (Parking Space Details - UC24, UC15, UC18, UC19).
5. The user requests to create a report.
6. The system displays a bottom sheet with a report form (Create Report - UC21).
7. The user selects the closure report.
8. The system opens a bottom sheet and prompts the user to select a closure reason (WF - Create Report Closure).
9. The user selects a reason and submits the form.

10. IF the user has already created a report less than 15 minutes ago THEN the system updates the existing report ELSE the system creates a new report.

11. The system opens a bottom sheet with parking space details with a newly created closure report.

Alternative scenario: The user selects the occupancy report (Step 7)

1. The system opens a bottom sheet and prompts the user to select the occupancy level (WF - Create Report Occupancy).

2. The user selects the occupancy level and submits the form.

3. IF the user has already created a report less than 15 minutes ago THEN the system updates the existing report ELSE the system creates a new report.

4. The system opens a bottom sheet with parking space details with a newly created occupancy report.

Alternative scenario: The user selects the charging stations report (Step 7)

1. The system opens a bottom sheet and prompts the user to select the number of available charging stations (WF - Create Report Charging Stations).

2. The user selects the number and submits the form.

3. IF the user has already created a report less than 15 minutes ago THEN the system updates the existing report ELSE the system creates a new report.

4. The system opens a bottom sheet with parking space details with a newly created charging stations report.

8.3.5 Test Scenario 5 - Creating a parking relation

Use Case ID: UC15, UC19
Actors: system, person.
Goal: to verify that the person can create a parking relation on a desired parking space.

Main scenario:

1. The user requests to show all parking spaces.

2. The system displays a screen with parking spaces on the map (WF - Map - UC11).

3. The user selects a desired parking space.

4. The system displays a bottom sheet with parking space details (WF - Parking Space Details - UC24, UC15, UC18, UC19).

5. The user requests to pay for parking.

6. The system prompts the user to select a parking duration (WF - Chose Duration - UC16).

7. The user selects the time and submits the form.

8. The system displays the screen with the current parking relation (WF - Current Parking - UC13, UC14).

Alternative scenario: The user wants to park in the residence district (Step 4)

1. The user requests to park as a resident.

2. The system creates the parking relation and displays the screen with the current parking relation (WF - Current Parking - UC13, UC14).

8.4 User Feedback

After the testing session, participants were given the opportunity to provide additional feedback on their experience using the application. Overall, they appreciated the design and intuitiveness of the user interface, noting that its layout was similar to other smart parking applications, which they found beneficial. Participants particularly liked that parking prices were displayed directly on the map. They also valued the application's in-app navigation feature on iOS, which simplified the process of locating parking spaces. Additionally, people highlighted the ease of creating reports.

Participants also shared what they wanted to be changed or added to the application. The most requested features included:

1. Parking payments
   For the context of the Master's thesis, the parking payment process was omitted. However, the smart parking application must have a parking payment capability.

2. In-app navigation for the Android platform

3. Automated parking detection

4. Notifications when parking comes to the end

5. Prolonging the parking relation

6. View the history of parking relations

Such features are essential for the smart parking application and will be delivered in future releases.

It is also important to mention that nearly half of the participants said they do not often want to create reports themselves. Because of this, future versions of the app will include features that automatically end a parking session when users leave and suggest parking spots when they stop driving. However, participants did like seeing reports from others. This means that if they see that these reports are helpful, they might also start to create their own reports. This shows that making user-generated content more visible and useful could encourage more participation in reporting.

8.5 Detected Bugs

During the testing sessions, multiple bugs were identified. These bugs are detailed in the list below ordered by priority (descending):

1. **Inconsistent Search**
   Sometimes, the application automatically selects a parking space after pressing on a search result, but sometimes, it does not select anything.
2. **Forms**
   Application forms currently do not display error messages when users input invalid data
3. **Missing Translations**
   Some UI elements were not translated into the Czech language.
4. **One-time Password Verification**
   OTP screens do not display information on how much time the OTP code is valid

All the mentioned bugs have low complexity and will be fixed in the nearest releases.

8.6 Summary

The user acceptance testing confirmed that the must-have and should-have system requirements were successfully met. The testing also revealed significant interest in the project, despite some functionalities not yet being implemented. Notably, three individuals expressed willingness to use the application in its current state, while other participants would prefer to wait until the parking payment feature is added.

Chapter 9

The Future of the Application

The user acceptance testing validated the project's viability and confirmed that it covers the requirements for the MVP version of the smart parking application. To further enhance user experience and support parking payments, the author has partnered with TSK Prague. This collaboration aims to integrate precise parking occupancy predictions and payment functionalities. After addressing the identified bugs and implementing essential features, the application will be launched on app stores to gather broader user feedback. The insights gained from user feedback and activity will be crucial in determining the application's future updates and development trajectory, ensuring it continues to meet user needs and expectations effectively.

Chapter 10
Conclusion

The objective of the Master thesis was to develop and validate the MVP version of the platform that aimed at gathering information on parking space availability through crowdsourcing. This objective was fulfilled.

The decision to use crowdsourcing was based on the analysis of existing solutions, indicating that it could be a viable approach due to its low maintenance costs and scalability compared to alternatives such as parking sensors or object detection algorithms.

The formulation of system requirements was guided by insights from the INRIX survey and the functionalities observed in other smart parking applications. These requirements were visualized with class and use case diagrams to ensure clarity and organization in the development process.

In the next step, the Lo-Fi prototype was created for usability testing. This testing phase helped identify and mitigate design flaws, ensuring the overall application intuitiveness and ease of use. Following the results from this phase, the author developed the Hi-Fi prototype that defined the final look of the application.

Based on the system requirements, the author analyzed and compared various technologies that can be used during the application development. Supabase was chosen for the BaaS as it provides a complete backend solution that fulfills all the requirements for the MVP. React Native and Expo were chosen on the frontend side mainly because of the multiplatform support and author's prior experience with these technologies.

The development process began with processing data from the Prague geoportal. Next, the author designed a custom map in Mapbox Studio, visualizing parking spaces, payment machines, and charging stations retrieved from the geoportal. The map setup is followed by the configuration of the Supabase backend. The overall process was straightforward and rarely required writing custom SQL queries. After the backend was ready to use, the author started building the mobile application using a single codebase in React Native. The developed application was then validated with user acceptance testing to ensure it satisfies users' needs.

The author hopes that the developed application will contribute to the city infrastructure and enhance the drivers' experience while searching for vacant parking spots.

Bibliography

[1] *Understanding Smart Cities: An Integrative Framework* [online]. [visited on 2024-01-20]. Available from: `https://scienzepolitiche.unical.it/bacheca/archivio/materiale/949/urbana,%202016-17/smart/S.pdf`.

[2] *The Internet of Things: Opportunities and Challenges* [online]. [visited on 2024-01-20]. Available from: `https://www.europarl.europa.eu/RegData/etudes/BRIE/2015/557012/EPRS_BRI(2015)557012_EN.pdf`.

[3] *TSK: Prague Transportation Yearbook 2022* [online]. [visited on 2024-01-20]. Available from: `https://www.tsk-praha.cz/static/udi-rocenka-2022-cz.pdf`.

[4] *ON- AND OFF-STREET PARKING* [online]. [visited on 2024-01-20]. Available from: `https://www.parking.net/about-parking/on-and-off-street-parking`.

[5] *Parking in Zones* [online]. [visited on 2024-01-20]. Available from: `https://parking.praha.eu/en/parking-options-in-prague/parking-in-zones/`.

[6] *Parking in Blue Zones* [online]. [visited on 2024-01-20]. Available from: `https://parking.praha.eu/en/parking-options-in-prague/parking-in-zones/blue-zone/`.

[7] *Parking in Purple Zones* [online]. [visited on 2024-01-20]. Available from: `https://parking.praha.eu/en/parking-options-in-prague/parking-in-zones/purple-zone/`.

[8] *Parking in Orange Zones* [online]. [visited on 2024-01-20]. Available from: `https://parking.praha.eu/en/parking-options-in-prague/parking-in-zones/orange-zone/`.

[9] *Parking Options in Prague - Kiss Ride* [online]. [visited on 2024-01-20]. Available from: `https://parking.praha.eu/en/parking-options-in-prague/kr-kiss-ride/`.

[10] *Parking Options in Prague - Park Ride* [online]. [visited on 2024-01-20]. Available from: `https://parking.praha.eu/en/parking-options-in-prague/pr-park-ride/`.

[11] *Tackling Traffic Congestion in Urban Areas* [online]. [visited on 2024-01-20]. Available from: `https://towardsdatascience.com/r-cnn-fast-r-cnn-faster-r-cnn-yolo-object-detection-algorithms-36d53571365e`.

[12] KIANPISHEH, MUSTAFFA, Norlia; LIMTRAIRUT, Pakapan; KEIKHOSROKIANI, Pantea. Smart Parking System (SPS) Architecture Using Ultrasonic Detector. *International Journal of Software Engineering and Its Application*. 2012, vol. 6.

[13] *Ultrasonic Sensor Accuracy* [online]. [visited on 2024-01-20]. Available from: `https://senix.com/ultrasonic-sensor-accuracy/#:~:text=Summary%20on%20Accuracy&text=The%20more%20accurate%20ultrasonic%20sensors,1%25%20and%203%25%20accuracy`.

[14] *Does Your Ultrasonic Sensor Need Maintenance?* [online]. [visited on 2024-01-20]. Available from: `https://www.apgsensors.com/about-us/blog/does-your-ultrasonic-sensor-need-maintenance`.

[15] CHINRUNGRUENG, Jatuporn; SUNANTACHAIKUL, Udomporn; TRIAMLUMLERD, Satien. Smart Parking: An Application of Optical Wireless Sensor Network. In: *2007 International Symposium on Applications and the Internet Workshops*. 2007, pp. 66–66. Available from DOI: `10.1109/SAINT-W.2007.98`.

[16] *Understanding Active Passive Infrared Sensors (PIR) and Their Uses | Arrow.com* [online]. [visited on 2024-01-20]. Available from: `https://www.arrow.com/en/research-and-events/articles/understanding-active-and-passive-infrared-sensors`.

[17] *Working Principle of Magnetic Sensors* [online]. [visited on 2024-01-20]. Available from: `https://www.azosensors.com/article.aspx?ArticleID=2620`.

[18] HUTAGALUNG, A J M; SJAMSUDIN, I K; HUTABARAT, D P. IoT-Based Parking Monitoring System using Magnetometer as the Sensor. *IOP Conference Series: Earth and Environmental Science*. 2021, vol. 794, no. 1, p. 012134. Available from DOI: `10.1088/1755-1315/794/1/012134`.

[19] *3 Advantages of Wireless Magnetometers for Vehicle Detection* [online]. [visited on 2024-01-20]. Available from: `https://www.bannerengineering.com/us/en/company/expert-insights/3-advantages-wireless-magnetometer-vehicle-detection.html`.

[20] VROEGOP, J. *Installation and Maintenance of Inductive Detector Loops*. 1991. Tech. rep. Transit New Zealand.

[21] *Tackling Traffic Congestion in Urban Areas* [online]. [visited on 2024-02-20]. Available from: `https://nortechdetection.com.au/wp-content/uploads/2019/09/Loopinstallation_an.pdf`.

[22] IBISCH, André; STÜMPER, Stefan; ALTINGER, Harald; NEUHAUSEN, Marcel; TSCHENTSCHER, Marc; SCHLIPSING, Marc; SALINEN, Jan; KNOLL, Alois. Towards autonomous driving in a parking garage: Vehicle localization and tracking using environment-embedded LIDAR sensors. 2013, pp. 829–834. Available from DOI: `10.1109/IVS.2013.6629569`.

[23] ZHANG, Tianya; JIN, Peter J. Roadside LiDAR Vehicle Detection and Tracking Using Range and Intensity Background Subtraction. *arXiv preprint arXiv:2201.04756*. 2022. Available also from: `https://arxiv.org/abs/2201.04756`.

[24] *What is Deep Learning? - Deep Learning Explained - AWS* [online]. [visited on 2024-01-20]. Available from: `https://aws.amazon.com/what-is/deep-learning/`.

[25] *R-CNN, Fast R-CNN, Faster R-CNN, YOLO — Object Detection Algorithms | by Rohith Gandhi | Towards Data Science* [online]. [visited on 2024-01-20]. Available from: `https://towardsdatascience.com/r-cnn-fast-r-cnn-faster-r-cnn-yolo-object-detection-algorithms-36d53571365e`.

[26] GIRSHICK, Ross. Fast R-CNN. *arXiv preprint arXiv:1504.08083*. 2015. Available also from: `https://arxiv.org/abs/1504.08083`.

[27] REN, Shaoqing; HE, Kaiming; GIRSHICK, Ross; SUN, Jian. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. *arXiv preprint arXiv:1506.01497*. 2015.

[28] HE, Kaiming; GKIOXARI, Georgia; DOLLÁR, Piotr; GIRSHICK, Ross. Mask R-CNN. *arXiv preprint arXiv:1703.06870*. 2017.

[29] LIN, Tsung-Yi; DOLLÁR, Piotr; GIRSHICK, Ross; HE, Kaiming; HARIHARAN, Bharath; BELONGIE, Serge. Feature Pyramid Networks for Object Detection. *arXiv preprint arXiv:1612.03144*. 2017.

[30] LIU, Wei; ANGUELOV, Dragomir; ERHAN, Dumitru; SZEGEDY, Christian; REED, Scott; FU, Cheng-Yang; BERG, Alexander C. SSD: Single Shot MultiBox Detector. In: *Lecture Notes in Computer Science*. Springer International Publishing, 2016, pp. 21–37. ISBN 9783319464480. ISSN 1611-3349. Available from DOI: `10.1007/978-3-319-46448-0_2`.

[31] REDMON, Joseph; DIVVALA, Santosh; GIRSHICK, Ross; FARHADI, Ali. *You Only Look Once: Unified, Real-Time Object Detection*. 2016. Available from arXiv: `1506.02640 [cs.CV]`.

[32] LIN, Tsung-Yi; GOYAL, Priya; GIRSHICK, Ross; HE, Kaiming; DOLLÁR, Piotr. *Focal Loss for Dense Object Detection*. 2018. Available from arXiv: `1708.02002 [cs.CV]`.

[33] HOWARD, Andrew G.; ZHU, Menglong; CHEN, Bo; KALENICHENKO, Dmitry; WANG, Weijun; WEYAND, Tobias; ANDREETTO, Marco; ADAM, Hartwig. *MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications.* 2017. Available from arXiv: `1704.04861 [cs.CV]`.

[34] *Crowdsourcing: Definition, How It Works, Types, and Examples* [online]. [visited on 2024-02-20]. Available from: `https://www.investopedia.com/terms/c/crowdsourcing.asp`.

[35] JI, Zhanlin; GANCHEV, Ivan; O'DROMA, Máirtín; ZHANG, Xueji. A cloud-based intelligent car parking services for smart cities. 2014, pp. 1–4. Available from DOI: `10.1109/URSIGASS.2014.6929280`.

[36] ZHENG, Yanxu; RAJASEGARAR, Sutharshan; LECKIE, Christopher. Parking Availability Prediction for Sensor-Enabled Car Parks in Smart Cities. In: *2015 IEEE Tenth International Conference on Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP).* IEEE, 2015, pp. 1–6. Available also from: `https://www.academia.edu/35482414/Parking_Availability_Prediction_for_Sensor_Enabled_Car_Parks_in_Smart_Cities`.

[37] *eParkomat - We predict real time parking situation* [online]. [visited on 2024-05-21]. Available from: `https://www.eparkomat.com/`.

[38] YIN, Ming; WORTMAN VAUGHAN, Jennifer; WALLACH, Hanna. Understanding the Effect of Accuracy on Trust in Machine Learning Models. In: *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems.* Glasgow, Scotland Uk: Association for Computing Machinery, 2019, pp. 1–12. CHI '19. ISBN 9781450359702. Available from DOI: `10.1145/3290605.3300509`.

[39] *The Impact of Parking Pain in the US, UK and Germany* [online]. [visited on 2024-02-20]. Available from: `https://s3.documentcloud.org/documents/3892952/INRIX-Parking-Research-FINAL-Low-Res.pdf`.

[40] *ParkMobile* [online]. [visited on 2024-02-20]. Available from: `https://parkmobile.io/`.

[41] *SpotHero* [online]. [visited on 2024-02-20]. Available from: `https://spothero.com/`.

[42] *ParkWhiz* [online]. [visited on 2024-02-20]. Available from: `https://www.parkwhiz.com/`.

[43] *CityMove* [online]. [visited on 2024-02-20]. Available from: `https://www.citymove.app/`.

[44] *EasyPark* [online]. [visited on 2024-02-20]. Available from: `https://easypark.cz/`.

[45] *PidLitacka* [online]. [visited on 2024-02-20]. Available from: `https://pidlitacka.cz/en`.

[46] *MPLA* [online]. [visited on 2024-02-20]. Available from: https://ke-utc.appspot.com/static/index.html.

[47] *Waze* [online]. [visited on 2024-02-20]. Available from: https://www.waze.com/.

[48] *Minimum Viable Product - What is a MVP and why is it important?* [online]. [visited on 2024-05-20]. Available from: https://www.productplan.com/glossary/minimum-viable-product/.

[49] FOWLER, Martin. *Destilované UML* [online]. 1. vyd. Boston: Grada, 2005 [visited on 2024-05-01]. ISBN 9788024720623.

[50] *Chapter 10: MoSCoW Prioritisation | The DSDM Handbook | Agile Business Consortium* [online]. [visited on 2024-05-01]. Available from: https://www.agilebusiness.org/page/ProjectFramework_10_MoSCoWPrioritisation.

[51] *Low-Fidelity Prototyping: What Is It and How Can It Help? | Figma* [online]. [visited on 2024-05-01]. Available from: https://www.figma.com/resource-library/low-fidelity-prototyping/.

[52] *The Wizard of Oz Method in UX* [online]. [visited on 2024-05-01]. Available from: https://www.nngroup.com/articles/wizard-of-oz/.

[53] *High-Fidelity Prototyping: What Is It And How Can It Help? | Figma* [online]. [visited on 2024-05-01]. Available from: https://www.figma.com/resource-library/high-fidelity-prototyping/.

[54] *Supabase vs Pocketbase comparison — Restack* [online]. [visited on 2024-05-01]. Available from: https://www.restack.io/docs/supabase-knowledge-supabase-vs-pocketbase-comparison.

[55] *Pricing Fees | Supabase* [online]. [visited on 2024-05-02]. Available from: https://supabase.com/pricing.

[56] *Pricing Fees | Supabase* [online]. [visited on 2024-05-02]. Available from: https://reactnative.dev/.

Appendix A
Abbreviations

Abbreviation	Meaning
MVP	Minimum Viable Product
LIDAR	Light Detection And Ranging
ICT	Information and Communication Technologies
ERPS	European Parliamentary Research Service
UAT	User Acceptance Testing
DBMS	Database Management Systems
GPS	Global Positioning System
K&R	Kiss & Ride
P&R	Park & Ride
IR	Infrared
CNN	Convolutional Neural Network
AI	Artificial Intelligence
RoI	Region Of Interest
FPN	Feature Pyramid Network
SSD	Single Shot Detection
RPN	Region Proposal Network
YOLO	You Only Look Once
(N)FR	(Non)Functional Requirement
UML	Unified Modeling Language
NPM	Node Package Manager
SSO	Single Sign On
UC	Use Case
Lo-Fi	Low-Fidelity
Hi-Fi	High-Fidelity
BaaS	Backend as a Service
API	Application Programming Interface
SDK	Software Development Kit
CPU	Central Processing Unit
RAM	Random-Access Memory
UI	User Interface
HTML	HyperText Markup Language
WF	WireFrame

Appendix B

Folder Structure of the Attached CD

```
├── diagrams/ .................................. The UML diagrams
├── prototypes/
│   ├── Lo-Fi/ ................................. The Lo-Fi prototype
│   ├── Hi-Fi/ ................................. The Hi-Fi prototype
│   └── Hi-Fi Description.pdf ........... The Hi-Fi prototype details
└── code/ ............................... The application source code
```