

# **Perfect Parking**

## **An AI System to Assist Drivers Finding Parking in Busy Cities**

---

**Rhys Quilter**  
**K00241356**

A Final Year Project submitted as a requirement of  
the Technological University of Shannon for the  
degree of Bachelor of Science (Honors) in Software  
Development.

Supervised by:  
John Jennings

## **Acknowledgments**

I would like to thank my supervisor John Jennings and my second reader Gerry Guinane for helping me to complete my research. In addition, I would like to thank my parents Sean and Beverley, my sister Ciara, my girlfriend Laura and my LSU tutor Mark for their support during my time at TUS.

## **Ethical Declaration**

I wish to declare that this project and document is wholly my own work except where I have made explicit reference to the work of others. I have read the Department of Information Technology Final Year Project guidelines and relevant institutional regulations, and hereby declare that this document is in line with these requirements.

I have discussed, agreed, and complied with whatever confidentiality or anonymity terms of reference were deemed appropriate by those participating in the research and dealt appropriately with any other ethical matters arising, in line with the TUS Research Ethics Guidelines for Undergraduate and Taught Postgraduate Programmes policy document.

*Rhys Quilter*

May 1, 2023

## **Abstract**

The aim of Perfect Parking is to create a parking system that will replace outdated systems and to help stop the widespread problem that is parking in our cities. The goal in this project is to improve the effectiveness of finding parking spaces and to also relieve the stress of the users looking for parking by implementing new and innovative features. This will be done by building a client server APIs and website that will show the user exactly where the parking is. The sever will be supported by parking monitor clients powered by OpenCV to detect if a parking spot has been filled or made empty.

# Table of Contents

Acknowledgments.....	ii
Ethical Declaration.....	iii
Abstract .....	iv
Table of Contents .....	v
Table of Figure.....	xi
Chapter 1    Introduction .....	13
1.1    The academic objectives.....	13
1.2    The problem .....	13
1.3    Objectives .....	14
1.4    The scope of the solution.....	15
1.5    The developed solution .....	16
1.6    Report structure .....	17
Chapter 2    Literature Review.....	18
2.1    Parking Business Sector .....	18
2.1.1    Irish Trends .....	18
2.1.2    Global Trends.....	18
2.1.3    Statistics .....	19
2.2    Real-time data.....	19
2.2.1    The importance of Real-time data.....	19
2.2.2    How Google collects and uses data.....	20
2.3    Statistics and Big Data .....	21
2.3.1    Why is Big Data important?.....	22
2.3.2    Problems with gathering data.....	22

2.3.3	Urban Planning and Infrastructure Development .....	23
2.4	Computer Vision .....	23
2.4.1	Applications of Computer Vision .....	23
2.4.2	Object Recognition and AI .....	24
2.4.3	Cascading classifiers .....	25
2.4.4	The background of Classifiers (Haar-like features) .....	25
2.4.5	Future implications of Computer Vision in Parking Systems.....	26
2.5	Distributed Systems.....	27
2.5.1	REST APIs .....	27
2.5.2	Benefits and Use Cases of Rest APIs.....	28
2.5.3	Role of REST APIs in Distributed Applications .....	28
2.6	A Software Solution .....	29
2.7	Agile Software Development Paradigm.....	30
Chapter 3	Analysis and Design.....	32
3.1.1	Providing the website using Django .....	32
3.1.2	Web Application Architecture .....	32
3.1.3	Model Creation and Database Management .....	32
3.1.4	User Request Handling .....	32
3.1.5	View Functions and Templates.....	33
3.1.6	Templates for Dynamic Content .....	33
3.1.7	Rendering the Response.....	33
3.2	Providing a Service to the End User.....	33
3.2.1	Real-Time Parking Space Availability.....	33
3.2.2	Geolocation Retrieval and Query .....	33

3.2.3	Probability Calculation and Presentation .....	34
3.2.4	User Friendly Interface .....	34
3.2.5	Real Time Updates .....	34
3.3	Real-Time Communication and Parking Data Updates: REST API Integration 34	
3.3.1	Car-Park Camera Monitoring API and Communication via REST .....	35
3.3.2	Client-Server Interaction .....	35
3.3.3	Parking Data Updates.....	35
3.3.4	Real Time Updates and Reliability .....	35
3.4	Automated Real-Time Parking Detection .....	36
3.4.1	Seamless Integration and Centralized Updates .....	36
3.5	Utilizing Computer Vision in Perfect Parking .....	36
3.6	The Client Application .....	36
3.6.1	Image Processing: Identifying a parking space's occupancy.....	36
3.7	Application Database design .....	36
3.8	System Actors.....	38
3.9	Use Case Descriptions .....	39
3.9.1	Use Case: Find Parking .....	39
3.9.2	Use Case: Register User .....	40
3.9.3	Use Case: Update Parking Lot Status .....	41
3.9.4	Use Case: User changes password. ....	42
3.10	User Parking Sequence diagram.....	43
3.11	Technologies .....	44
3.11.1	Computer Vision: OpenCV.....	44

3.11.2	Anaconda.....	45
3.11.3	Django .....	45
3.12	The Website.....	46
3.13	Website Layout.....	47
3.13.1	Parking Lots View .....	47
3.13.2	Parking Lot View .....	47
3.13.3	Search View .....	48
3.13.4	Parking Lot Monitor.....	49
3.13.5	Know your Location Feature .....	49
3.13.6	Search Near Me.....	50
3.13.7	Login .....	50
3.13.8	Sign up (User Registration).....	51
3.14	Rest API layout .....	52
3.14.1	API Root .....	52
3.14.2	Parking Lots .....	52
3.14.3	Parking Lot Monitors .....	53
3.14.4	Parking Lot Monitor.....	54
3.14.5	Parking Lot.....	54
Chapter 4	Implementation .....	56
4.1	Project Management.....	56
4.1.1	Tools Used .....	56
4.1.2	How to Install Anaconda.....	57
4.1.3	Source Control and versioning.....	58
4.2	To install and run the project.....	58



4.2.1	Run Website Server.....	58
4.2.2	Run the Client Parking Monitor .....	59
4.2.3	Mock Client Parking Monitor .....	64
4.3	How Object Recognition works in Perfect Parking clients .....	65
4.4	Implementation of Perfect Parking Web Application .....	65
4.4.1	Model Based Data Management .....	65
4.4.2	User Friendly Interface .....	66
4.4.3	Views and Templates .....	66
4.4.4	Real time Data Integration .....	66
4.4.5	User Query Tracking.....	67
4.4.6	Implementation Highlights .....	67
4.5	Implementation of Parking Monitoring Client App .....	67
4.5.1	A quick note on the original source code quality.....	68
4.5.2	Refactoring/Modifications .....	70
4.6	Perfect Parking with Django .....	75
4.6.1	Customising Django.....	76
4.7	Enable Django REST framework.....	81
4.7.1	Server .....	82
4.7.2	Working with GPS data .....	82
4.8	Custom tags .....	83
4.9	HTML Template for Displaying Parking Monitor Information with Distance Calculation Getting user's location.....	84
4.9.1	How They Work Together .....	85
4.10	Logging Data for Statistical Analysis. ....	86

4.10.1	Parking Lot Log .....	87
4.10.2	Parking Request Log .....	87
Chapter 5	Testing and Results .....	89
5.1	Unit Testing .....	89
5.1.1	Unit Tests Used in The Project .....	89
5.2	Behaviour Driven Development (BDD) Testing .....	90
5.2.1	BDD In Perfect Parking .....	91
5.3	Test Driven Development (TDD).....	91
5.3.1	TDD in Perfect Parking.....	93
5.4	Functionality .....	94
5.5	Usability .....	95
Chapter 6	Conclusions .....	96
6.1	Application Performance.....	96
6.2	Future Development .....	96
6.3	Reflections .....	98
References	.....	99

## Table of Figure

Figure 1: Map Camera Counting .....	16
Figure 2: CSO.ie - Factors that would encourage more driving, .....	18
Figure 3: Google Example Data.....	21
Figure 4: Distributed Clients communicating with a server on a Star Network .....	27
Figure 5: Agile Development (Feer, 2020) .....	31
Figure 6: Database Design .....	37
Figure 7: User Use Case Diagram.....	38
Figure 8: User Parking Sequence Diagram.....	43
Figure 9: Parking Lots View .....	47
Figure 10: Parking Lot View.....	48
Figure 11: Search View.....	48
Figure 12: Parking Lot Monitor View .....	49
Figure 13: Location Sharing .....	50
Figure 14: Search Near Me Feature .....	50
Figure 15: User Login .....	51
Figure 16: User Registration .....	51
Figure 17: Django Rest Framework API Root.....	52
Figure 18: Parking Lot List.....	53
Figure 19: Parking Lot Monitor List.....	54
Figure 20: Parking Monitor Instance .....	54
Figure 21: Parking lot Instance .....	55
Figure 22: Gantt Chart .....	56
Figure 23: Class Diagram .....	81

Figure 24: How they work together .....	86
Figure 25: BDD development cycle (Collidu, n.d.) .....	91
Figure 26: How TDD Works (BasuMallick, 2022) .....	92
Figure 27: Parking Test Data Histogram .....	96
Figure 28: User Request Talley for A Given Monday .....	97

## **Chapter 1 Introduction**

Due to an increase in in the number of cars being used in limerick city and other cities in Ireland, finding a solution to car parks has now become vital. The old-fashioned way of parking was that everyone would just leave their cars parked in the streets until they were needed again, this however caused major traffic congestions in towns and cities. Shop owners also got hugely impacted as there wouldn't be enough room for staff to park no mind the customers looking to go into their shops which was damaging for their business, it is undeniable that car parks are a very important factor in society, by having parking spaces it reduces illegal parking which would have increased congestions on the road and increasing travel time.

Parking back in the 1980s – 1990s wasn't such a big issue as there wasn't very many cars on the road as people couldn't afford to have a car unless they were wealthy, nowadays however it is very hard to find available parking spaces in places such as cities, and of course universities especially during rush hour. Since limerick City is a big city for students to come and study in with there being over 16,000 students attending university of limerick and just under 2000 students in TUS and Mary I, this brings so much more motor vehicles into limerick city which is a city already struggling with car parking. (University of Limerick, 2022)

The goal is to try and reduce this issue by building a new innovative parking app and to try to help motorists stop stressing about this parking crisis.

### **1.1 The academic objectives**

The academic objectives of this project are to study and gain experience working with AI and object detection in images. The chosen problem used for this study is to help to reduce the traffic congestion in cities such as Limerick.

### **1.2 The problem**

Ineffective ways of finding an available parking space which is a waste of time, very fuel consuming and causes traffic jams. hen road users are looking to find an available parking space they end up wasting time and using a lot of fuel from them driving around the car spark or the block multiple times hopping to find a space, on average people spend 17

hours per year driving around looking for parking spaces (Quellmalz, 2021). By developing Perfect Parking, it is hoped to make the parking process in college campus and in the city seamless and stress free, by doing this it is hoped that time and fuel waste can be reduced for drivers when they are looking for parking.

#### **Lack of visual parking space availability.**

When road users are driving the visuals of the eye are limited, the vision is blocked by many obstacles such as the cars frame causing blind spots, other cars, trees and much more. It can be understood that the road users can find it difficult to spot an available parking space if it's in the distance or behind other cars. This has turned into an important problem with road users and as a result they waste time and fuel trying to spot an available space. (Ponnambalam, 2020)

#### **Lack of knowledge of towns or cities.**

Limerick is a city where people migrate to for education or tourism, and with this brings more road users. When drivers first come to Limerick City, they must learn the road routes and with this where the parking is. A lot of road users that drive in a new place start to panic and get anxious when they try and find parking. This can cause them to be in the wrong lanes and cause traffic congestion. But with the Perfect Parking app it will allow users to plan their route to the parking of their choice and follow directions on the phone to the car park. By doing this that it will keep new road users in the city calm so they can enjoy their holiday or for students teach them the road routes and the best places to park.

### **1.3 Objectives**

The Perfect Parking application aims to develop a user-friendly parking system that utilizes object detection technology.

The envisioned app is a distributed application that leverages the power of computer vision and client-server architecture working across the internet. The app's goal is to illustrate the possibility and potential of using computer vision technology to manage parking spaces more efficiently. Perfect Parking uses a distributed strategy, using remote clients equipped with cameras to capture images of parking spaces. These clients use computer vision techniques to perform real-time image processing and monitoring, sending the parking-occupancy (which indicates whether a parking space is vacant or

occupied) result to a central server. Through this proof-of-concept application, Perfect Parking showcases the benefits of distributed monitoring and computer vision for effective parking space management.

The technology watches a video feed and allows the user to query for parking spaces in parking areas. Once the parking monitor detects a change to a parking space it pushes an update to the website with a probability of parking being available. This application could be used in parking lots and towns nationwide.

The objectives of this project solution are:

- To provide a solution that can determine the status of the parking spots i.e., free or occupied.
- To reduce the time motorists spend looking for parking in cities.

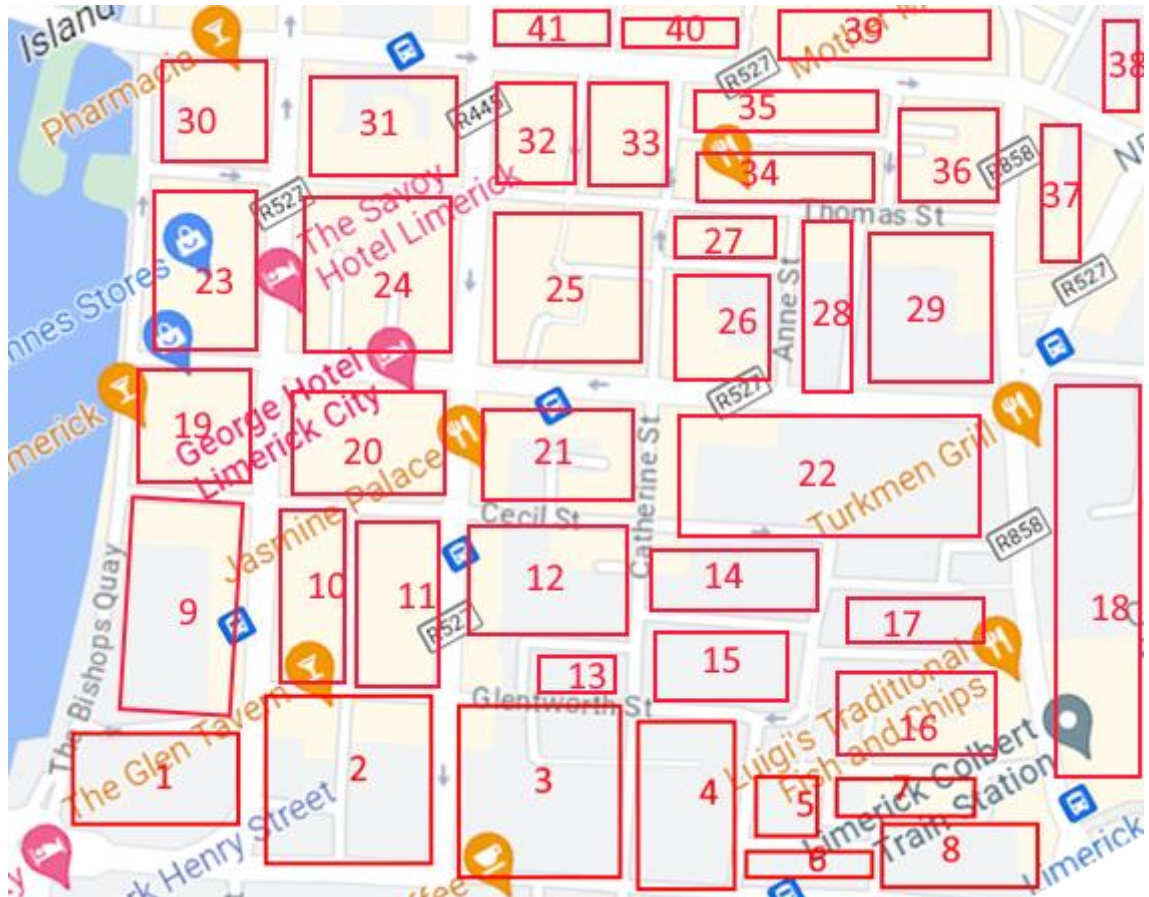
#### **1.4 The scope of the solution**

The scope of the Perfect Parking solution includes both a model car park made from cardboard and toy cars, and prerecorded video feeds, utilizing a laptop and camera setup. In addition to showcasing the car detection capabilities through prerecorded videos, a live feed option was introduced using a camera and laptop to demonstrate the feasibility of real-time video. Due to time and budget constraints, live feeds from cities or colleges were not extensively integrated within the scope of the project. Instead, prerecorded and model car park live feed remains a primary method to illustrate the application's proficiency in detecting available parking spaces. The solution also encompasses a server demonstration, highlighting end-user interactions with the application and its overall viability as a marketable product. By addressing these key components, the solution's focus is refined, facilitating a more concentrated and efficient development process.

It is estimated that it would take about 960 camera-feeds/clients to comprehensively monitor the parking spots of Limerick City. The estimate is derived from the assumption that a street like O'Connell St. will need 2 to 3 camera-feeds/clients for one side of a street.

The calculations are:

- A street needs 3 camera-feeds/clients.
- A block has 4 sides i.e., 4 streets.
- Total number of cameras per block: 12 cameras (3 cameras for 4 streets)
- Limerick has approximately 40 blocks of interest.
- Total number of cameras: 480 cameras (12 cameras for 40 blocks)



**Figure 1: Map Camera Counting**

It's important to note that the actual number of cameras required may differ based on factors such as street layout, street parking allowed, parking density, and camera coverage angles.

## **1.5 The developed solution**

The developed solution is an AI application to assist drivers finding parking in busy cities. The applications are written in Python; The server is built on Django framework and, the client use the OpenCV library to achieve computer object detection. The source code for



the applications can be viewed at <https://github.com/rhysquilter/perfect-parking-fyp/tree/master/notebooks> .

## **1.6 Report structure**

This thesis is structured into five chapters:

### **Literature Review:**

This chapter discusses the different types of data and the problems with gathering data, it also discusses about object recognition and AI. The literature review also touches on the parking business sector and the global and Irish trends, while also discussing the software solution.

### **Analysis and Design:**

This chapter discusses the design of the application such as the use cases and the database design. The chapter also goes on to discuss about computer vision and how OpenCV makes it possible.

### **Implementation:**

This chapter discusses how the project was implemented. It also discusses the tools that were used to develop this application such as the Django framework and how other tools such as GitHub were used to make the development possible.

### **Testing and Results:**

The testing and results chapter goes on to show how different software testing techniques were used to test the program such as Unit Testing.

### **Conclusions:**

The Analysis and Design also discusses a look to the future which is about how this project could be scaled up in the future.

## Chapter 2 Literature Review

A brief review of the literature about Parking monitor software, the business sector, Big data/live, disturbing comped system and AI/Computer vision

### 2.1 Parking Business Sector

Parking management software is expected to see a compound annual growth rate (CAGR) of 12.8% and expand to a global market size of \$11.3 billion by 2024. Additionally, it is anticipated that the market for artificial intelligence (AI) in parking management will increase by 20% yearly to \$1.5 billion by 2025. With these numbers in mind, it is obvious that the parking sector needs creative solutions that can harness the potential of AI to enhance parking management.

#### 2.1.1 Irish Trends

Statistics from the Central Statistics office (CSO) of Ireland gives several suggestions that could encourage people to drive more in the future. In the top three are traffic congestion, and access to parking. (CSO.ie, 2019)

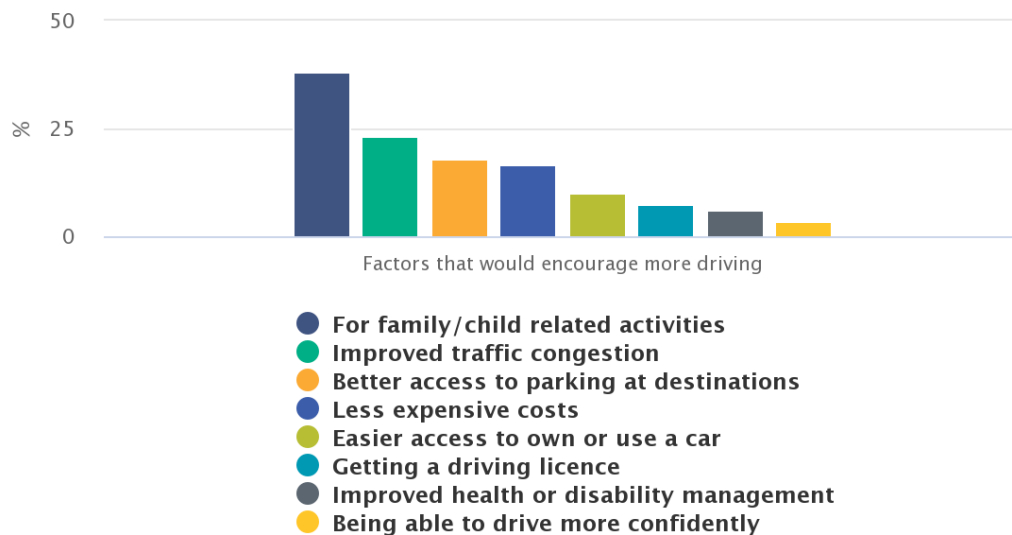


Figure 2: CSO.ie - Factors that would encourage more driving,

#### 2.1.2 Global Trends

The present approaches for locating parking spaces are inefficient and wasteful, requiring a large amount of time and fuel. People spend an average of 17 hours a year searching for

parking spaces, according to (McCoy, n.d.) and Irish people waste four days a year. (Sawer, 2017) the solution intends to end this time and energy waste which will be advantageous to both users and the environment.

It is anticipated that the global market for automated parking systems will increase from \$1.3 billion in 2020 to \$3.6 billion in 2025. By creating an AI-powered parking management system that can maximize parking spot use, lessen traffic congestion, and more, the Perfect Parking project seeks to meet this expanding need. and improve the overall user experience for drivers. (abdalslam, 2023).

### **2.1.3 Statistics**

The utilization of statistics in modern technology has altered the way we interact with our surroundings, especially in settings where population density and activity levels can have a substantial impact on decision-making. An example of this integration may be found in the digital realm, where platforms such as Google utilize data to present users with real-time information into the congestion and availability of various businesses.

In such circumstances, the combination of statistics and technology demonstrates the power of data-driven insights. As we progress into the digital age, the refinement of algorithms and the integration of more diverse data sources will most likely improve the accuracy of these estimates, providing users with a valuable tool for efficiently planning their activities and businesses with a way to better cater to their customers' needs.

## **2.2 Real-time data**

Real time data is data that is available as soon as its created and acquired. The data is pushed to clients as soon as its collected and is immediately available without delay. T this is crucial for supporting live, in the moment decision making (Splunk, 2021). Parking Applications need a large amount of live data to provide users with useable information.

### **2.2.1 The importance of Real-time data**

Real-Time data is a necessity to stay relevant for today's business and it needs to be delivered by sophisticated electronic communications tools such as digital signage and data dashboards, to remain appealing to today's tech savvy workforce from call centres to retailers. (Barnett, 2017)

Real time data is important in many parking applications, these applications use real-time data to show users if there is parking spaces in the carpark which they have selected, this data can be gathered by the carpark having a barrier that counts the amount of cars that go in or the amount of cars that exit, some carparks also have sensors on each of the parking spaces this allows users to see what actual spaces are available, this is the most ideal as it allows people that need disabled parking to see if that type of parking space is available.

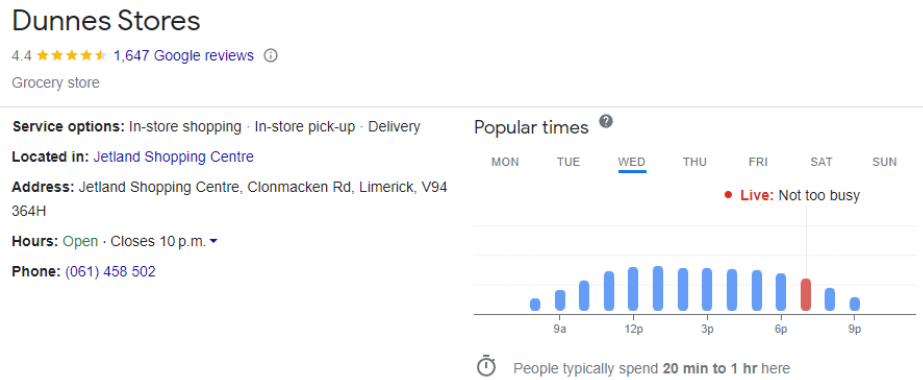
### **2.2.2 How Google collects and uses data**

Google collects endless amounts of real time data. They use a device that billions of people in the world have i.e., smart phones. (Turner, 2023). When people sign into google on their phone Google starts creating real time data and transmitting in real time to google.

When looking for a business on Google basic information such as location and opening hours is provided and data that indicates the company's level of activity at various times of the day.

Google accomplishes this combination of data sources, which includes real-time mobile device GPS position data, the user's query, user-generated evaluations to create reliable estimates of how crowded a location may be at any given time. (Reid, 2016)

Google using the saved data and statistical analysis can predict peak hours of a business and thus potentially longer wait times by evaluating patterns and trends from a wide range of data points. It can pinpoint times when a site is less busy, providing users to avail of times with shorter lines and a more convenient experience for consumers. This data not only helps individuals make informed decisions about when to visit a location, but it also allows businesses to optimize their operations, staffing, and customer service methods based on anticipated traffic. (Ashish, 2022)



**Figure 3: Google Example Data**

## 2.3 Statistics and Big Data

A user wants to find parking for dinner this evening at 6 pm. Live data does not exist for the future, but predictions can be made using historical data. We looked at how Google collects large amounts of live data and uses statistics to make predictions for shops. This large amount of data is referred to as big data.

Big Data is a combination of structured, semi structured, and unstructured data that is collected by organizations, this data can be mined for information to be used in many projects such as machine learning, predictive modelling, and other analytics applications. Big Data is often characterized by the three Vs.

- The large **Volume** of data in many environments.
- The wide **Variety** of data types frequently stored in Big Data systems.
- The **velocity** at which much of the data is generated, collected, and processed.

These characteristics were first identified in 2001 by Doug Laney, they were then further popularized in 2005 by an analyst at a consulting firm called Meta Group.

Big Data doesn't equate to any specific amount of data, Big Data deployments often involve terabytes, petabytes or even in some cases exabytes of data that is created and collected over time. (Botelho, 2020).

Big data refers to a wide range of digital sources that provide valuable insights into human behaviour and preferences. Examples such as:

- User browsing history provides web interactions and engagement patterns.
- Search queries provide insights into information-seeking behaviours.

- Emails reveals communication patterns.

### **2.3.1 Why is Big Data important?**

Big Data importance lies in the fact of how a company utilizes the gathered data. Every company uses its gathered data in its own way, the more a company can gather its data the more the company can grow.

Big Data provides valuable insights into customers that companies can use to refine their marketing, advertising, and promotions by doing this they can increase customer engagement and conversion rates. (TechVidan, 2022)

Big Data is huge in the medical industry, medical researchers can identify disease signs and risk factors, this can help the doctors diagnose illnesses and medical conditions in patients. A combination of data from electronic health records and the web can give healthcare organizations and government agencies up to date information on infectious diseases threats or outbreaks, we have seen this in the past with the pandemic and how the HSE in Ireland were able to monitor the amount of covid – 19 infections per county, and how they were able to create the Covid App with this data. (Botelho, 2020)

### **2.3.2 Problems with gathering data.**

Collecting Big Data can be a challenging task that presents various problems, irrespective of the domain or industry. The sheer amount of data involved, which may be tiresome and challenging to handle, is one of the main problems. Furthermore, the development and implementation of the specialized tools and techniques needed for Big Data collection can be expensive and time-consuming. Additionally, because the data is frequently gathered from a variety of sources, some of which might not be trustworthy, ensuring the quality and accuracy of the data can be a significant challenge. In addition, Big Data analysis and interpretation can be challenging tasks that need for specialized knowledge and abilities to spot patterns, trends, and insights. Addressing these challenges effectively is critical to realizing the potential benefits of Big Data, including improved decision-making, increased efficiency, and enhanced innovation. (Sharma, 2022)

### **2.3.3 Urban Planning and Infrastructure Development**

The integration of data-driven insights from platforms such as Google has become a useful tool in determining the need for new car parking facilities within a city or locality in the realm of urban planning and infrastructure development. The abundance of information collected and made available to users via such platforms not only assists individual decision-making but also provides a complete overview of traffic patterns and congestion levels, which can inform larger-scale urban planning projects.

Data offered by platforms such as Google, particularly real-time information on company activity, provides insight into the dynamics of urban mobility and the need for parking spots. When aggregated over time and across multiple sites, this data can reveal major trends and patterns in parking demand. This data can be used by urban planners and city officials to identify places that typically experience high congestion and low parking availability.

In turn, the data can reveal situations when existing parking infrastructure is underutilized. If specific places continually display low traffic levels and plenty of parking, it may imply that the present parking facilities are properly fulfilling the needs of the community. In such instances, investing in new car parking structures may not be a high-priority activity, allowing city planners to more effectively allocate resources to other important urban development initiatives.

## **2.4 Computer Vision**

Computer vision is a multifaceted field of study aimed at training computers how to interpret and process visual information from images or videos. It entails the creation of algorithms and approaches for extracting useful insights from visual data. (Simplilearn, 2023). In the context of the proposed Parking application Computer vision can detect the presence of vehicle in a parking spot in real-time by utilizing image processing, pattern recognition, and feature extraction.

### **2.4.1 Applications of Computer Vision**

Computer vision has a wide range of applications across various industries. Some notable applications include:

- **Object Detection and Recognition:** Computer vision algorithms can identify and classify objects within images or video streams. This technology is used in autonomous vehicles, surveillance systems, and facial recognition systems.
- **Augmented Reality:** Computer vision enables the overlay of digital information onto the real world, enhancing user experiences in areas such as gaming, advertising, and training simulations.
- **Medical Imaging:** Computer vision techniques assist in medical image analysis, aiding in the detection and diagnosis of diseases, as well as in surgical planning and monitoring.
- **Quality Control and Inspection:** Computer vision can automate quality control processes by identifying defects or anomalies in products on assembly lines, ensuring consistency and accuracy.
- **Robotics and Automation:** Computer vision plays a vital role in robotics, enabling robots to perceive and understand their environment, navigate autonomously, and interact with objects and humans. **(needs Citation)**

#### 2.4.2 Object Recognition and AI

Object recognition refers to the process of teaching a computer how to identify and classify objects within digital images or videos. It's like teaching a child to recognize different objects such as cars, chairs, or animals. Artificial intelligence, or AI, is the field of computer science that deals with creating machines that can perform tasks that typically require human intelligence, such as learning, reasoning, and problem-solving. AI techniques like deep learning, which is a subset of machine learning, are often used in object recognition systems to train algorithms to recognize and classify objects. This technology has a wide range of applications, from self-driving cars to medical diagnosis to robotics. (Tech Target, n.d.)

Object detection is a technique in computer vision that involves detecting objects of interest within an image or video stream. Finding objects within a picture and categorizing them into various categories are the goals of object detection. Since object detection requires locating and recognizing multiple objects inside a picture, it is a more advanced technique than object recognition (Patel, 2020).



### **2.4.3 Cascading classifiers**

Cascade classifiers One kind of machine learning technique used in computer vision for object detection is called a cascading classifier. In their groundbreaking study "Rapid Object Detection using a Boosted Cascade of Simple Features" published in 2001, Viola and Jones introduced them for the first time. The approach is based on the concept of "cascading" the solution of a complex detection problem into several smaller, easier sub-problems. (Michael Jones, 2001)

The basic idea behind cascading classifiers is to use a series of classifiers, each with increasing complexity, to detect objects of interest. Utilizing a series of classifiers, each with a higher level of complexity, to find things of interest is the main notion underlying cascading classifiers. The input image is classified as either containing the object of interest or not by each classifier in the cascade using a collection of features. Each classifier's attributes are chosen based on their capacity to distinguish between positive and negative samples. (Lee, 2022)

One of the main advantages of cascading classifiers is their ability to achieve high detection rates with low false positive rates. This is accomplished by employing several classifiers, each of which is trained to quickly reject negative samples. As a result, there are fewer false positives because the algorithm can swiftly reject pictures that don't include the object of interest. (Bak, 2023)

When used in object detection tasks like face detection, cascading classifiers have been shown to be highly accurate and effective. (Bak, 2023).

### **2.4.4 The background of Classifiers (Haar-like features)**

Classifiers are based upon Haar wavelets theory using Haar-like features, which are extracted from images using rectangular filters and then fed into a classifier to learn to distinguish between different classes based on the extracted features.

An image feature type used in computer vision for object detection is called a Haar-like feature. They have the name of the Haar wavelet, which Alfred Haar, a Hungarian mathematician, initially proposed in 1909. (Seal, n.d.) The mathematical function known as the Haar wavelet can be used to break down a signal or image into a collection of wavelet coefficients.

By comparing the average pixel values in adjacent rectangular regions of an image, Haar-like features can be extracted from the Haar wavelet. The difference between the sum of pixel intensities in a rectangular region with a light color and the sum of pixel intensities in a rectangle region with a dark color is the precise definition of Haar-like features (Arunachalam, 2014).

These rectangular areas can be positioned anywhere in the image and come in a variety of sizes and shapes. It is feasible to gather details about the texture and structure of a picture at various levels of granularity by computing Haar-like features at various scales and positions in the image.

A Haar Cascade Classifier basically works by detecting features in an image that are characteristic of the object to be scanned for, such as edges, corners, and lines. These features are then used to classify the object.

Haar Cascade Classifiers can be trained to recognize specific objects, such as faces, eyes, and cars, and the code is able to detect and localize these objects within an image.

The Viola-Jones object detection technique, a well-liked algorithm for face detection in photos, makes use of Haar-like features. In this approach, a classifier is trained to differentiate between positive instances (pictures containing the item of interest, such as faces) and negative examples (images devoid of the object of interest), using a set of Haar-like characteristics computed for each sub-region of an input image. (Tyagi, 2021)

One of the advantages of using Haar-like features for object detection is their computational efficiency. They are suitable for real-time applications like video surveillance since they are rapid and effective to compute utilizing integral images. (Bak, 2023)

#### **2.4.5 Future implications of Computer Vision in Parking Systems**

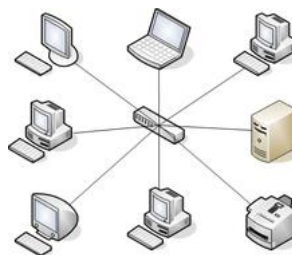
The use of computer vision techniques in parking systems has significant implications for parking management in the future. Parking systems can become more efficient, user-friendly, and automated with the use of computer vision algorithms. Aside from detecting car park occupancy, computer vision can be used for actions such as license plate recognition, vehicle tracking, and advanced parking analytics. Computer vision in parking systems can simplify parking operations, maximize space utilization, minimize

congestion, and improve overall user experience. As computer vision technology advances, it will most certainly disrupt the parking sector and open the door for creative parking solutions. (brouton Lab, n.d.)

## 2.5 Distributed Systems

A distributed system is a networked collection of autonomous entities (such as computers or devices) that collaborate to achieve a common goal or perform a particular task. Unlike a centralized system, in which all processing and decision-making takes place within a single entity, a distributed system divides and distributes the workload across entities over a large area (Lutkevich, 2022).

Each entity in a distributed system, known as a node or client, functions independently and can communicate with other client nodes or a server on a network. These nodes can be distributed geographically and have varying capabilities and rolls within the system. By leveraging the collective resources and capabilities of multiple nodes, a distributed system can achieve higher scalability, fault tolerance, and performance compared to a centralized system. (S.Gillis, 2022)



**Figure 4: Distributed Clients communicating with a server on a Star Network**

The use of distributed systems and apps has grown in popularity in a variety of fields, including e-commerce, social networks, cloud computing, and the Internet of Things (IoT). These systems have advantages such as higher availability, fault tolerance, and scalability, allowing for more efficient resource utilization and improved performance. (Cirrus, 2018).

### 2.5.1 REST APIs

A REST API (Representational State Transfer Application Programming Interface) is a design style and set of constraints used for developing online services that allow systems to communicate and interact with one another via the internet. RESTful APIs are built on

the REST concepts introduced by Roy Fielding in his doctoral dissertation in the year 2000. These APIs enable the seamless interaction between distributed components, offering a standardized approach for accessing and manipulating resources. (Red Hat, 2020)

Resources in a REST API are identified by unique URLs (Uniform Resource Locators), and standard HTTP methods such as GET (retrieve data), POST (create data), PUT (update data), and DELETE (delete data) are employed to manipulate these resources. The key characteristics of a RESTful API include being stateless, having a uniform interface, following a client-server architecture, using self-descriptive messages, and incorporating HATEOAS (Hypermedia as the Engine of Application State) to guide clients through state transitions. (Nolle, 2021)

### **2.5.2 Benefits and Use Cases of Rest APIs**

Rest APIs offer numerous advantages for web and mobile application development. They provide simplicity through their straightforward design, using conventional HTTP methods for ease of implementation. Additionally, they support scalability, enabling applications to efficiently handle increased traffic by distributing requests across servers. The ease of integration is another benefit, as Restful APIs utilize widely recognized web standards like HTTP and JSON, promoting compatibility across various platforms and languages. (Roca, n.d.)

Restful APIs find application in various domains. They are crucial for social media integration, enabling sharing and user authentication services. E-commerce websites use them to manage product catalogues and process orders, while mapping providers leverage Restful APIs for location-based services. On the Internet of Things (IoT), these APIs facilitate communication between devices and cloud platforms, contributing to smart home and industrial automation applications. (PubNub, n.d.) (Ismail, 2023)

### **2.5.3 Role of REST APIs in Distributed Applications**

REST APIs play a pivotal role in enabling seamless communication and interaction between distributed components within applications. REST APIs provide a standardized and scalable approach for developing distributed online applications.

Key advantages of REST APIs include a standardized interface for client-server communication, statelessness which simplifies request processing, decoupling of components for flexibility and scalability, distributed data and resource access through HTTP methods, and support for integration and interoperability with external systems.

## **2.6 A Software Solution**

By leveraging the latest technologies and market trends, the Perfect Parking project seeks to revolutionize the parking industry and provide a comprehensive solution to the challenges faced by parking operators and drivers alike.

In this project, the parking application is being created for educational purposes to solve the widespread parking issue in our cities. Time is lost, gasoline is consumed, and traffic is backed up due to Limerick City's old and inefficient parking systems. To increase the efficiency of identifying parking spaces and reduce the stress experienced by users searching for parking spaces, a new and creative software solution is required.

The lack of obvious availability of parking spaces is another issue with the present parking schemes. Due to the limited vision created by numerous obstructions like the car's frame, trees, or other vehicles, drivers frequently struggle to find an open parking place. Users now spend much more time and fuel because of this issue.

Furthermore, for new road users, finding parking spaces in a new place can be a challenging task. In Limerick City, students and tourists come from different regions to study or visit, and they must learn the road routes and where the parking is. This lack of knowledge can cause them to be in the wrong lanes and create traffic congestion. However, by providing users with the option to plan their route to the parking of their choice and follow directions on their phones to the car park, the application can help users navigate the city's roads and reduce traffic congestion.

Perfect Parking seeks to improve scalability, fault tolerance, and real-time monitoring capabilities by incorporating distributed system principles. Using a distributed design for a client app enables effective utilization of local resources and seamless cooperation of many entities, thereby boosting the overall (coverage) functionality and performance of the system.

To conclude, there is a pressing need for a software solution to address the problems associated with parking in our cities. Perfect Parking aims to provide a solution that is innovative, effective, and user-friendly, with the potential to reduce time and fuel consumption, improve traffic flow, and create a stress-free parking experience for all road users.

## **2.7 Agile Software Development Paradigm**

Agile development is a software development methodology that emphasizes flexibility, collaboration, and iterative development. Agile development teams divide the work into smaller, more manageable chunks called "sprints" rather than working on a project in a linear fashion with a precise plan set in stone from the beginning. A working prototype or product increment that can be tested and evaluated is produced by each sprint, which lasts typically two to four weeks. Agile development promotes regular communication and collaboration between team members and stakeholders and favours working software over documentation. The Agile manifesto identifies four basic values: valuing people over processes and technologies, emphasizing working software over thorough documentation, prioritizing communication with customers over contract negotiations, and reacting to change over planning ahead. (atlassian, n.d.)

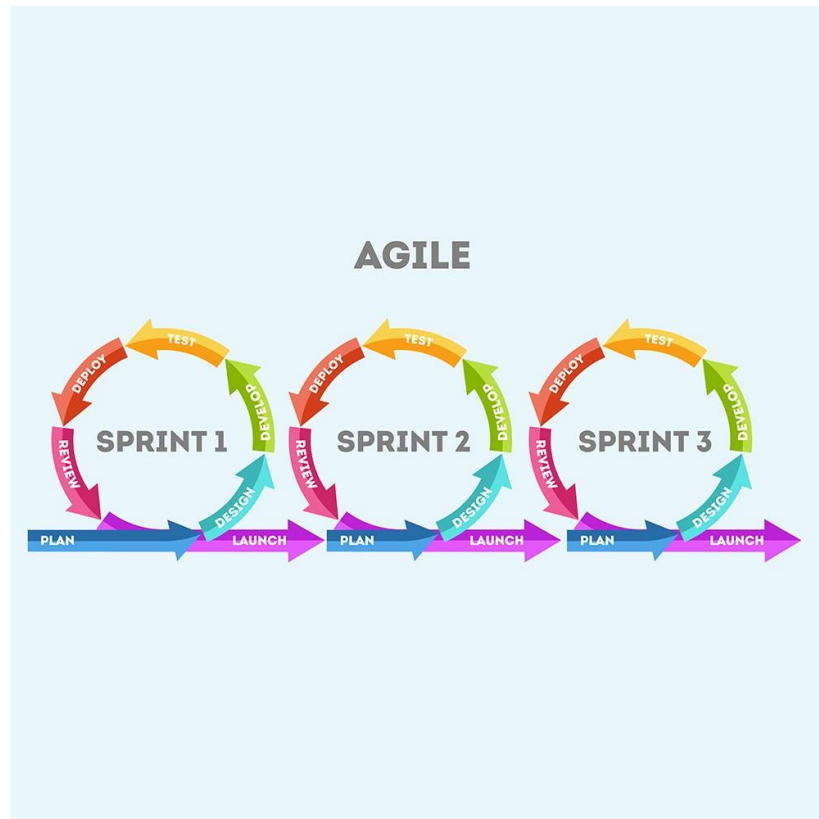


Figure 5: Agile Development (Feer, 2020)

## **Chapter 3 Analysis and Design**

This subsection delves into the architectural analysis and design considerations of the Perfect Parking web system. This analysis provides insights into the underlying technology and design principles that contribute to the Perfect Parking application's user-friendliness and efficiency.

### **3.1.1 Providing the website using Django**

Django, a high-level Python web framework, plays an important role in providing the web-based application to users. Its reliable architecture makes developing and deploying web applications easier. In the context of this project, Django acts as the server-side framework responsible for handling user requests and delivering the website.

Here's a rundown of how Django is utilized to provide the website:

### **3.1.2 Web Application Architecture**

The architecture of the Perfect Parking web application is built on Django's Model-View-Controller (MVC) framework, also known as Model-View-Template (MVT) in Django. This architecture ensures a clear separation of concerns, making it easier to manage the application's various components.

### **3.1.3 Model Creation and Database Management**

The heart of the Perfect Parking web application's database structure is defined within the `models.py` class using Django's Object-Relational Mapping (ORM) system. This class defines the attributes and structure of the database tables, including parking lots and monitors. For instance, the `ParkingLot` model encapsulates crucial details such as name, address, hours, and availability. Similarly, the `ParkingLotMonitor` model represents monitor data, including location, availability probability, and last update timestamp.

### **3.1.4 User Request Handling**

When a user accesses the website, their request is routed through Django's URL dispatcher. This dispatcher maps the URL to a specific view function that will process the request.



### **3.1.5 View Functions and Templates**

Python view functions handle the processing of user requests. They interact with the models of the application to retrieve data, perform calculations, and generate dynamic content. The view functions render templates once the processing is complete.

### **3.1.6 Templates for Dynamic Content**

Templates are HTML files with placeholders for dynamic content. These templates are populated with data from the view functions and are responsible for generating the final HTML that is sent to the user's browser.

### **3.1.7 Rendering the Response**

Django's template engine renders the dynamic content into HTML, creating a complete response. This response is then sent to the user's browser, which interprets and displays the content.

## **3.2 Providing a Service to the End User**

The Perfect Parking website is designed to provide an efficient and user-friendly service to end users seeking real-time information on parking space availability. Here's how the website serves the end user:

### **3.2.1 Real-Time Parking Space Availability**

The website allows users to initiate a search for available parking spaces within Limerick City and will show the user the distance in kilometres to the parking lots. Users are presented with a "Search Near Me" button, which triggers the geolocation retrieval and data query process.

### **3.2.2 Geolocation Retrieval and Query**

Upon clicking a "Search Near Me" button, the website retrieves the user's current geolocation through JavaScript. The obtained latitude and longitude coordinates are sent to the server to query carpark monitor data within the specified radius.

### **3.2.3 Probability Calculation and Presentation**

The server processes the carpark monitor data and calculates the vacancy rate percentage (%) to suggest the probability of parking space in a carpark. The results are dynamically presented in a table format on the website.

### **3.2.4 User Friendly Interface**

The website has an easy-to-use layout that allows users to readily interpret the results. The table provides useful details such as car-park names, addresses, operating hours, and the probability of parking availability.

### **3.2.5 Real Time Updates**

As users interact with the website, they are presented with a table of car parks based on their location to the carpark and how busy the carpark is. Real-time updates from client apps ensure that users can receive accurate and up-to-date information.

In summary, Django serves as the backbone of the Perfect Parking web application, facilitating the delivery of the website to users. The website offers an easy-to-use service, leveraging geolocation and dynamic data processing to provide real-time parking space availability information, thereby enhancing parking management and user convenience.

## **3.3 Real-Time Communication and Parking Data Updates: REST API Integration**

By leveraging RESTful APIs, Perfect Parking achieves efficient communication, data exchange, and collaboration among its distributed components. The RESTful API acts as a bridge, allowing the app to harness the web's capabilities for constructing scalable, interoperable, and robust distributed applications.

This subsection delves into the intricacies of integrating the car-park camera monitoring API within the Perfect Parking system. It explores how the REST architecture facilitates communication between client applications (parking monitors) and the server, leading to real-time updates of parking data. This analysis sheds light on the efficiency and reliability of the communication process, ensuring accurate parking availability information for end users.

### **3.3.1 Car-Park Camera Monitoring API and Communication via REST**

The Parking Monitor client uses the REST API to update the Perfect Parking system, enabling seamless communication between the client applications.

Each parking lot monitor is treated as a resource, identified by a unique URL. For instance, a parking lot monitor can be accessed via its own URL, and actions like updating its status or availability can be triggered using standard HTTP methods like POST, PUT, or DELETE

### **3.3.2 Client-Server Interaction**

The client applications, which represent the carpark monitors, initiate communication with the server using RESTful endpoints. These endpoints correspond to specific functionalities related to parking data updates. For instance:

- The client application can send a POST request to update the availability status of parking spaces as cars enter or exit.
- A PUT request can be utilized to update the probability of parking availability, calculated based on real-time data analysis.

### **3.3.3 Parking Data Updates**

During this communication, the clients update various parking data attributes on the server:

- **Availability Status:** As vehicles enter or leave parking spaces, the car-park monitors send data to the server, updating the status of each parking spot. The server interprets these updates and adjusts the availability status accordingly.
- **Probability Calculation:** The clients periodically send data related to occupancy, which the server uses to calculate the probability of available parking spaces. This dynamic probability is then communicated back to users in real-time through the website.

### **3.3.4 Real Time Updates and Reliability**

The RESTful communication between clients and the server ensures that the parking data is consistently updated, maintaining the accuracy and reliability of the information

presented to users. This real-time interaction enables users to access current parking availability probabilities, enhancing their decision-making process.

### **3.4 Automated Real-Time Parking Detection**

The Parking Monitor clients monitor the car parks registered in the database. This network of client applications will use advanced video processing capabilities, powered by the OpenCV library to autonomously identify parking spaces within the video footage to declare if the parking spots are available or occupied.

#### **3.4.1 Seamless Integration and Centralized Updates**

Real-time data updates and occupancy changes are automatically pushed to the server allowing for smooth integration with the Perfect Parking application. This centralized data flow ensures that the server receives real-time updates on parking space availability.

### **3.5 Utilizing Computer Vision in Perfect Parking**

Cameras deployed within the car park capture video in real-time. The Perfect Parking monitor application will use OpenCV to detect a car park spot's occupancy by analysing video frames. The monitor application continuously monitors the car park and provide up-to-date information on parking space availability to the server.

### **3.6 The Client Application**

The client application monitors the video feed of a parking lots across a city and provides live updates to the Perfect Parking website once a change in the car park is detected such as a spot becomes available or occupied.

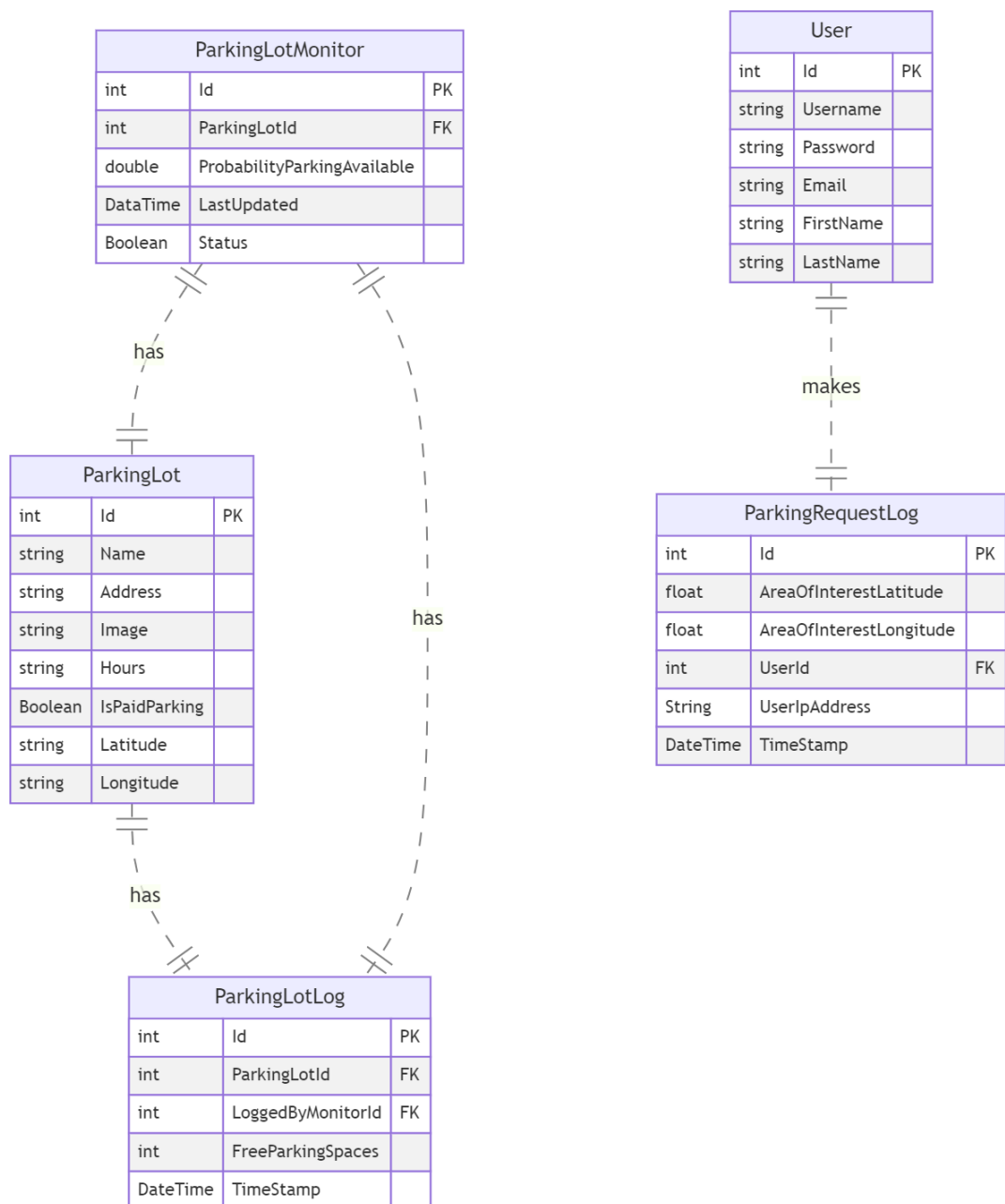
#### **3.6.1 Image Processing: Identifying a parking space's occupancy.**

To accomplish this goal, the client application will use image processing to identify if a car parking space is free or busy. If the client detects a change, it will send updated data to the server.

### **3.7 Application Database design**

The database for this project comprises three key tables: "User", "ParkingLotMonitor", and "ParkingLot". Login information for users of the parking application is kept in the

"User" table. The "ParkingLot" table keeps records of each parking lot's name, address, image, operating hours, and method of payment while the "ParkingLotMonitor" table keeps track of parking availability for monitored car parks. The "ParkingLotMonitor" table is linked to the "ParkingLot" table through the "ParkingLotId" column to enable the parking application to track parking availability at each location.



**Figure 6: Database Design**

### 3.8 System Actors

- Administrator: The administrator is responsible for managing the application. The administrator can add new parking locations to the database and can also remove parking locations from the database.
- User: The user is the person who will be using the application. The user can search for parking near a specific location.
- Guest: The guest is a person who is not logged in to the application. The guest can only search for parking near a specific location.
- Monitor Bot: A monitor is a bot that will be monitoring a car park. The monitor will be updating the status of the car park.

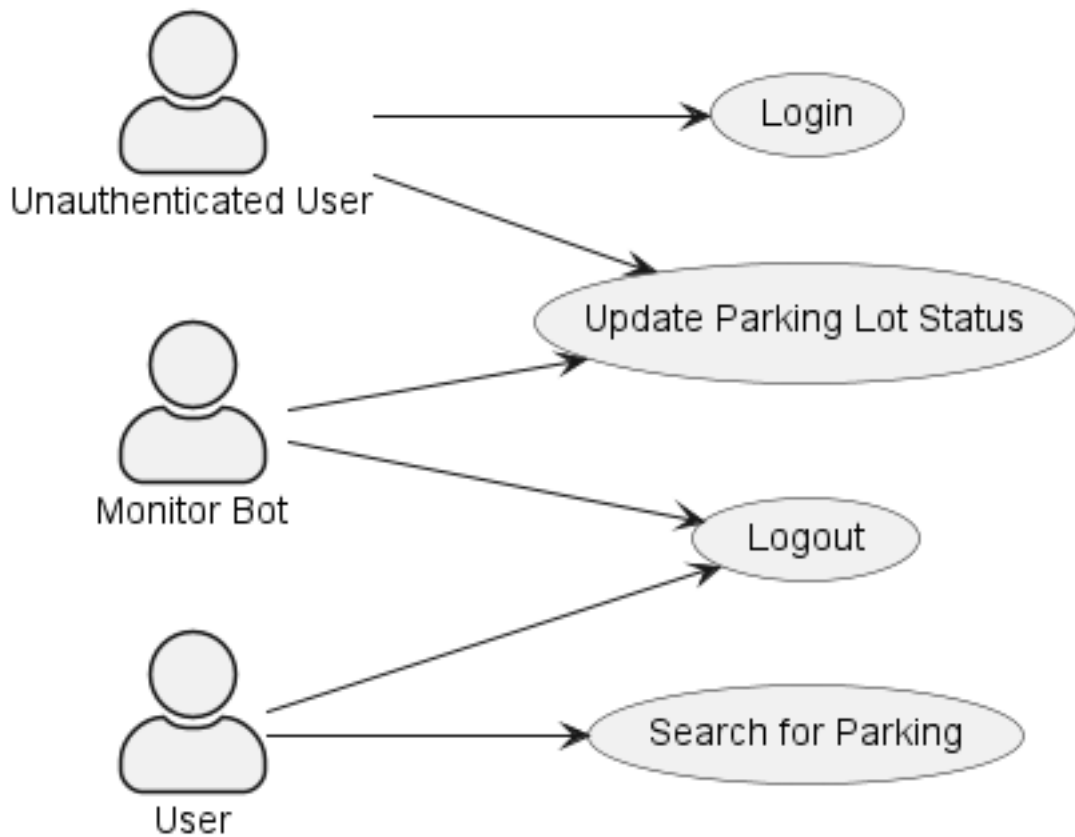


Figure 7: User Use Case Diagram

### **3.9 Use Case Descriptions**

#### **3.9.1 Use Case: Find Parking**

**Description:**

A user searches for parking near a specific location.

**Actors:**

- User

**Trigger Event:**

- A user wants to find parking near a specific location.

**Preconditions:**

- The user is logged in to the application.
- The website has permission to access the user's GPS location.

**Post conditions:**

- The user is shown a list of parking locations near the location they searched for.

**Main Flow:**

1. The user details the location they want to find parking nearby:
2. by searching for a specific address in the search bar.
3. by clicking on a location on the map.
4. by clicking on a location on the list of parking locations.
5. using the current location of the user.
6. The application shows the user a list of parking locations near the location they searched for.

**Alternative Flows:**

- If the user does not have permission to access their GPS location, the user can search for a specific address in the search bar.

### **3.9.2 Use Case: Register User**

#### **Description:**

A user registers for an account on the application.

#### **Actors:**

- Guest user

#### **Trigger Event:**

- A guest user wants to register for an account on the application.

#### **Preconditions:**

- The guest user is not logged in to the application.
- The guest user has not registered for an account on the application.
- The guest has a valid email address.

#### **Post conditions:**

- A user account is created for the guest user.

#### **Main Flow:**

1. The guest user clicks on the “Register” button.
2. The guest user enters their details into the registration form.
3. The guest user clicks on the “Register” button.
4. The application creates a user account for the guest user.
5. The guest logs in to the application.

#### **Alternative Flows:**

- If the guest user enters an email address that is already registered to an account, the application will display an error message.

### **Use Case: Login User:**

#### **Description:**

A user logs in to the application.



**Actors:**

- User

**Trigger Event:**

- A user wants to log in to the application.

**Preconditions:**

- The user is not logged in to the application.

**Post conditions:**

- The user is logged in to the application.

**Main Flow:**

1. The user clicks on the “Login” button.
2. The user enters their details into the login form.
3. The user clicks on the “Login” button.
4. The application logs the user into the application.

**Alternative Flows:**

- If the user enters an incorrect username and password, the application will display an error message.
- If the user enters a username that is not registered to an account, the application will display an error message.
- If the user account is disabled, the application will display an error message.

**3.9.3 Use Case: Update Parking Lot Status****Description:**

A monitor bot automatically updates the status of a parking lot.

**Actors:**

- Monitor

**Trigger Event:**

- A monitor updates the status of a parking lot.

**Preconditions:**

- The status of the parking lot is not updated

**Post conditions:**

- The status of the parking lot is updated.

**Main Flow:**

- The monitor sends a PUT request to the application REST API.
- The application updates the status of the parking lot.

**Alternative Flows:**

- If the monitor is not connected to the internet, the monitor will not be able to update the status of the parking lot.
- If the monitor API access token is invalid or has expired, the monitor will not be able to update the status of the parking lot.
- If the monitor sends an invalid request to the application REST API, the application will not update the status of the parking lot.
- If the parking lot does not exist in the database, the application will not update the status of the parking lot.

**3.9.4 Use Case: User changes password.****Description:**

- A user changes their password.

**Actors:**

- User

**Trigger Event:**

- A user wants or is required to change their password.

### Preconditions:

- The user is logged in to the application.

### Post conditions:

- The user's password is changed.

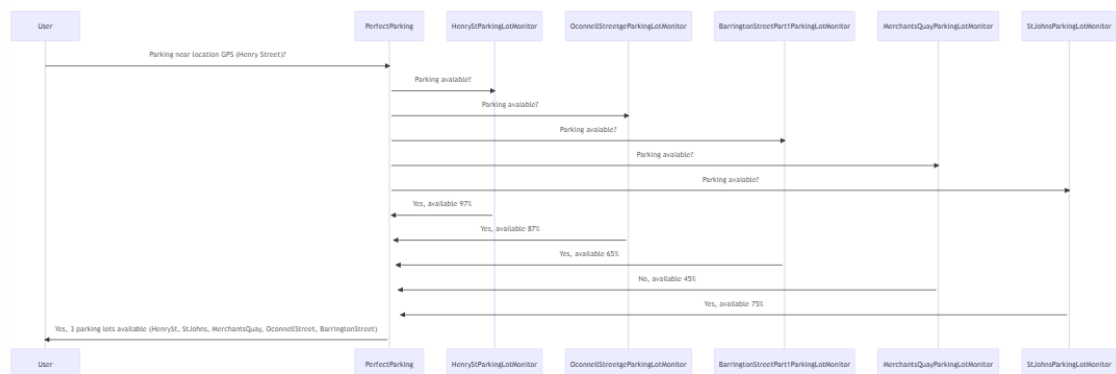
### Main Flow:

1. The user clicks on the “Change Password” button.
2. The user enters their details into the change password form.
3. The user clicks on the “Change Password” button.

### Alternative Flows:

- If the user enters an incorrect password, the application will display an error message.
- If the user enters a new password that does not meet the password requirements, the application will display an error message.

## 3.10 User Parking Sequence diagram



**Figure 8: User Parking Sequence Diagram**

This is the sequence diagram of the process where a user is searching for parking near their location using the application. The user asks the app if there is parking available near their GPS location on Henry Street.

To check if parking is available in each lot the software queries multiple parking lot monitors:

- Henry St Parking Lot Monitor
- O'Connell Street Parking Lot Monitor
- Merchants Quay Parking Lot Monitor
- Barrington Street Parking Lot Monitor
- St. Johns Parking Lot Monitor

The Henry St Parking Lot Monitor responds that parking is 97% available, O'Connell Street Parking Lot Monitor responds 87%, Merchants Quay Parking Lot Monitor 65%, Barrington Street Parking Lot Monitor 45%. And St. Johns Parking Lot Monitor 75%

Finally, the application then sends a response to the user indicating that there are 5 parking lots available near their location, with the names The Henry St, O'Connell Street, Merchants Quay, Barrington Street.

### **3.11 Technologies**

The following section explores a range of technologies and assesses their relevance for implementation within the project.

#### **3.11.1 Computer Vision: OpenCV**

Machine learning is a key component of the application as it will be used to identify if a car parking space is free or busy. The powerful library OpenCV provides a wide range of features for computer vision applications. It is widely used across many different industries, including robotics, driverless cars, medical imaging, and more.[refencer]

Image filtering, feature detection, object recognition, and tracking are just a few of the techniques that OpenCV offers to process and analyze visual data. (Boesch, 2023)

Several pre-trained cascading classifiers, including the well-known Haar cascades for face detection, are available in OpenCV for object detection. A parking monitor app powered by OpenCV will be used to detect if a parking spot has been filled or made empty, further improving the effectiveness of the system.

The machine learning algorithm will be trained using data collected from the sensors (video camera). The machine learning algorithm will then be used to identify if a car parking space is free or busy.

### **3.11.2 Anaconda**

Anaconda is a popular distribution of the Python programming language that is widely used for data science and scientific computing. It comes with a sizable number of pre-installed libraries and tools that are frequently used in these domains, including Jupyter Notebook, NumPy, Pandas, and Matplotlib. (Root, 2020)

Anaconda is designed to make it easy to set up and manage Python environments, which are essentially separate installations of Python with their own dependencies and libraries. When working on several projects with various requirements, this is especially helpful because it enables you to keep them separate from one another. (Root, 2020)

The Conda package manager, which lets you easily install, update, and manage additional software packages and libraries, is included with Anaconda in addition to Python and its libraries. When working with non-Python libraries that are necessary for your project, this can be helpful. (McKinney, 2022)

Utilising Anaconda has several benefits, one of which is how much easier it makes it to set up a Python environment for data research or scientific computing. It removes the need to individually install and configure each library, which can be a time-consuming and error-prone operation, by offering a pre-built distribution with many of the frequently used libraries already installed. (O Reilly, n.d.)

Overall, Anaconda is a robust and adaptable tool that is well-liked by those who work in data research and scientific computing. Researchers, developers, and data analysts all favour it because of how simple it is to use and the extensive library of tools that are already installed.

### **3.11.3 Django**

Django is a popular web development framework that is written in Python. It provides a set of tools and features that make it easy to build complex web applications quickly and efficiently. Django was created by Adrian Holovaty and Simon Willison in 2005, it features a vast collection of classes, libraries and modules that can be implemented in individual projects. With Django, you can create web applications that follow the Model-View-Controller (MVC) architecture, which helps to separate the different components of your application and make it easier to manage. Additionally, Django comes with a lot

of built-in functionality, including an ORM for database interactions, an admin interface for managing site content, and a templating system for rendering HTML pages. Overall, Django is a powerful and flexible framework that is well-suited for building all kinds of web applications. (Johnson, n.d.)

### 3.12 The Website

Perfect Parking is a web application that will allow users to search for parking in a city.

The application will allow users to search for parking near a specific location and will show the user data the nearest parking to their location.

The web application allows the user to access real time data on parking space availability in multiple carparks and shows the user the distance to each car park in the database. Here is how the system works.

1. **Geolocation Retrieval:** When a user accesses the web application, the system utilizes JavaScript to retrieve the user's current geolocation (latitude and longitude) through the browser's geolocation service this is triggered once the user clicks the "Search Near Me" button on the Parking lot page.
2. **Data Query and Processing:** Upon clicking the "Search Near Me" button, the web application queries a server-side backend or API to obtain carpark monitor data within the specified radius from the user's location.
3. **Vacancy Rate:** The received carpark monitor data is processed to calculate the rate of vacant parking spots to indicate the probability of parking space availability in each car park. The system determines the likelihood of finding an available parking spot based on the data obtained from the carpark monitors i.e., how many spots are free or occupied.
4. **User-Friendly Presentation:** The web application presents the results in a user-friendly format, allowing users to easily view and compare parking space availability in various carparks and shows the distance to each carpark so the user can verify which carpark is more convenient.

To conclude, the web application provides users with real-time information into parking space availability, allowing them to make decisions when searching for a parking lot. The

system enhances the parking management efficiency and user convenience, contributing to a smoother parking experience in urban areas.

### 3.13 Website Layout

#### 3.13.1 Parking Lots View

Clicking on the parking lots in the navigation bar will display a list of the parking lots available with some information about the different parking lots.

### Parking Lots

Parking Lots






Name	Address	Hours	Is Paid Parking	Latitude	Longitude	Image
<a href="#">Henry Street Left Strip</a>	Henry Street	Paid 09- 17, 2hr limit	True	52.663797090256100	-8.628752240173640	
<a href="#">O'Connell Street</a>	O'Connell Street, Limerick	09:00-17:00	True	52.662742547675100	-8.627915644146310	
<a href="#">Barington Street</a>	Barington Street, Limerick	09:00-17:00	True	52.658504565802500	-8.629324200623040	
<a href="#">Outside St. Johns</a>	New Rd, Limerick, Ireland	09:00-17:00	True	52.663888808578300	-8.616959972127420	
<a href="#">Merchants Quay</a>	Merchants Quay, Limerick	09:00-17:00	True	52.667553203563400	-8.624196369450390	

Figure 9: Parking Lots View

#### 3.13.2 Parking Lot View

Clicking on a parking lot name will display information on the selected parking lot such as a map and an image of the car park as well as other information about that specific parking lot.

## Henry Street Left Strip

### Details

#### Address

Henry Street

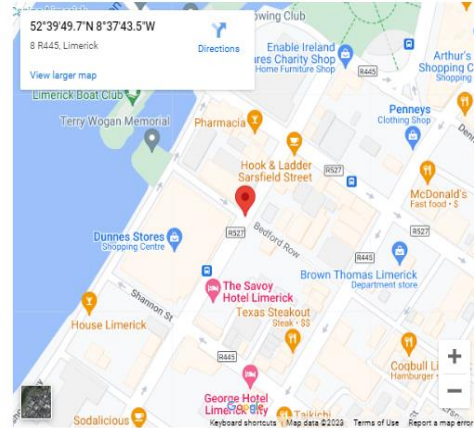
#### Hours

Paid 09- 17, 2hr limit

#### Total Spaces

1

### Map



### Images



Figure 10: Parking Lot View

### 3.13.3 Search View

Clicking on Search in the navigation bar will display a list of the parking lots available that are being monitored. The user will be asked to share the location by the browser. Information such as the probability of parking available is displayed which updates whenever there is a change in parking behaviour.

[Home](#) [Parking Lots](#) [Search](#) [Sign in](#) [Sign up](#)

## Search

[Search Near Me](#)

Parking Monitors

Name	Address	Hours	Is Paid Parking	Latitude	Longitude	Vacancy Rate (%)	Last Updated
<a href="#">Henry Street #4</a>	Henry Street	Paid 09- 17, 2hr limit	True	52.66380	-8.62875	100%	Aug. 20, 2023, 1:30 p.m.
<a href="#">O'Connell Street Monitor</a>	O'Connell Street, Limerick	09:00-17:00	True	52.66274	-8.62792	92%	Aug. 20, 2023, 1:44 p.m.
<a href="#">Merchants Quay Monitor</a>	Merchants Quay, Limerick	09:00-17:00	True	52.66755	-8.62420	50%	Aug. 20, 2023, 1:04 p.m.
<a href="#">Barington Street Monitor</a>	Barington Street, Limerick	09:00-17:00	True	52.65850	-8.62932	50%	Aug. 20, 2023, 1:23 p.m.
<a href="#">Outside St. Johns Monitor</a>	New Rd, Limerick, Ireland	09:00-17:00	True	52.66389	-8.61696	60%	April 27, 2023, 11:46 p.m.

[Privacy Policy](#) [GitHub](#) [Report](#) © Rhys Quilter 2021

Figure 11: Search View



### 3.13.4 Parking Lot Monitor

Clicking on the parking Monitor name it will display information on that specific parking monitor such as a location on the google map number of free spaces and the probability of the spaces available which updates every time there is a change and the website is refreshed. In the image below you can see that the probability has changed for this specific parking lot compared to Figure 5 above.

#### Henry Street #4

##### Parking

**Free Spaces**

5

**Total Spaces**

5

**Probability Parking available**

1.00

**Last Updated**

April 28, 2023, 8:34 a.m.

##### Details

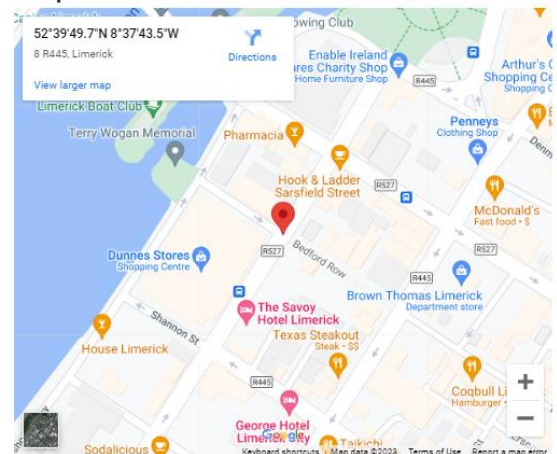
**Address**

Henry Street

**Hours**

Paid 09- 17, 2hr limit

##### Map



##### Image

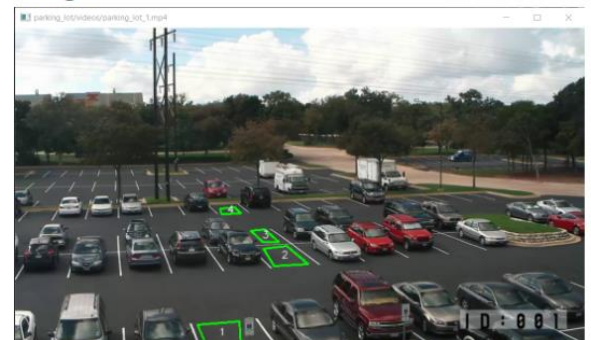
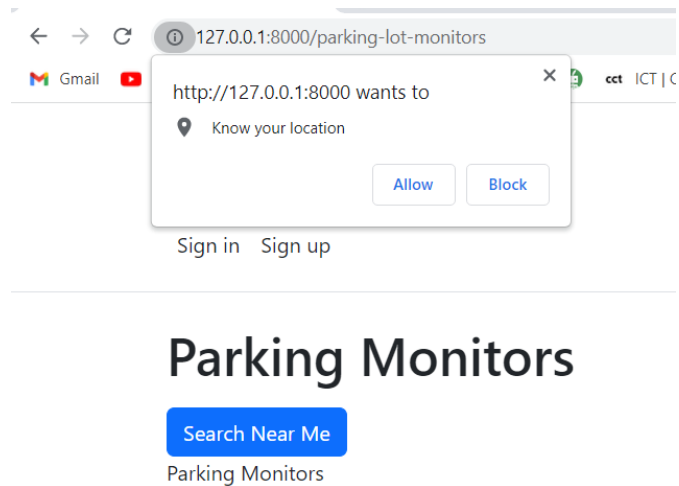


Figure 12: Parking Lot Monitor View

### 3.13.5 Know your Location Feature

When the parking lot monitors page is loaded for the first time the browser will ask you to share your location. This is done so the Search near me feature can be used.



**Figure 13: Location Sharing**

### 3.13.6 Search Near Me

When the search near me button is clicked it organises the parking lots that are monitored by how close they are to your location.

## Parking Monitors

[Search Near Me](#)  
Parking Monitors

Name	Address	Hours	Is Paid Parking	Latitude	Longitude	Available Parking Probability	Distance	Last Updated
<a href="#">Henry Street #4</a>	Henry Street	Paid 09- 17, 2hr limit	True	52.66379	-8.62875	0.83	0.75km	April 28, 2023, 12:28 a.m.
<a href="#">OConnell Street Monitor</a>	O'Connell Street, Limerick	09:00-17:00	True	52.66274	-8.62745	0.08	1.23km	April 28, 2023, 12:28 a.m.
<a href="#">Merchants Quay Monitor</a>	Merchants Quay, Limerick	09:00-17:00	True	52.66755	-8.62419	0.80	1.45km	April 28, 2023, 12:25 a.m.
<a href="#">Barington Street Monitor</a>	Barington Street, Limerick	09:00-17:00	True	52.65850	-8.62932	0.60	1.57km	April 28, 2023, 12:26 a.m.
<a href="#">Outside St. Johns Monitor</a>	New Rd, Limerick, Ireland	09:00-17:00	True	52.66388	-8.61695	0.60	2.23km	April 27, 2023, 11:46 p.m.

**Figure 14: Search Near Me Feature**

### 3.13.7 Login

This is the login for the application. Django comes with a built-in authentication system to help with assisting the user logging in.

[Sign in](#) [Sign up](#)

---

## Login

Username:

Password:

Login

**Figure 15: User Login**

### 3.13.8 Sign up (User Registration)

This is the sign-up page for the application. As with the login feature Django also has built-in authentication system to help with assisting the user during sign up.

---

## Register

Username:  Required. 150 characters or fewer. Letters, digits and @/./+/-/\_ only.

Password:

- Your password can't be too similar to your other personal information.
- Your password must contain at least 8 characters.
- Your password can't be a commonly used password.
- Your password can't be entirely numeric.

Password confirmation:  Enter the same password as before, for verification.

First Name:

Last Name:

Email:

Register

**Figure 16: User Registration**

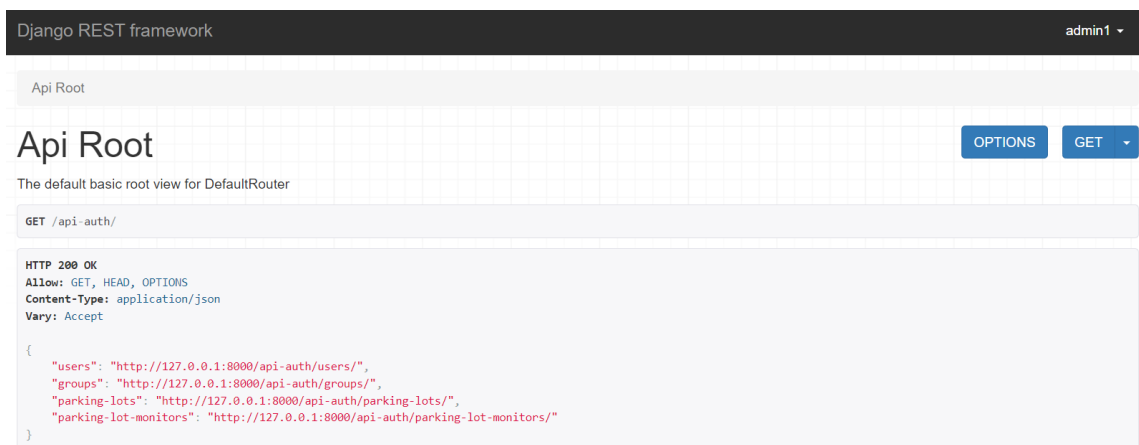
## 3.14 Rest API layout

Django Rest Framework is a powerful and flexible toolkit for building Web APIs. Some of the key features of interest in the Django Rest framework:

- The Web Browsable API that allows developers to explore and interact with the API using a web browser.
- Authentication policies
- Serialization: Django Rest Framework provides powerful serialization capabilities

### 3.14.1 API Root

This is the API Root page; you can see all the endpoints from the application.

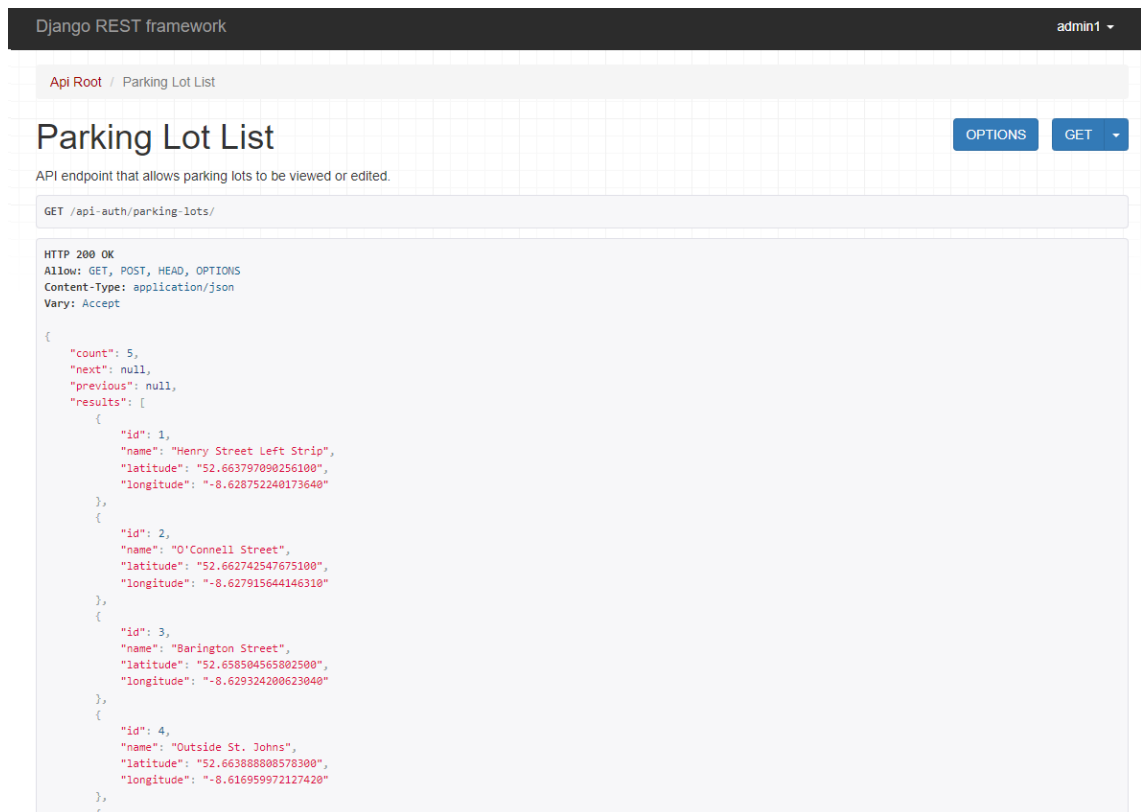


**Figure 17: Django Rest Framework API Root**

The two of the most interesting endpoints would be parking-lots and parking-lot-monitors:

### 3.14.2 Parking Lots

Parking Lots is a list of five endpoint, each with a unique identifier, latitude, and longitude coordinates. The "count" field indicates the total number of locations in the list, and the "next" and "previous" fields provide links to the next and previous pages of results if available.



**Figure 18: Parking Lot List**

### 3.14.3 Parking Lot Monitors

This is a list of five parking monitors, each with a unique identifier, latitude, longitude coordinates, and a probability of available parking at the corresponding location. The "count" field indicates the total number of monitors in the list, and the "next" and "previous" fields provide links to the next and previous pages of results if available.

## Parking Lot Monitor List

[OPTIONS](#)[GET](#)

API endpoint that allows parking lot monitors to be viewed or edited.

GET /api-auth/parking-lot-monitors/

```
HTTP 200 OK
Allow: GET, POST, HEAD, OPTIONS
Content-Type: application/json
Vary: Accept

{
  "count": 5,
  "next": null,
  "previous": null,
  "results": [
    {
      "id": 1,
      "name": "Henry Street #4",
      "latitude": "52.663797090256100",
      "longitude": "-8.628752240173640",
      "probabilityParkingAvailable": "0.83"
    },
    {
      "id": 2,
      "name": "OConnell Street Monitor",
      "latitude": "52.662742547675100",
      "longitude": "-8.627915644146310",
      "probabilityParkingAvailable": "0.08"
    },
    {
      "id": 3,
      "name": "Merchants Quay Monitor",
      "latitude": "52.667553203563400",
      "longitude": "-8.624196369450390",
      "probabilityParkingAvailable": "0.80"
    },
    {
      "id": 4,
      "name": "Barrington Street Monitor",
      "latitude": "52.658504565802500",
      "longitude": "-8.629324200623040",
      "probabilityParkingAvailable": "0.60"
    }
  ]
}
```

Figure 19: Parking Lot Monitor List

### 3.14.4 Parking Lot Monitor

This is one instance of the parking lot monitor from here you can do all the GET for Json and APIs. You can also see the id, name, latitude, longitude, and the probability of parking available for that for that certain parking lot.

## Parking Lot Monitor Instance

[OPTIONS](#)[GET](#)

API endpoint that allows parking lot monitors to be viewed or edited.

GET /api-auth/parking-lot-monitors/1/

```
HTTP 200 OK
Allow: GET, PUT, PATCH, DELETE, HEAD, OPTIONS
Content-Type: application/json
Vary: Accept

{
  "id": 1,
  "name": "Henry Street #4",
  "latitude": "52.663797090256100",
  "longitude": "-8.628752240173640",
  "probabilityParkingAvailable": "0.83"
}
```

Figure 20: Parking Monitor Instance

### 3.14.5 Parking Lot

This is one instance of the parking lot, from here you can see the id, name, latitude, and longitude.

Api Root / Parking Lot List / Parking Lot Instance

# Parking Lot Instance

OPTIONSGET

API endpoint that allows parking lots to be viewed or edited.

GET /api-auth/parking-lots/1/

HTTP 200 OK

Allow: GET, PUT, PATCH, DELETE, HEAD, OPTIONS

Content-Type: application/json

Vary: Accept

```
{
  "id": 1,
  "name": "Henry Street Left Strip",
  "latitude": "52.663797898256188",
  "longitude": "-8.628752240173640"
}
```

**Figure 21: Parking lot Instance**

## Chapter 4 Implementation

This chapter details the implementation of the server/website and client app.

### 4.1 Project Management

During the time doing the final year project weekly supervisor meetings were held for feedback and to track the milestones to ensure that the project was kept on track.

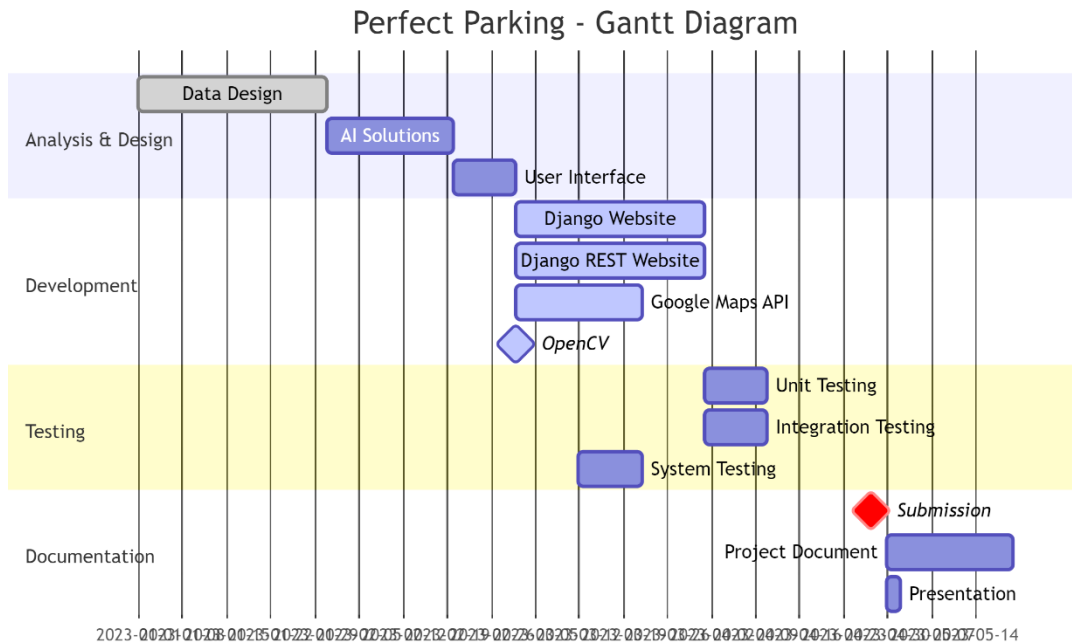


Figure 22: Gantt Chart

#### 4.1.1 Tools Used

Python: An interpreted, object-oriented, high-level programming language with dynamic semantics.

#### Libraries and Frameworks:

- Django web framework, which allowed for rapid development of the application website/server and easy maintenance.
- OpenCV, an open-source computer vision and machine learning software library, which was used for image processing and analysis.



## Tools:

- VS Code: A lightweight code editor with support for many languages and debugging.
- Anaconda: a package management, and deployment tool to install and manage required libraries and dependencies.
- Microsoft Word: used to write the documentation.
- Canva: a graphic design platform used to create the project poster.
- Mermaid.live: used to create markdown diagrams.

By utilizing these tools, languages, and frameworks, the project was completed more efficiently, with greater accuracy and precision.

### 4.1.2 How to Install Anaconda

- Download the Appropriate Anaconda installer from the [Anaconda Website](#)
- Open VS Code and open a new terminal window by selecting "Terminal" from the top menu and then selecting "New Terminal".
- Navigate to the directory where you downloaded the Anaconda installer using the command **cd <directory>**.
- For example, if you downloaded the Anaconda installer for Windows and saved it in your Downloads folder, you would type the following command in the terminal:

```
cd Downloads
```

- Run the Anaconda installer by typing the command **bash <Anaconda installer filename>** in the terminal, where **<Anaconda installer filename>** is the name of the Anaconda installer file you downloaded.

```
bash Anaconda3-2021.05--x86_64.sh
```

- Follow the instructions in the Anaconda installer to complete the installation process.
- Once the installation is complete, you can use Anaconda in the VS Code terminal by activating the Anaconda environment with the command **conda activate**. You can then use the various Anaconda packages and tools in the terminal as needed.

- For example, if you want to use the Pandas library in your Python script, you can first activate the Anaconda environment by typing the following command in the terminal:

```
1.      conda activate
```

- Then, you can import the Pandas library in your Python script using the following line of code:

```
1.      import pandas as pd
```

- This will allow you to use the various functions and methods provided by the Pandas library in your project.

### 4.1.3 Source Control and versioning

For source control and versioning, GitHub was utilized to manage the codebase for the project. GitHub was chosen because of prior experience using it during other studies, and it provided a reliable platform for version control and collaboration with the supervisor. It allows for easy upkeep and to keep track of changes made during each week and easily roll back to previous versions if needed.

Additionally, the supervisor was added as a collaborator on the repository, allowing him to view the progress and provide feedback on the code and documents. This facilitated effective communication and ensured that the project was aligned with the objectives.

One significant advantage of using GitHub was that it provided a safe and secure backup of the code. In the event of file corruption, it would be possible to pull down the last push request and continue the work without losing progress.

## 4.2 To install and run the project.

Clone the repository

```
git clone https://github.com/rhysquilter/perfect-parking-fyp.git
```

There are three applications in this project: the website server, a client parking monitor and a mock client parking monitor.

### 4.2.1 Run Website Server

1. Install the dependencies

- With Pip (To fast execute, run on Windows pip-install.bat)

```
pip install -r requirements.txt
```

- With conda (To fast execute, run on Windows conda-install.bat)

```
conda install --file requirements.txt
```

2. Run the app Open VS Code and run the app by selecting the Run and Debug (Ctrl + Shift + D) tab and selecting Server - Django from the dropdown menu. Then click the green play button to run the app, or Open a terminal and run the following command:

```
python manage.py runserver
```

3. Open the app in your browser <http://localhost:8000>

#### 4.2.1.1 Built-in User Accounts

Built-in user accounts are:

Username	Password	Role	Note
john	John123456	admin	Built in admin account
admin1	letmein	admin	Built in admin account
parkingMonitor	Letmein1\$	user	Account for parking monitor apps

#### 4.2.2 Run the Client Parking Monitor

For the client parking monitor to run, the server must be running.

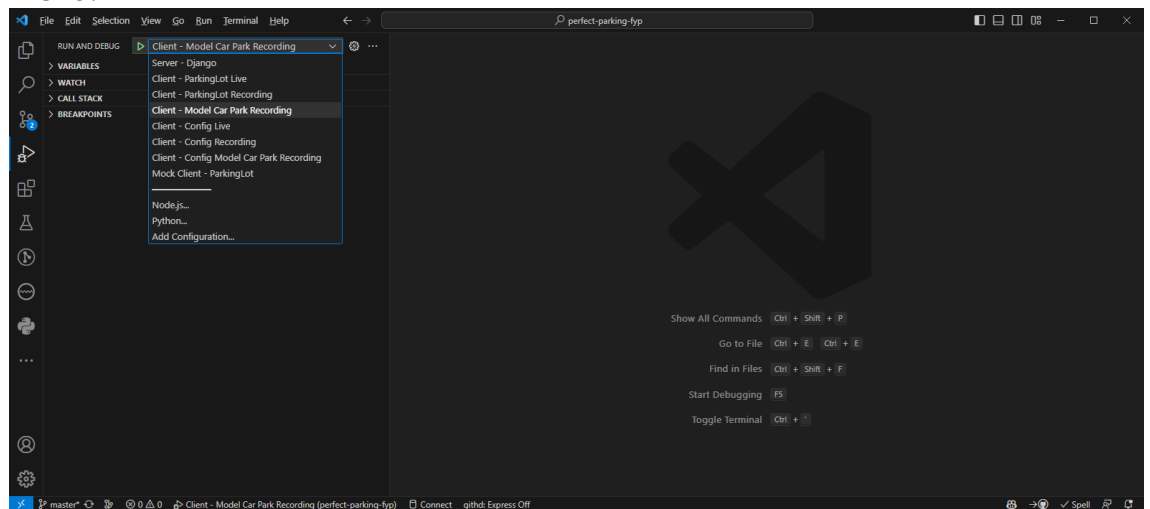
1. Install dependencies

- With Pip (To fast execute, run on Windows [PerfectParkingClient/pip-install.bat](#))

```
pip install -r PerfectParkingClient/requirements.txt
```

- With conda (To fast execute, run on Windows [PerfectParkingClient/conda-install.bat](#))
- `conda install --file PerfectParkingClient/requirements.txt`

2. The server must be running
3. Open VS Code and run the app by selecting the Run and Debug (Ctrl + Shift + D) tab and select one of the client configurations e.g., Client - Config Model Car Park Recording from the dropdown menu.



Then click the green play button to run the app, or Open a terminal and run the following command:

1. `python PerfectParkingClient/main.py '--image' 'PerfectParkingClient/images/live-sample-3.png' '--data' 'PerfectParkingClient/data/coordinates-live-sample.yml' '--video' 'PerfectParkingClient/videos/live-sample-3.mp4' '--start-frame' '400'`
- 2.

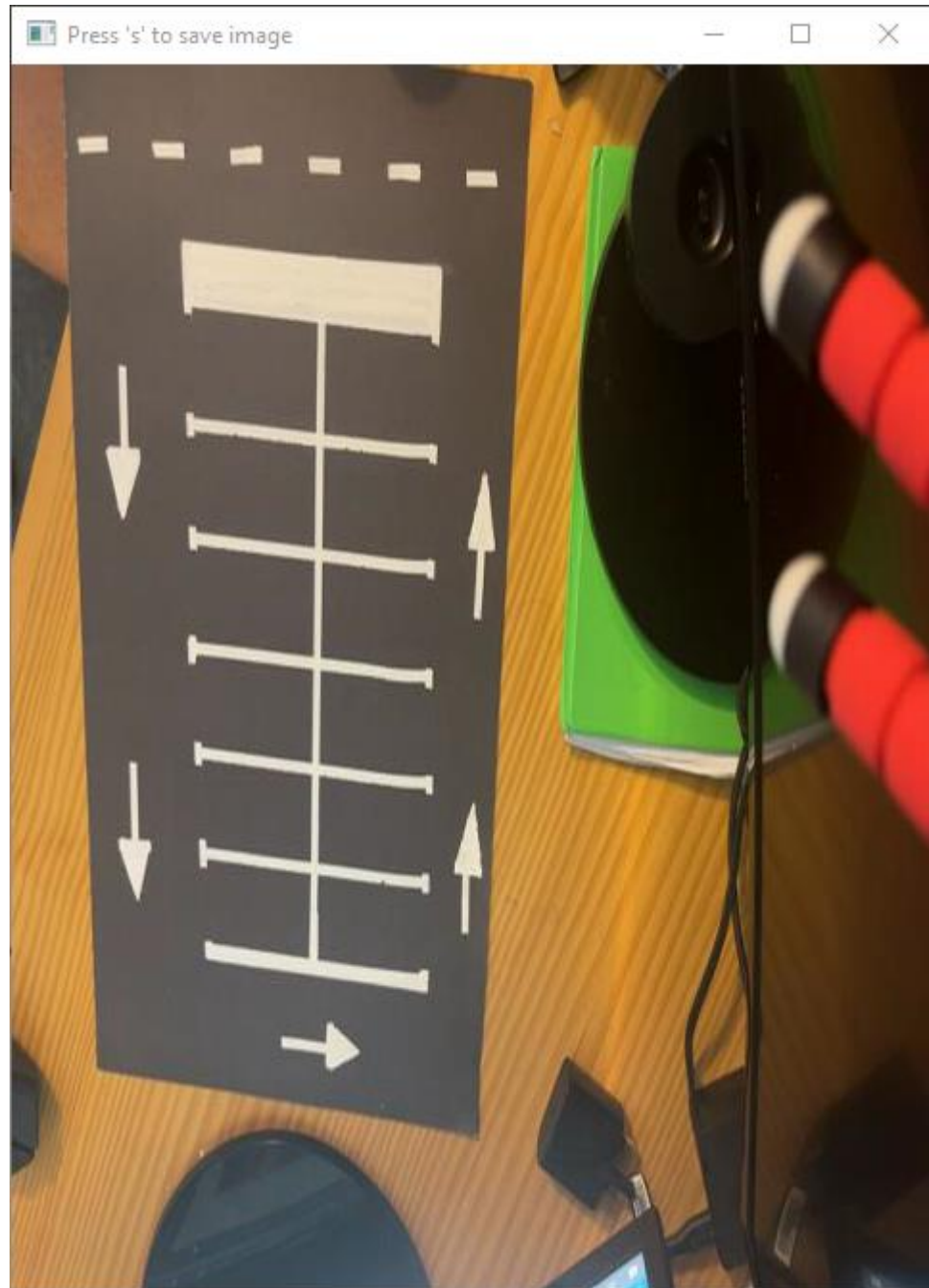
4. Follow the instructions in the Window Title Bar.

#### 4.2.2.1 Sample Live Client Run

The live client configuration is used to get video from a phone app such as DroidCam. Note that the IP address of 192.168.188.106:4747 for Client Live configurations may need to be changed depending on your Network Configuration. This can be changed in the [launch.json](#) file. After running the app, the following steps will be shown in the Window Title Bar.

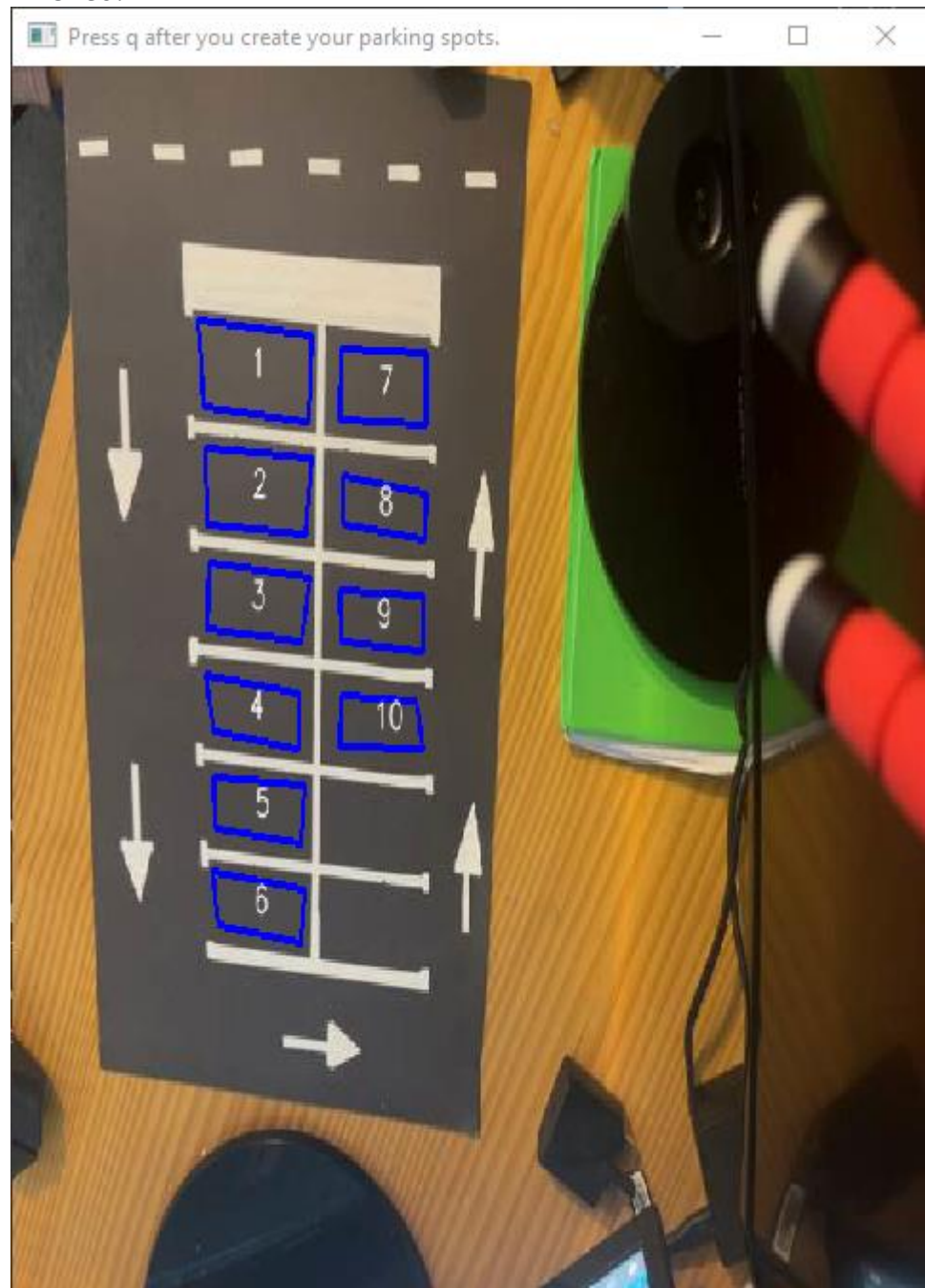
1. Press `s` to select an image from the video feed to make your parking spaces on. Ideally, this should be an image of the car park with no cars in

it.

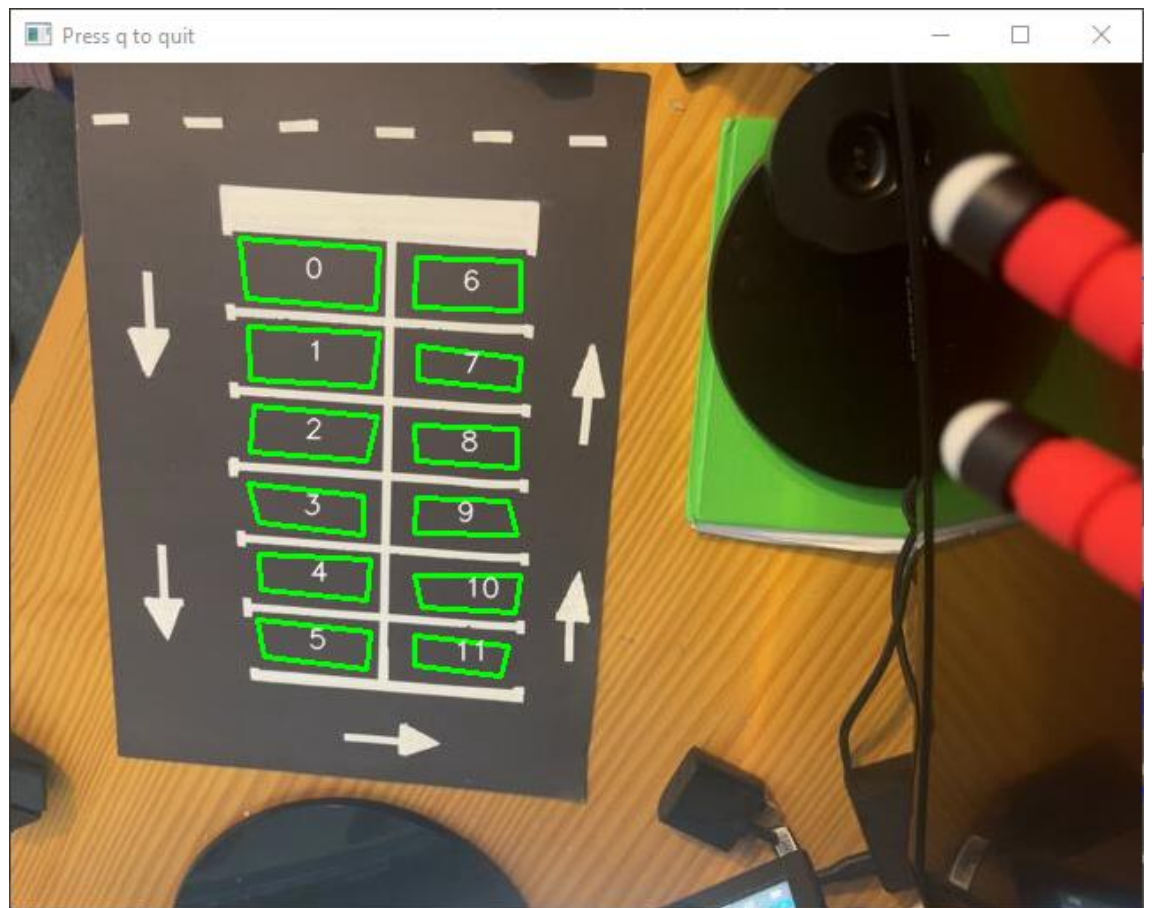


2. Mark the car parking spaces by clicking on the four corners of each parking space. Try not to include any white lines in the parking space. Press q when

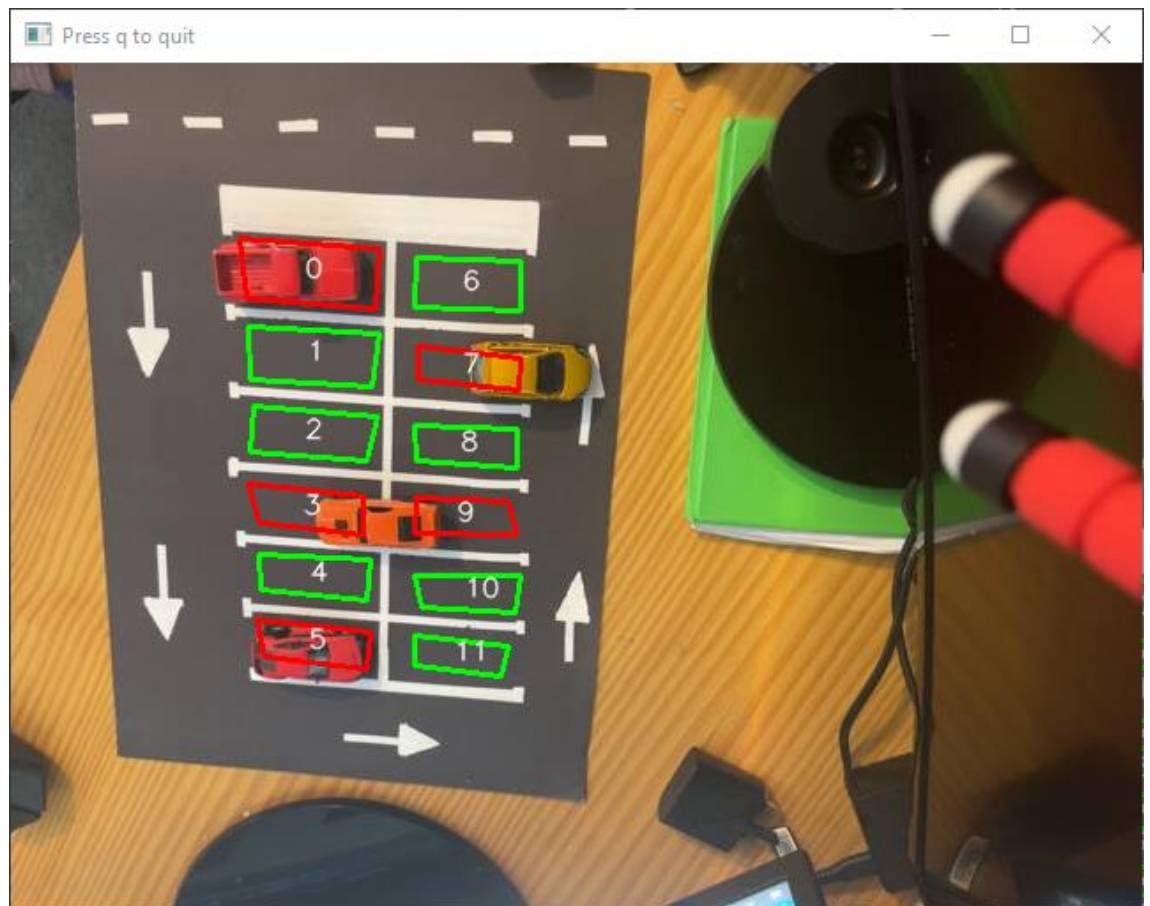
finished.



3. Step 3: Watch the video feed and see the parking spaces being detected. Press `q` to quit when you are finished.







### 4.2.3 Mock Client Parking Monitor

The mock client parking monitor is used to test the server without the need for a client parking monitor. It loads the configs for five parking monitors and sends randomly generated data to the server at random times.

For the mock client parking monitor to run, the server must be running. The mock client dependencies are the same as the client parking monitor.

To run the mock client parking monitor, Open VS Code and run the app by selecting the Run and Debug (Ctrl + Shift + D) tab and selecting Mock Client - ParkingLot from the dropdown menu. Then click the green play button to run the app, or

Open a terminal and run the following command:

```
python PerfectParkingClient/mock.py
```



You can stop the mock client by pressing `q` at any time.

### **4.3 How Object Recognition works in Perfect Parking clients**

The Perfect Parking Server receives parking status data from client applications. A client monitor app is responsible for processing video and determining if parking is available. A proof-of-concept project by Olga Rocheeva was sourced on GitHub and built upon to work with Perfect Parking (Rocheeva, 2018).

### **4.4 Implementation of Perfect Parking Web Application**

The Perfect Parking web application is developed using the Django framework, enabling the creation of a robust parking management system. This application acts as a hub for users to access real-time parking information and monitor the availability of parking spaces in multiple parking lots.

#### **4.4.1 Model Based Data Management**

The data structure of the application is designed using Django's model-driven architecture. The application defines two key classes in the `models.py` file: `ParkingLot` and `ParkingLotMonitor`. The `ParkingLot` class contains information about parking lots such as their name, address, hours of operation, and available parking spaces. The `ParkingLotMonitor` class, on the other hand, contains information about the parking lot monitors, such as parking availability probability, and status.

```

1. from geopy.distance import geodesic
2. from django.db import models
3.
4. class ParkingLotMonitor(models.Model):
5.     id = models.AutoField(primary_key=True)
6.     parkingLot = models.ForeignKey(ParkingLot, on_delete=models.CASCADE)
7.     name = models.CharField(max_length=100, unique=True)
8.     latitude = models.DecimalField(max_digits=17, decimal_places=15)
9.     longitude = models.DecimalField(max_digits=17, decimal_places=15)
10.    probabilityParkingAvailable = models.DecimalField(max_digits=5, decimal_places=2)
11.    free_parking_spaces = models.IntegerField(default=0)
12.    dateTimeLastUpdated = models.DateTimeField(auto_now=True)
13.    status = models.BooleanField(default=True)
14.    image = models.ImageField(upload_to="images/parking-lot-monitor/", blank=True)
15.
16.    def get_distance_from_lat_lang(self, latitude, longitude) -> float:
17.        point: tuple = (latitude, longitude)
18.        return self.get_distance_from_point(point)
19.
20.    def get_distance_from_point(self, user_point: tuple) -> float:
21.        return round(geodesic(self.get_gps_point(), user_point).km, 2)
22.
23.    #...

```

#### Parking Lot Monitoring Model

### 4.4.2 User Friendly Interface

The Perfect Parking application's front-end was designed to be simple and easy to use. Users can easily interact with the web application by web browsers. When users use the app, they are met with an interactive interface that displays the most recent parking information.

### 4.4.3 Views and Templates

To successfully present data to users, the application takes advantage of Django's views and templates. Views, which are implemented as Python functions, handle incoming HTTP requests, and generate appropriate responses. The user interface is rendered using templates, which are structured as HTML files with placeholders for dynamic data.

The views of the application retrieve data from the models and send it to templates for rendering. For example, a view could query the database to get information about available parking lots and the monitoring devices that go with them. This data is then organized and sent to templates to generate user-readable output.

### 4.4.4 Real time Data Integration

The application's real-time data update approach is one of its most notable features. This is done by frequent interactions with parking lot monitors' REST APIs. These APIs

provide the most recent parking availability data. This data is processed by the application's views, which dynamically update the user interface to reflect changes in parking status.

#### **4.4.5 User Query Tracking**

The application also tracks user queries and interactions. When users search for parking lots based on specific geographic coordinates, these queries are logged and stored in the database. This feature not only enhances user experience but also provides valuable insights into user preferences and trends.

#### **4.4.6 Implementation Highlights**

The Perfect Parking web application implementation demonstrates the use of Django's models for structured data management. The user interface is simple to use, and templates dynamically render real-time data. The application keeps users informed about parking space availability by interacting with REST APIs. Furthermore, the system logs user inquiries, providing a detailed view of user behaviour.

### **4.5 Implementation of Parking Monitoring Client App**

This section provides an in-depth insight into the practical implementation of the client application responsible for carpark monitoring. The client application's core functionality is to detect parking space availability and communicate this data to the central server using a REST API.

The client application, known as `PerfectParkingClient`, encapsulates crucial classes that enable the monitoring and reporting of parking spaces:

- `ParkingMonitorData`: This class stores key parking monitor data. It receives configuration data from a file and stores attributes such as the Id, name, position, parking spots, and authentication credentials of the monitor.
- `RestApiUtility`: This class is essential for providing updates to the central server over the REST API. It creates and transmits PUT requests containing updated parking monitor data, such as the number of available parking spaces and the probability of parking availability.

- **ParkingSpot:** This class represents individual parking spaces, recording their coordinates as well as their occupancy state. Based on visual processing, it determines whether a parking space is occupied.
- **MotionDetector:** This class manages the monitoring of parking spaces. It detects the occupancy state of each parking spot in a video feed using image processing algorithms. When the parking occupancy changes, the RestApiUtility sends the information to the server.

The process involves capturing a video feed, processing frames to determine parking spot occupancy, and then reporting the number of available spaces and parking availability probability to the server through the REST API.

In the class RestApiUtility the communication with the server utilizes the method `update_server_parking_monitor_data`. This method constructs a JSON object with the relevant parking monitor data, encapsulates it within a PUT request, and sends it to the server. The server responds with the outcome of the update.

Real-time parking space monitoring and reporting are made possible by seamlessly integrating the client application with the server via REST communication, resulting in an accurate and up-to-date parking availability display for end users.

#### **4.5.1 A quick note on the original source code quality**

The client app was inspired by (Rocheeva, 2018). The original code seemed to lack a consistent intuitive naming convention coupled with a severe lack of much needed comments. It was designed with array based data structures and little Object Orientated (OO) design. The primary functionality was in the `detectmotion()` function and had a Cognitive Complexity value of 33 well above the allowed 15 according to sonarlint (Sonar Rules, n.d.) and refactoring the function was required.

```

1. class MotionDetector:
2.     LAPLACIAN = 1.4
3.     DETECT_DELAY = 1
4.
5.     def __init__(self, video, coordinates, start_frame):
6.         self.video = video
7.         self.coordinates_data = coordinates
8.         self.start_frame = start_frame
9.         self.contours = []
10.        self.bounds = []
11.        self.mask = []
12.
13.    def detect_motion(self):
14.        capture = open_cv.VideoCapture(self.video)
15.        capture.set(open_cv.CAP_PROP_POS_FRAMES, self.start_frame)
16.
17.        coordinates_data = self.coordinates_data
18.        logging.debug("coordinates data: %s", coordinates_data)
19.
20.        for p in coordinates_data:
21.            coordinates = self._coordinates(p)
22.            logging.debug("coordinates: %s", coordinates)
23.
24.            rect = open_cv.boundingRect(coordinates)
25.            logging.debug("rect: %s", rect)
26.
27.            new_coordinates = coordinates.copy()
28.            new_coordinates[:, 0] = coordinates[:, 0] - rect[0]
29.            new_coordinates[:, 1] = coordinates[:, 1] - rect[1]
30.            logging.debug("new_coordinates: %s", new_coordinates)
31.
32.            self.contours.append(coordinates)
33.            self.bounds.append(rect)
34.
35.            mask = open_cv.drawContours(
36.                np.zeros((rect[3], rect[2]), dtype=np.uint8),
37.                [new_coordinates],
38.                contourIdx=-1,
39.                color=255,
40.                thickness=-1,
41.                lineType=open_cv.LINE_8)
42.
43.            mask = mask == 255
44.            self.mask.append(mask)
45.            logging.debug("mask: %s", self.mask)
46.
47.        statuses = [False] * len(coordinates_data)
48.        times = [None] * len(coordinates_data)
49.
50.        while capture.isOpened():
51.            result, frame = capture.read()
52.            if frame is None:
53.                break

```

```

55.         if not result:
56.             raise CaptureReadError("Error reading video capture on frame %s" %
str(frame))
57.
58.         blurred = open_cv.GaussianBlur(frame.copy(), (5, 5), 3)
59.         grayed = open_cv.cvtColor(blurred, open_cv.COLOR_BGR2GRAY)
60.         new_frame = frame.copy()
61.         logging.debug("new_frame: %s", new_frame)
62.
63.         position_in_seconds = capture.get(open_cv.CAP_PROP_POS_MSEC) / 1000.0
64.
65.         for index, c in enumerate(coordinates_data):
66.             status = self.__apply(grayed, index, c)
67.
68.             if times[index] is not None and self.same_status(statuses, index, status):
69.                 times[index] = None
70.                 continue
71.
72.             if times[index] is not None and self.status_changed(statuses, index,
status):
73.                 if position_in_seconds - times[index] >= MotionDetector.DETECT_DELAY:
74.                     statuses[index] = status
75.                     times[index] = None
76.                     continue
77.
78.             if times[index] is None and self.status_changed(statuses, index, status):
79.                 times[index] = position_in_seconds
80.
81.         for index, p in enumerate(coordinates_data):
82.             coordinates = self._coordinates(p)
83.
84.             color = COLOR_GREEN if statuses[index] else COLOR_BLUE
85.             draw_contours(new_frame, coordinates, str(p["id"] + 1), COLOR_WHITE,
color)
86.
87.         open_cv.imshow(str(self.video), new_frame)
88.         k = open_cv.waitKey(1)
89.         if k == ord("q"):
90.             break
91.         capture.release()
92.         open_cv.destroyAllWindows()

```

Original detect\_motion by (Rocheeva, 2018) with Code Complexity of 33

## 4.5.2 Refactoring/Modifications

The code was refactored to introduce Python naming conventions, OO designs and classes and reduce function Cognitive Complexity

### 4.5.2.1 Parking Spot

```
1. class ParkingSpot:
2.     def __init__(self, coordinates: ndarray, parking_spot_id: int):
3.         self.coordinates: ndarray = coordinates
4.         self.is_occupied: bool = False
5.         self.parking_spot_id: int = parking_spot_id
6.         self.rect = boundingRect(self.coordinates)
7.         self.mask:list = self.create_contours_mask()
8.
9.     def create_contours_mask(self)->list:
10.         # ...
11.
12.     def determine_and_mark_occupancy_from_image(self, blurred_grayed_image:Mat):
13.         #...
14.
15.     def has_changed(self, is_occupied: bool) -> bool:
16.         return self.is_occupied != is_occupied
17.
18.     def has_not_changed(self, is_occupied: bool) -> bool:
19.         return self.is_occupied == is_occupied
20.
```

**Introduced OO design of ParkingSpot class.**

### 4.5.2.2 How a parking space status is determined

The code can identify when a car is present in a location by analyzing the average pixel intensity within the marked area and comparing it to a threshold value. A location is regarded as available if the average intensity is below the threshold value and seen as occupied if it is above the threshold.

The function `determine_and_mark_occupancy_from_image()` in `motion_detector.py` is responsible for determining if the status of parking spaces. The Laplacian operator helps enhance and emphasize edges and intensity variations in an image aiding in detecting features and boundaries.

```
1. def determine_and_mark_occupancy_from_image(self, blurred_grayed_image:Mat):
2.     x, y, width, height = self.rect
3.
4.     parking_spot_image:Mat = blurred_grayed_image[y:(y + height), x:(x + width)]
5.     laplacian = cv2.Laplacian(parking_spot_image, cv2.CV_64F)
6.
7.     laplacian_mean: float = numpy.mean(numpy.abs(laplacian * self.mask))
8.     self.is_occupied = laplacian_mean < MotionDetector.LAPLACIAN_UPPER_LIMIT
```

**Determining and Marking Parking Spot Occupancy from Processed Image**

In the context of the code, it assists in determining the occupancy of a parking space by analysing the intensity changes within a region of interest. The contours, on the other hand, help create a mask that represents the detected object's boundary, which can be useful for further analysis and segmentation.

```

1.     def create_contours_mask(self)->list:
2.         new_coordinates = self.coordinates.copy()
3.
4.         x, y, width, height = self.rect
5.         new_coordinates[:, 0] = self.coordinates[:, 0] - x
6.         new_coordinates[:, 1] = self.coordinates[:, 1] - y
7.
8.         contours:list = drawContours(
9.             numpy.zeros((height, width), dtype=numpy.uint8),
10.            [new_coordinates],
11.            contourIdx=-1,
12.            color=255,
13.            thickness=-1,
14.            lineType=cv2.LINE_8)
15.
16.         contours = contours == 255
17.         return contours

```

#### Generating Contours Mask for Parking Spot Region

#### 4.5.2.3 Analysing the video

The client app pre-processes an image by first blurring and then grey scaling the image to reduce noise to improve the accuracy of parking spot occupancy detection.

```

1. def detect_motion(self)->bool:
2.     #...
3.     free_spaces: int = 0
4.     while True:
5.         #...
6.         is_open, video_frame = video_capture.read()
7.         #...
8.         blurred = GaussianBlur(video_frame.copy(), (5, 5), 3)
9.         grayed_image: Mat = cvtColor(blurred, cv2.COLOR_BGR2GRAY)
10.
11.         for parking_spot in self.parking_spots:
12.             parking_spot.determine_and_mark_occupancy_from_image(grayed_image)
13.

```

#### PerfectParking/ PerfectParkingClient/motion\_detector.py

The programme overlays the designated parking spaces onto the video, initialising them as available or occupied depending on the presence of cars.

```

1. def display_image(self, video_frame:Mat):
2.     for parking_spot in self.parking_spots:
3.         color:tuple = COLOR_GREEN if parking_spot.is_occupied else COLOR_BLUE
4.         parking_spot_text:str = str(parking_spot.parking_spot_id)
5.         draw_contours(video_frame, parking_spot.coordinates, parking_spot_text,
6.            COLOR_WHITE, color)
7.         imshow(str(self.video), video_frame)

```

#### Function to Display Parking Spot Occupancy on Video Frame



#### 4.5.2.4 Add/Using REST API

##### Changes

- Config files to hold application configuration data and sensitive information.

```
1. [ParkingLotMonitor]
2. Id=1
3. Name=Henry Street
4. Latitude=52.663797090256100
5. Longitude=-8.628752240173640
6.
7. [App]
8. Token=YOUR_TOKEN
9. Username=YOUR_USERNAME
10. Password=YOUR_PASSWORD
11. ServerUrl=http://127.0.0.1:8000/api-auth/parking-lot-monitors/
```

sample config.ini

- A Plain Old Common Object (POCO) ParkingMonitorData class was added to hold data about a parking lot monitor i.e., A Parking Lot Monitor's Id, Name, Latitude, Longitude, Number of Parking Spaces, and the App's Authentication Token, Username, Password, and the Server's URL. The ParkingMonitorData class uses Config Parser to load the Parking Lot Monitor's data from config files keeping best practices of separating sensitive data from the source code.

```
1. from configparser import ConfigParser
2.
3. class ParkingMonitorData:
4.
5.     def __init__(self, config_filepath: str =
"PerfectParkingClient/config.ini"):
6.         config_parser: ConfigParser = ConfigParser()
7.
8.         config_parser.read(config_filepath)
9.
10.        self.id = config_parser["ParkingLotMonitor"]["Id"]
11.        self.name = config_parser["ParkingLotMonitor"]["Name"]
12.        self.latitude = config_parser["ParkingLotMonitor"]["Latitude"]
13.        self.longitude = config_parser["ParkingLotMonitor"]["Longitude"]
14.        self.parking_spaces =
config_parser["ParkingLotMonitor"]["ParkingSpaces"]
15.
16.        self.app_token = config_parser["App"]["Token"]
17.        self.app_username = config_parser["App"]["Username"]
18.        self.app_password = config_parser["App"]["Password"]
19.        self.server_url = config_parser["App"]["ServerUrl"]
```

perfectparking.py – class ParkingMonitorData

- For educational and assessment purposes only the config files (containing usernames and passwords) have been uploaded to GitHub. The gitignore file was modified to allow sensitive .ini files to be posted to GitHub.

```
1. #*.ini
```

**.gitignore**

- The constructor of MotionDetector was modified to inject in ParkingMonitorData.

```
1. class MotionDetector:
2.     LAPLACIAN_UPPER_LIMIT = 1.4
3.
4.     def __init__(self, video, parking_spots_json_dict, start_frame,
parking_monitor_data: ParkingMonitorData):
5.         self.video = video
6.         self.parking_spots: list = [
7.             ParkingSpot(numpy.array(parking_spot_json_dict["coordinates"]),
parking_spot_json_dict["id"])
8.             for parking_spot_json_dict in parking_spots_json_dict
9.         ]
10.
11.         self.start_frame = start_frame
12.         self.parking_monitor_data = parking_monitor_data
```

**motion\_detector.py - class MotionDetector**

- A “method on\_free\_parking\_spaces\_changed” was added to handle the event of a change to the number of available parking spaces.

```
1. def on_free_parking_spaces_changed(self, spaces_in_frame: int,
free_spaces_in_frame: float) -> None:
2.     probability_parking_available = free_spaces_in_frame/spaces_in_frame
3.     RestApiUtility.update_server_parking_monitor_data(
4.         self.parking_monitor_data, free_spaces_in_frame,
probability_parking_available)
```

**motion\_detector.py - class MotionDetector - new method**

- A Rest API class was added to send updates to the server.

```

1. from requests.auth import HTTPBasicAuth
2. import requests
3.
4. #...
5. class RestApiUtility:
6.
7.     @staticmethod
8.     def update_server_parking_monitor_data(parking_monitor_data:
ParkingMonitorData, free_spaces_in_frame: float,
9.                                           probability_parking_available:
float) -> requests.Response:
10.         """..."""
11.
12.     @staticmethod
13.     def build_parking_monitor_data_json(parking_monitor_data:
ParkingMonitorData, free_spaces_in_frame: float,
14.                                       probability_parking_available: float) -
> dict:
15.         """..."""
16.
17.     @staticmethod
18.     def build_and_send_parking_monitor_data_put_request(parking_monitor_data:
ParkingMonitorData,
19.                                                         request_json: dict) ->
requests.Response:
20.         """..."""
21.
22.     @staticmethod
23.     def send_put_request(parking_monitor_data: ParkingMonitorData,
request_headers: dict,
24.                        request_json: dict, request_url: str) ->
requests.Response:
25.         http_basic_auth = HTTPBasicAuth(parking_monitor_data.app_username,
26.                                         parking_monitor_data.app_password)
27.         response = requests.put(request_url,
28.                                auth=http_basic_auth,
29.                                headers=request_headers,
30.                                json=request_json)
31.
32.         return response

```

perfectparking.py – class RestApiUtility

- The purpose of this class is to update and send parking monitor data to a server using HTTP PUT request.

## 4.6 Perfect Parking with Django

The Perfect Parking application was created using the popular web framework Django for a variety of reasons.

Firstly, Django is a high-level web framework that follows the Model-View-Controller (MVC) architectural pattern, which promotes code organization and separation of concerns. This allows for the development of complex programmes with numerous components without compromising the maintainability of the code.

Secondly, Django provides a lot of built-in functionality out of the box, which saves time and effort during development. For example, Django includes an Object-Relational Mapping (ORM) system that allows developers to interact with databases using Python objects, as well as a robust authentication system for user management.

Thirdly, Django has an engaged community that actively supports the framework's growth and upkeep. This indicates that a wide variety of third-party packages and extensions are readily available and can increase the capabilities of the framework and speed up development.

Along with these benefits, Django's outstanding documentation, scalability, and security capabilities are some of the other benefits of adopting it for web development. Django is also open-source and free, which makes it available to a variety of developers and organisations.

Overall, the decision to use Django for the Perfect Parking application was based on its combination of ease of use, built-in functionality, and strong community support, which makes it a popular choice for building web applications of all sizes and complexities.

#### **4.6.1 Customising Django**

The server-side code for this Django project consists of several files, including `admin.py`, `models.py`, `urls.py`, and `views.py`.

`settings.py` in `PerfectParkingWebsite` defines the app's configuration, including its name and default auto field.

```

1. """
2. Django settings for PerfectParkingWebsite project.
3. """
4.
5. from pathlib import Path
6.
7. # Build paths inside the project like this: BASE_DIR / 'subdir'.
8. BASE_DIR = Path(__file__).resolve().parent.parent
9.
10. #...
11.
12. # Application definition
13.
14. INSTALLED_APPS = [
15.     'django.contrib.admin',
16.     'django.contrib.auth',
17.     'django.contrib.contenttypes',
18.     'django.contrib.sessions',
19.     'django.contrib.messages',
20.     'django.contrib.staticfiles',
21.     'PerfectParking.apps.PerfectParkingConfig', #Import my app
22.     'rest_framework', # Import rest_framework
23.     'rest_framework.authtoken', # Import rest_framework.authtoken
24.     'django.contrib.humanize', # Import django.contrib.humanize
25. ]
26.
27. #...
28. #...
29.
30. # Static files (CSS, JavaScript, Images)
31. # https://docs.djangoproject.com/en/3.2/howto/static-files/
32.
33. STATIC_URL = '/static/'
34.
35. # Default primary key field type
36. # https://docs.djangoproject.com/en/3.2/ref/settings/#default-auto-field
37.
38. DEFAULT_AUTO_FIELD = 'django.db.models.BigAutoField'
39.
40. REST_FRAMEWORK = {
41.     'DEFAULT_PAGINATION_CLASS': 'rest_framework.pagination.PageNumberPagination',
42.     # Use Django's standard `django.contrib.auth` permissions, or allow read-only
43.     # access for unauthenticated users.
44.     'DEFAULT_PERMISSION_CLASSES': [
45.         'rest_framework.permissions.DjangoModelPermissionsOrAnonReadOnly'
46.     ],
47.     'PAGE_SIZE': 10
48. }
49. LOGIN_REDIRECT_URL = "/"
50.
51.

```

setting.py

models.py defines the data structures of ParkingLot and ParkingLotMoniton.

```
1. from geopy.distance import geodesic
2. import random
3. from django.db import models
4.
5. # Create your models here.
6. class ParkingLot(models.Model):
7.     id = models.AutoField(primary_key=True)
8.     name = models.CharField(max_length=100, unique=True)
9.     address = models.CharField(max_length=255)
10.    hours = models.CharField(max_length=255)
11.    isPaidParking = models.BooleanField(default=True)
12.    latitude = models.DecimalField(max_digits=17, decimal_places=15)
13.    longitude = models.DecimalField(max_digits=17, decimal_places=15)
14.    image = models.ImageField(upload_to='images/parking-lot/', blank=True)
15.    parking_spaces = models.IntegerField(default=1)
16.
17.    def __str__(self):
18.        return self.name
19.
```

**models.py**

urls.py defines the app's URLs, including paths for the home page, parking lots list, parking lot detail page, user registration,

```
1. from django.urls import include, path
2. from . import WebPaths
3. from . import views
4. from django.contrib.auth import views as auth_views
5.
6. urlpatterns = [
7.     path(WebPaths.ROOT, views.index, name='index'),
8.     path('logout-user/', views.logout_user, name='logout-user'),
9.     path(WebPaths.PARKING_LOTS, views.parking_lots, name='parking-lots'),
10.    path(f'{WebPaths.PARKING_LOTS}/<int:parking_lot_id>', views.parking_lot,
name='parking-lot'),
11.    path(WebPaths.REGISTER_USER, views.register_user, name='register-user'),
12.    path(WebPaths.PARKING_LOT_MONITORS, views.parking_lot_monitors, name='parking-lot-
monitors'),
13.    path(f'{WebPaths.PARKING_LOT_MONITOR}/<int:parking_lot_monitor_id>',
views.parking_lot_monitor, name='parking-lot-monitor'),
14.
```

**PerfectParking\urls.py**

To add the app URLs to the website use PerfectParkingWebsite\urls.py

```

1. from django.conf import settings
2. from django.conf.urls.static import static
3. from django.contrib import admin
4. from django.urls import include, path
5. from rest_framework import routers
6. import PerfectParking.viewsets as viewsets
7.
8. router = routers.DefaultRouter()
9. router.register(r'users', viewsets.UserViewSet)
10. router.register(r'groups', viewsets.GroupViewSet)
11. router.register(r'parking-lots', viewsets.ParkingLotViewSet)
12. router.register(r'parking-lot-monitors', viewsets.ParkingLotMonitorViewSet)
13.
14. urlpatterns = [
15.     path('', include('PerfectParking.urls')),
16.     path('admin/', admin.site.urls),
17.     path("accounts/", include("django.contrib.auth.urls")),
18.     path('api-auth/', include(router.urls)),
19.     path('api-auth/', include('rest_framework.urls', namespace='rest_framework')),
20. ]
21. urlpatterns += static(settings.MEDIA_URL, document_root=settings.MEDIA_ROOT)
22.
23.

```

#### PerfectParkingWebsite\urls.py

user login, user logout, and API endpoints for the ParkingLot and ParkingLotMonitor models.

views.py contains the logic for rendering the app's web pages, handling user input, and providing data to the app's API endpoints. It includes functions for rendering the home page, parking lot list, parking lot detail page, user registration, user login, and user logout pages. It also includes functions for handling API requests, including getting a list of all parking lots, getting details for a specific parking lot, and updating the parking lot monitor data.

```

1. from django.shortcuts import render, redirect
2. from django.contrib.auth.forms import UserCreationForm
3.
4. from django.urls import reverse_lazy
5. from . import WebPaths
6.
7. # ...
8.
9. def parking_lot_monitors(request):
10.
11.     parking_lot_monitors: list = ParkingLotMonitor.objects.all()
12.
13.     if request.method == "POST": # FORM SUBMITTED
14.         latitude = request.POST["latitude"]
15.         longitude = request.POST["longitude"]
16.
17.         context = {
18.             "parking_lot_monitors": parking_lot_monitors,
19.             "user_point": {"latitude": latitude, "longitude": longitude},
20.         }
21.         return render(request, WebPages.PARKING_LOT_MONITORS, context)
22.
23.     return render(
24.         request,
25.         WebPages.PARKING_LOT_MONITORS,
26.         {"parking_lot_monitors": parking_lot_monitors},
27.     )
28.
29.

```

#### views.py

admin.py registers the app's ParkingLot and ParkingLotMonitor models with Django's admin site.

Overall, the client-side code is responsible for providing a user-friendly interface for the app, handling user input and interactions, and communicating with the server-side code to retrieve and display data.



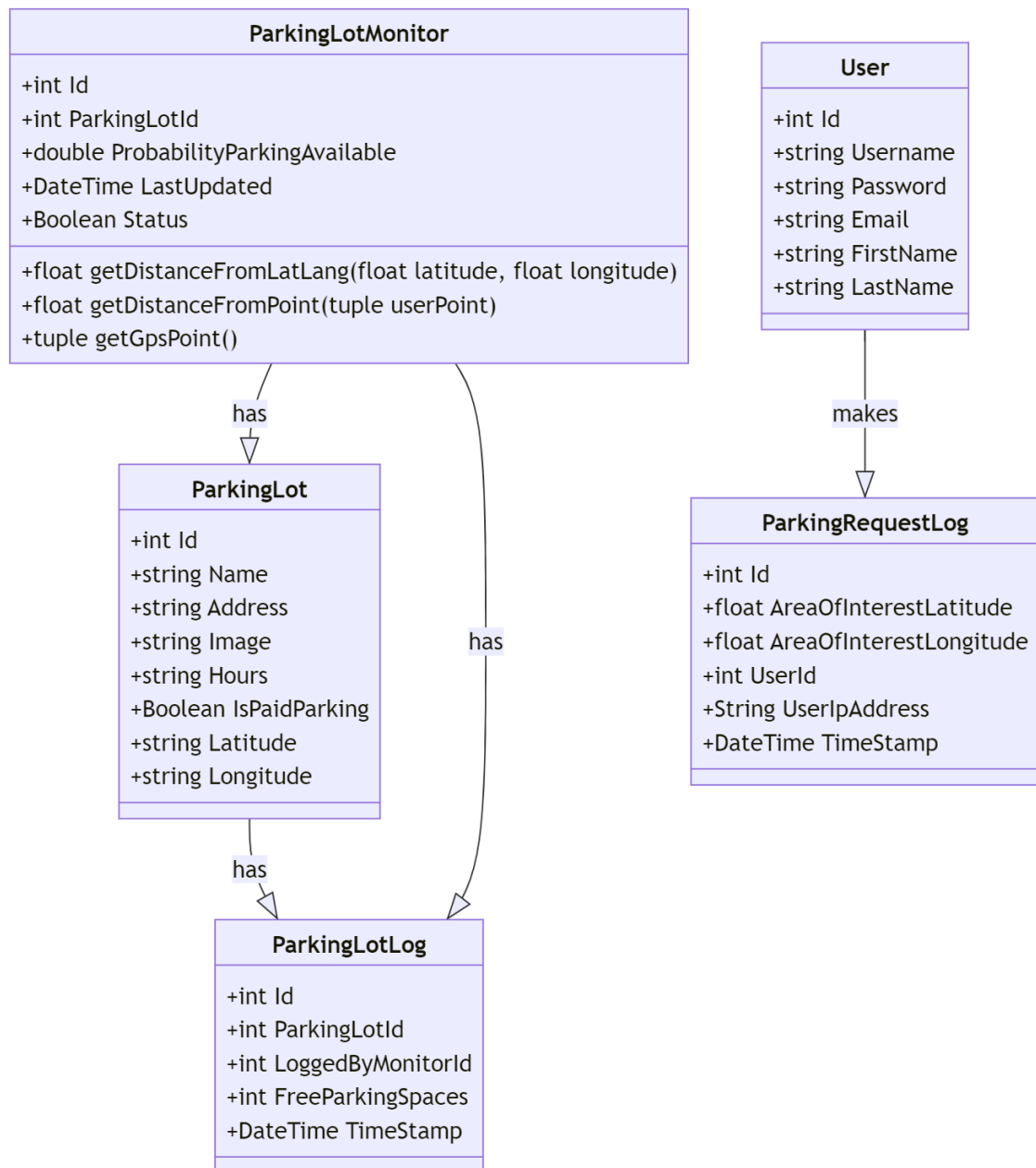


Figure 23: Class Diagram

## 4.7 Enable Django REST framework.

To enable the client-server rest functionality the Django REST framework toolkit needs to be installed. Add 'rest\_framework' to your INSTALLED\_APPS setting. All pip install requirements are handles by the requirements.txt

```
1. INSTALLED_APPS = [  
2.     #...  
3.     'rest_framework',  
4. ]
```

**PerfectParkingWebsite\settings.py**

If you're intending to use the browsable API, you'll probably also want to add REST framework's login and logout views. Add the following to your root urls.py file.

```
1. urlpatterns = [  
2.     #...  
3.     path('api-auth/', include('rest_framework.urls'))  
4. ]
```

**PerfectParkingWebsite\urls.py**

Note that the URL path value of 'api-auth/' can be whatever you want.

### 4.7.1 Server

serializers.py

serializers.py defines the serializers used to convert the ParkingLot and ParkingLotMonitor models to JSON format for use in the app's API.

```
1. class ParkingLotSerializer(serializers.HyperlinkedModelSerializer):  
2.     class Meta:  
3.         model = ParkingLot  
4.         fields = ['id', 'name', 'latitude', 'longitude']
```

**PerfectParking\serializers.py – ParkingLotSerializer**

### 4.7.2 Working with GPS data

The function `get_gps_point` creates a GPS for the ParkingLotMonitor from its' latitude and longitude values. The function `get_distance_from_point` calculates the distance between a user GPS point and the ParkingLotMonitor GPS point . Note this distance value is the distance of a direct line (path) and does not account for paths of roads.

```

1. from geopy.distance import geodesic
2. from django.db import models
3.
4. #...
5.
6. class ParkingLotMonitor(models.Model):
7.     id = models.AutoField(primary_key=True)
8.     #...
9.     image = models.ImageField(upload_to='images/parking-lot-monitor/', blank=True)
10.
11.     def get_distance_from_lat_lang(self, latitude, longitude) -> float:
12.         point:tuple = (latitude, longitude)
13.         return self.get_distance_from_point(point)
14.
15.     def get_distance_from_point(self, user_point: tuple) -> float:
16.         return round(geodesic(self.get_gps_point(), user_point).km, 2)
17.
18.
19.     def get_gps_point(self) -> tuple:
20.         return (self.latitude, self.longitude)
21.
22.     #...
23.
24.

```

PerfectParking\models.py

## 4.8 Custom tags

```

1. from django import template
2.
3. register = template.Library()
4.
5. @register.simple_tag
6. def get_distance_from_lat_lang(parking_lot_monitor, latitude, longitude):
7.     return parking_lot_monitor.get_distance_from_lat_lang(latitude, longitude)

```

PerfectParking\templatetags\get\_distance\_from\_lat\_lang.py

This snippet defines a custom template tag named "get\_distance\_from\_lat\_lang". The function retrieves the distance between the parking lot monitor's coordinates and the given latitude and longitude coordinates and returns this value to be used in the template.

```

1. {% extends 'master.html' %}
2. {% load get_distance_from_lat_lang %}
3. {% load humanize %}
4. {% block title %}Parking Monitors{% endblock %}
5.
6. {% block main %}
7. <!--...-->
8.
9. <p id="mydesc">Parking Monitors</p>
10. <table class="table table-striped" aria-describedby="mydesc">
11.     <thead>
12.         <!--...-->
13.     </thead>
14.     <tbody>
15.         {% for parking_monitor in parking_lot_monitors %}
16.         <tr>
17.             <!--...-->
18.             <td>{{ parking_monitor.latitude|floatformat:5 }}</td>
19.             <td>{{ parking_monitor.longitude|floatformat:5 }}</td>
20.             <td>{{ parking_monitor.probabilityParkingAvailable|floatformat:2 }}</td>
21.
22.             {% if user_point %}
23.             <td>{% get_distance_from_lat_lang parking_monitor user_point.latitude
24.             user_point.longitude %}</td>
25.             {% endif %}
26.             <!--...-->
27.         {% endfor %}
28.     </tbody>
29. </table>
30. {% endblock %}

```

## 4.9 HTML Template for Displaying Parking Monitor Information with Distance Calculation Getting user's location

This JavaScript code defines a function called `setPosition` that checks if the user's browser supports geolocation. If it does, it uses the browser's geolocation to get the user's current position and assigns the latitude and longitude values to form fields with IDs "latitude" and "longitude". If geolocation is not supported, it displays a message in an HTML element with ID "display" and disables a button with ID "search-near-me-button". The

```

1. <script type="text/javascript">
2.     var displayElement = document.getElementById("display");
3.     var searchNearMeButton = document.getElementById("search-near-me-button");
4.
5.     function setPosition() {
6.         if (navigator && navigator.geolocation) {
7.             navigator.geolocation.getCurrentPosition(assignPositionToForm);
8.             searchNearMeButton.disabled = false;
9.         }
10.        else {
11.            displayElement.innerHTML = "Geolocation is not supported by this
browser.";
12.            searchNearMeButton.disabled = true;
13.        }
14.    }
15.    function assignPositionToForm(position) {
16.        document.getElementById("latitude").value = position.coords.latitude;
17.        document.getElementById("longitude").value = position.coords.longitude;
18.    }
19.
20.    $(document).ready(documentReady);
21.
22.    function documentReady() {
23.        setPosition();
24.    }
25. </script>

```

#### PerfectParking\templates\website\parking-lot-monitors.html - Javascript

This is a HTML form that has two hidden input fields for latitude and longitude values. When the "Search Near Me" button is clicked, the form is submitted using the HTTP POST method, including the latitude and longitude values in the form data.

```

1. <form method="post">
2.     {% csrf_token %}
3.     <p id="display" class="text-danger"></p>
4.     <input type="hidden" name="latitude" id="latitude"/>
5.     <input type="hidden" name="longitude" id="longitude"/>
6.     <button type="submit" id="search-near-me-button" class="btn btn-primary">Search
Near Me</button>
7. </form>
8.

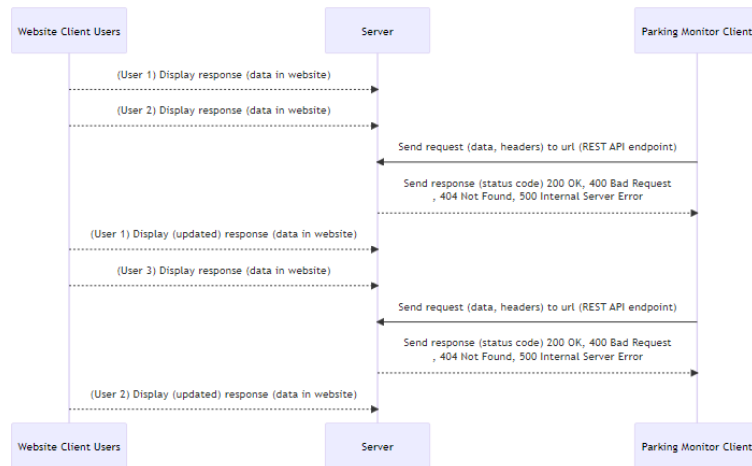
```

#### PerfectParking\templates\website\parking-lot-monitors.html – HTML Form

### 4.9.1 How They Work Together

The client and server communicate with each other through the REST API service. When the client sends a request to the server, it includes information such as the endpoint to access, any data to send in the request body, and any headers to include with the request. The server-side code receives the request and processes it, querying the database or performing other operations based on the data included in the request. The server then sends a response back to the client, which includes a status code indicating whether the

request was successful or not, any data to include in the response body, and any headers to include with the response. The server-side then processes the response, displaying the data on the website for probability of parking available so the user can view it. This cycle of request and response is how the client and server are linked together in a web application.



**Figure 24: How they work together**

## 4.10 Logging Data for Statistical Analysis.

To perform statistical analysis data must first be recorded. Defined in models.py are two classes to record data:

```

1. class ParkingLotLog(models.Model):
2.     id = models.AutoField(primary_key=True)
3.     parking_lot = models.ForeignKey(ParkingLot, on_delete=models.CASCADE)
4.     logged_by_monitor = models.ForeignKey(ParkingLotMonitor, on_delete=models.CASCADE)
5.     free_parking_spaces = models.IntegerField(default=0)
6.     time_stamp = models.DateTimeField(auto_now=True)
7.
8.     def __str__(self) -> str:
9.         return f"{self.parking_lot.name} - {self.time_stamp}"
  
```

**The Parking Lot Log Class**

```

1. class ParkingRequestLog(models.Model):
2.     id = models.AutoField(primary_key=True)
3.     area_of_interest_latitude = models.DecimalField(max_digits=17, decimal_places=15)
4.     area_of_interest_longitude = models.DecimalField(max_digits=17, decimal_places=15)
5.     time_stamp = models.DateTimeField(auto_now=True)
6.     user = models.ForeignKey(User, on_delete=models.CASCADE, null=True)
7.     user_ip_address = models.CharField(max_length=15)
8.
  
```

**The Parking Request Log Class**

### 4.10.1 Parking Lot Log

When a parking lot monitors record is saved or updated the server a historical log (Parking Lot Log) of the car park data by overriding the two built in functions of model i.e., update and save.

```
1. class ParkingLotMonitor(models.Model):
2.     id = models.AutoField(primary_key=True)
3.     parkingLot = models.ForeignKey(ParkingLot, on_delete=models.CASCADE)
4.     name = models.CharField(max_length=100, unique=True)
5.     #...
6.     free_parking_spaces = models.IntegerField(default=0)
7.     # ...
8.
9.     def update(self, *args, **kwargs):
10.         """Overrides the update method to create a parking lot log."""
11.         super(ParkingLotMonitor, self).save(*args, **kwargs)
12.         ParkingLotLog.objects.create(parking_lot=self.parkingLot,
logged_by_monitor=self, free_parking_spaces=self.free_parking_spaces)
13.
14.     def save(self, *args, **kwargs):
15.         """Overrides the save method to create a parking lot log."""
16.         super(ParkingLotMonitor, self).save(*args, **kwargs)
17.         ParkingLotLog.objects.create(parking_lot=self.parkingLot,
logged_by_monitor=self, free_parking_spaces=self.free_parking_spaces)
```

#### Parking Lot Monitor overridden methods

### 4.10.2 Parking Request Log

When a user makes a parking search request, data about the request is saved by a function record\_user\_query in utility.py called in the function 'parking\_lot\_monitors' in views.py.

```
1. """ Utility for the PerfectParking project"""
2. from django.contrib.auth.models import User
3. from .models import ParkingRequestLog
4.
5. def record_user_query(area_of_interest_latitude, area_of_interest_longitude, request):
6.     parking_request_log: ParkingRequestLog = ParkingRequestLog()
7.     parking_request_log.area_of_interest_latitude = area_of_interest_latitude
8.     parking_request_log.area_of_interest_longitude = area_of_interest_longitude
9.     parking_request_log.user =
User.objects.filter(username=request.user.username).first()
10.     parking_request_log.user_ip_address = request.META.get("REMOTE_ADDR")
11.
12.     parking_request_log.save()
```

#### Record User Query

```
1. def parking_lot_monitors(request):
2.     # ...
3.     if request.method == "POST": # FORM SUBMITTED
4.         # ...
5.         record_user_query(latitude, longitude, request)
6.         #...
7.         return render(request, WebPages.PARKING_LOT_MONITORS, context)
8.     #...
9.     return render(request, WebPages.PARKING_LOT_MONITORS, context)
```

### **Parking Lot Monitors View**



## Chapter 5 Testing and Results

### 5.1 Unit Testing

Unit tests are a type of software testing that involves testing individual units or components of a larger software system. Unit testing's goal is to verify that every single piece of code is operating as intended and adhering to design specifications. (TechTarget , 2023)

In most cases, developers write unit tests as they write the application's code, these tests are usually automated. They are made to be quickly and regularly executed, enabling programmers to and correct flaws early in the development cycle. (Ingrassellino, 2021)

Developers write code to test every individual function, method, or class in the software system during unit testing. To make sure that the code generates the desired output for each combination of inputs, they supply test data and anticipated results for each unit before running the tests. (Kolodiy, n.d.)

For software to be dependable, manageable, and scalable, unit tests are crucial. They can assist in identifying issues early in the development process, which makes fixing them simpler and less expensive. Additionally, they aid in ensuring that modifications to the code don't ruin already-existing functionality. (Ingrassellino, 2021)

Overall, unit tests are an important part of the software development process and are essential for ensuring the quality and reliability of software applications.

#### 5.1.1 Unit Tests Used in The Project

this unit test is verifying that the `update_server_parking_monitor_data` method is correctly updating the parking monitor data on the server and returning the expected response status code.

```
1. def test_update_server_parking_monitor_data(self):
2.     parking_monitor_data = ParkingMonitorData()
3.     free_spaces_in_frame: float = 3
4.     probability_parking_available: 0.5 # (3/6)
5.     response: request.Response = RestApiUtility.update_server_parking_monitor_data(
6.         parking_monitor_data, free_spaces_in_frame, probability_parking_available
7.     )
8.     expected_response_status_code = 200
9.     self.assertEqual(response.status_code, expected_response_status_code)
```

ParkingLot/tests/test.py

To run the tests, open a terminal and enter the code:

```
1. python -m unittest /parking_lot/tests/parking_lot_tests.py
```

**Unit Test Terminal run command.**

## **5.2 Behaviour Driven Development (BDD) Testing**

A software development process called behaviour-driven development (BDD) expands on the ideas behind test-driven development (TDD). But it emphasises cooperation and communication more than ever between technical and non-technical stakeholders. The goal of BDD is to guarantee that everyone involved with the development process is aware of the specifications and goals for the software that is being created. (Tricentis, 2018)

Instead of just developing and passing tests, BDD focuses on defining and implementing the software's expected behaviour. This is accomplished by using the specialised language "Gherkin," which enables programmers, testers, and business stakeholders to specify the desired behaviour of the software in a form that is clear to all parties. (Das, 2022)

One of the key benefits of BDD is that it helps to ensure that everyone involved in the development process is on the same page. The danger of miscommunication and misunderstandings between technical and non-technical stakeholders is reduced by establishing a common vocabulary and outlining the desired behaviour of the software upfront. (Das, 2022)

BDD also has the potential to raise the bar on the software that is being created. Developers and testers may make sure that the software satisfies the needs and expectations of the end users by concentrating on the desired behaviour of the product.

BDD also promotes team member participation and communication, which can assist to break down silos and make sure that everyone is working towards the same objective. This can lead to greater efficiency and productivity, as well as a more positive and collaborative working environment. (Testingxperts, 2023)

Overall, Behaviour-driven development (BDD) is an approach to software development that places a strong emphasis on collaboration and communication between technical and non-technical stakeholders. By defining the desired behaviour of the software upfront and using a common language, it can help to ensure that everyone involved in the

development process is working towards a common goal and that the software being developed meets the requirements and expectations of the end-users.

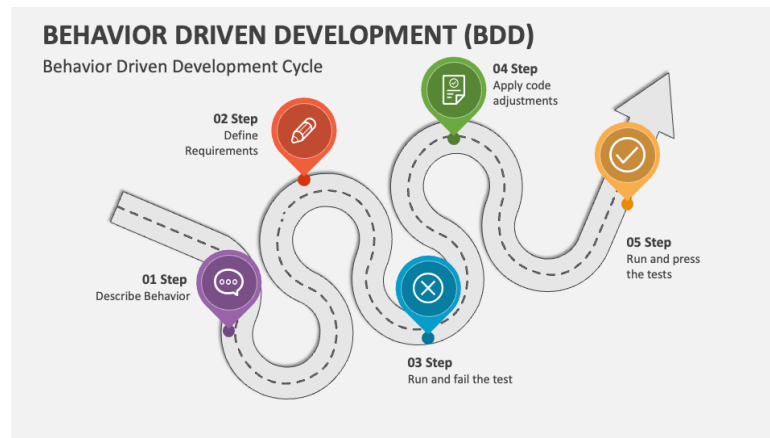


Figure 25: BDD development cycle (Collidu, n.d.)

### 5.2.1 BDD In Perfect Parking

BDD testing was not used during the development of the Perfect Parking application due to limited time available for its implementation. As a solo developer in the project, collaboration with others was not possible, the time and effort required for BDD testing was perceived as a barrier to meeting project deadlines.

However, if there were plans to develop a large new project, there are still several reasons why BDD testing would be considered. First, BDD testing can help ensure that software meets the business requirements and user needs, which can result in a higher quality product that better meets the needs of users.

Finally, BDD testing can also help improve developers understanding of the project requirements and improve the efficiency of the development process.

Overall, while limited time and a lack of collaboration may have made it difficult to implement BDD testing in the development of the Perfect Parking application, there are still many benefits to using BDD testing in a new solo project that make it worth considering.

## 5.3 Test Driven Development (TDD)

A software development process called Test-Driven Development (TDD) places a strong emphasis on creating automated tests before creating any production code. In test-driven

development (TDD), the developer creates a test that outlines a desired software feature or behaviour, runs the test to discover its failure, and then writes the code to make the test pass. The process is then repeated with additional tests until the desired functionality is complete. (Unadkat, 2023)

With TDD, the developer is forced to consider the desired behaviour of the code before actually developing it because tests must be written first. This produces more modular, maintainable code that is also less prone to problems. It also allows for easier refactoring of code later, as the tests act as a safety net to catch any unintended changes that may have been introduced. (Santacroce, 2022)

TDD typically involves theses following steps:

- **Writing a failing test:** The developer purposefully constructs a test that fails even though it defines the desired behaviour of the code. The test is written before writing any code for production.
- **Write the minimum amount of code to make the test pass:** The developer then creates the minimum amount of code required to pass the failing test. This code was written exclusively for the purpose of passing the test.
- **Refactor the code:** When the test passes, the developer can refactor the code without affecting its functionality in order to make it easier to maintain.
- **Repeat:** The developer then repeats the process with additional tests until the desired functionality is complete. (Steinfeld, 2020)

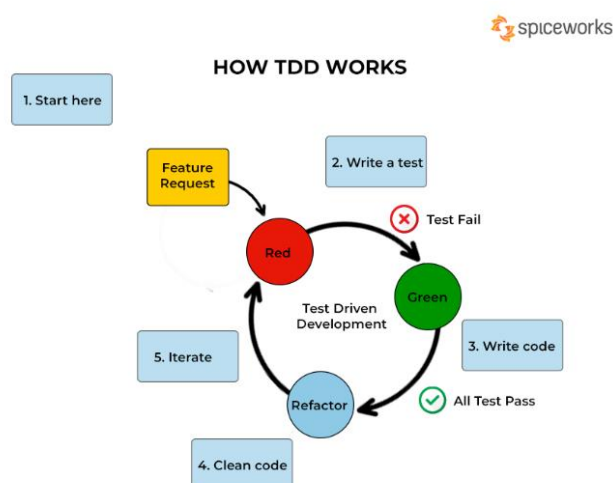


Figure 26: How TDD Works (BasuMallick, 2022)

TDD can help increase productivity in addition to code quality by decreasing time spent on debugging and by establishing a clear set of requirements for the software being built. It does, however, require some upfront effort in writing the tests upfront, which may not be desirable for smaller projects or teams with fewer resources.

Overall, TDD is a powerful technique for writing high-quality, maintainable code, and can be especially useful in larger, more complex software projects.

### **5.3.1 TDD in Perfect Parking**

TDD wasn't used in the development of Perfect Parking for several reasons. Firstly, when working on a solo project, it didn't feel necessary to write the tests because the code could be simply tested as its written. Additionally, it can be difficult to justify the additional time investment required for TDD when working on the project solo. However, even on a solo project, TDD can help to ensure that the code is of high quality and easier to maintain. Additionally, writing tests upfront can help to prevent regressions and ensure that the code works as expected, even if you come back to the project months or years later.

Secondly, TDD also wasn't used in the development of the solo project, such as the Perfect Parking application, was due to the limited knowledge of TDD and the lack of time to learn how to implement it effectively.

In conclusion, there are several good reasons why TDD might not have been used in the creation of your Perfect Parking. These include the lack of familiarity with TDD and the lack of free time to learn more about it. However, learning about TDD might be a worthwhile time and effort investment if interested in developing abilities and producing higher quality code.

If another project was to be developed in the future TDD would be implemented for several reasons:

Firstly, TDD can help ensure that the code is well-structured and maintainable, which can be especially important for larger projects.

Secondly, TDD can help improve the quality of the code. By promoting modular, reusable, and maintainable code, TDD can ensure that each component of the code works as intended and fits into the larger system.

Thirdly, TDD can increase confidence in code changes. By having a comprehensive suite of tests, developers can be confident that their code changes will not negatively affect other parts of the system.

Lastly, TDD can improve collaboration between developers and teams. By providing a common language for discussing code changes and ensuring that everyone is on the same page in terms of what the code should do, TDD can reduce communication errors and misunderstandings between developers.

In conclusion, using TDD in a software development project can result in higher-quality code, a better user experience, and increased developer efficiency. TDD can ensure the success of upcoming project by identifying problems earlier in the development process, encouraging modular and maintainable code, boosting confidence in code modifications, and enhancing teamwork.

## **5.4 Functionality**

During testing, the Perfect Parking application demonstrated strong functionality in displaying the availability of parking spots based on real-time data collected by the ParkingLotMonitor. The ParkingLotMonitor allows the user to mark out parking spots that they want to be continuously monitored for their occupancy status. Whenever there is a change in the availability of a parking spot the mapped-out spaces turn green for available and red for occupied, the monitor sends the probability of it being available to the Perfect Parking application. The application then displays the availability of the car park to the user in real-time by displaying the probability of available parking spaces. Overall, the application's functionality met the project's objectives and proved to be a valuable tool for drivers seeking parking in urban or congested areas. However, some minor issues were identified during testing, such as if a shadow is casted over a parking spot and it changes the colour gradient in the space the monitor would think that the space is full when in theory it's not. If the project was to go further in the future this minor issue would be fixed.

## **5.5 Usability**

The usability of the ParkingLotMonitor and Perfect Parking website were both tested during the evaluation process. The ParkingLotMonitor interface was found to be intuitive and user-friendly, allowing a user to easily mark out and monitor parking spots. The monitor also accurately reported the availability of parking spots in real-time, with color-coding providing a clear visual indication of the status of each spot. The Perfect Parking website was also found to be intuitive and easy to use. The website provided an interface where users could view the availability of parking spots in real-time. The website allowed signed-in users to view a car park's availability, and when the car park was selected, it showed a Google map of the parking location, where they could get directions to the car park if they wish. Overall, the usability of both the ParkingLotMonitor and Perfect Parking website was strong, with minimal issues or confusion reported during testing.

## Chapter 6 Conclusions

Based on the research and analysis conducted in this project, it can be concluded that the use of AI and object detection in images is an effective solution for identifying the status of parking spots (free/taken) in towns and cities such as Limerick. By implementing this technology, it is possible to monitor car parks in real-time,

### 6.1 Application Performance

The PerfectParking application performed exceptionally well, exceeding expectations. The application was able to efficiently send updates to the server and display the results in the application, which was even better than expected. This highlights the effectiveness of the application's design and development, as it was able to meet and surpass the needs and expectations.

### 6.2 Future Development

While developing the Perfect Parking application the use of statistical analysis and machine learning on big data was limited by time. For example, here is a graph based on randomly generated records of car park activity generated.

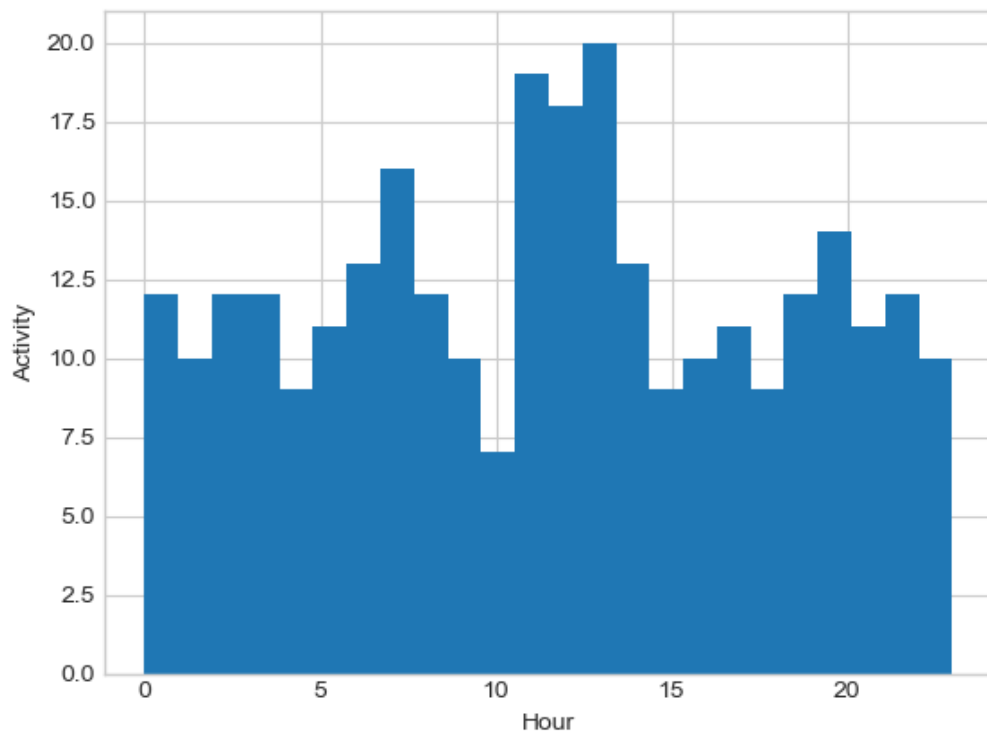


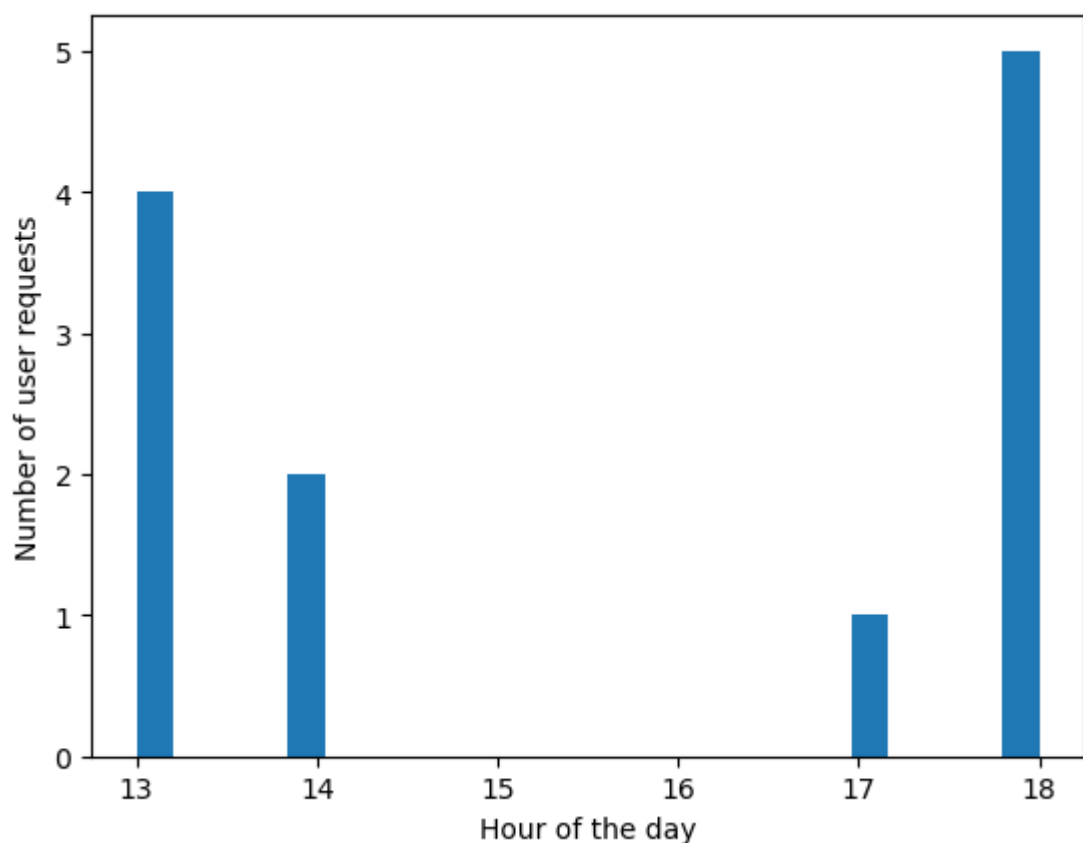
Figure 27: Parking Test Data Histogram



Using Machine learning with collected real world big data the application could give predictions of a car park's occupancy for a time in the future and answer user queries e.g., “Will there be available parking like tomorrow at 3:00 PM in Limerick city?”

By adding local and cultural context to data points predictions could be improved beyond a typical Monday to Sunday and hours predication e.g., parking in Limerick city at 3:00 PM on an average Tuesday is not the same as parking on 3:00 PM on Christmas EVE on a Tuesday. With enough collected data over long time spans the application could make predictions for holidays.

This data analysis and the predictions could be used to help with urban planning i.e., the need to build (or not) more car parks.



**Figure 28: User Request Talley for A Given Monday**

The initial work for this is detailed in two Jupyter notebooks: [parking-log-data-analysis.ipynb](#) and [parking-request-data-analysis.ipynb](#) both available on GitHub at <https://github.com/rhysquilter/perfect-parking-fyp/tree/master/notebooks> .

### **6.3 Reflections**

During this FYP, a deep understanding of computer vision techniques and their practical applications using OpenCV was gained. Additionally, the historical development of Haar classifiers and wavelets was studied, which helped appreciate the evolution of these key concepts in the field.

Furthermore, programming skills, particularly in Python, were honed through the implementation of this project. This experience taught valuable lessons in project management, design, and implementation. Accomplishments were made and a meaningful contribution to the field of computer vision was achieved.

Overall, the knowledge gained, and the skills developed broadened the understanding of computer vision and prepared for taking on new challenges in this exciting and rapidly evolving field. The opportunity to pursue this research was appreciated and the new skills and knowledge will be applied in future endeavours.

To conclude this thesis, a challenging yet rewarding journey was experienced that allowed delving deeply into the field of computer vision. Through research and implementation of the FYP project, valuable knowledge and skills were gained that will serve well in future academic and professional pursuits.

The skills and knowledge acquired through this thesis will be utilized in future academic and professional endeavours. The prospects of this journey are exciting and there is a strong belief that the lessons learned, and experiences gained will serve as valuable guidance throughout the upcoming career.

## References

- abdalslam, 2023. *Parking Management Statistics, Trends And Facts 2023*. [Online]  
Available at: <https://abdalslam.com/parking-management-statistics#parking-management-statistics>  
[Accessed 27 04 2023].
- Arunachalam, H., 2014. *Image Segmentation for the Extraction of Face*. , Rajiv Gandhi:  
s.n.
- Ashish, 2022. [Online]  
Available at: <https://www.scienceabc.com/innovation/how-does-google-maps-know-about-traffic-conditions.html#:~:text=Google%20Traffic%20works%20by%20crowdsourcing,geographic%20location%20with%20the%20app>.
- atlassian, n.d. *What is Agile?*. [Online]  
Available at: <https://www.atlassian.com/agile>  
[Accessed 27 04 2023].
- Bak, K., 2023. *Cascade Classifier: Approach to Object Detection*. [Online]  
Available at: <https://53jk1.medium.com/cascade-classifier-approach-to-object-detection-890ef859cc53>  
[Accessed 24 04 2023].
- Barnett, M., 2017. [Online]  
Available at: <https://www.fourthsource.com/data/importance-real-time-data-five-reasons-need-22014>
- BasuMallick, C., 2022. *What Is TDD (Test Driven Development)? Process, Importance, and Limitations*. [Online]  
Available at: <https://www.spiceworks.com/tech/devops/articles/what-is-tdd/>  
[Accessed 27 04 2023].
- Boesch, G., 2023. *What is OpenCV? The Complete Guide (2023)*. [Online]  
Available at: <https://viso.ai/computer-vision/opencv/>  
[Accessed 23 May 2023].

Botelho, B., 2020. *Big Data*. [Online]  
Available at: <https://www.techtarget.com/searchdatamanagement/definition/big-data>  
[Accessed 16 02 2022].

brouton Lab, n.d. *How IoT and Computer Vision Revolutionize Parkings*. [Online]  
Available at: <https://broutonlab.com/blog/iot-and-computer-vision-in-parkings#:~:text=Parking%20systems%20can%20also%20implement,spots%20that%20may%20cost%20less>.  
[Accessed 25 06 2023].

Cirrus, O., 2018. *What is Distributed Computing, its Pros and Cons?*. [Online]  
Available at: <https://opencirrus.org/distributed-computing-pros-cons/#:~:text=Although%20distributed%20computing%20has%20its,high%20workloads%20and%20big%20data>.  
[Accessed 16 07 2023].

Collidu, n.d. *Behavior Driven Development (BDD)*. [Online]  
Available at: <https://www.collidu.com/presentation-behavior-driven-development-bdd>  
[Accessed 26 04 2023].

CSO.ie, 2019. *National Travel Survey 2019*. [Online]  
Available at: <https://www.cso.ie/en/releasesandpublications/ep/p-nts/nationaltravelsurvey2019/carusage/>  
[Accessed 27 04 2023].

Das, S., 2022. *BDD Testing: A Detailed Guide*. [Online]  
Available at: <https://www.browserstack.com/guide/what-is-bdd-testing>  
[Accessed 26 04 2023].

Das, S., 2022. *Benefits of Test Management and BDD in Software Testing Process*. [Online]  
Available at: <https://www.browserstack.com/guide/benefits-of-test-management-and-bdd>  
[Accessed 26 04 2023].

Feer, J. v. d., 2020. *Be a Boss*. [Online]  
Available at: <https://fiches-pratiques.chefdentreprise.com/Thematique/gestion->

1050/FichePratique/Les-raisons-utiliser-methodes-Agile-entreprise-354321.htm

[Accessed 27 04 2023].

Ingrassellino, J., 2021. *Why Should Software Testers Understand Unit Testing?*. [Online]  
Available at: <https://applitools.com/blog/why-should-software-testers-understand-unit-testing/>

[Accessed 26 04 2023].

Ismail, K., 2023. *API eCommerce: The Role of APIs In The Shopping Journey*. [Online]  
Available at: <https://instantcommerce.io/blog/api-e-commerce>  
[Accessed 21 08 2023].

Johnson, S., n.d. *What Is Django and What Is Django Used for?*. [Online]  
Available at: <https://www.stxnext.com/blog/what-is-django/>  
[Accessed 24 04 2023].

Kolodiy, S., n.d. *Unit Testing and Coding: Why Testable Code Matters*. [Online]  
Available at: <https://www.toptal.com/qa/how-to-write-testable-code-and-why-it-matters>  
[Accessed 26 04 2023].

Lee, D., 2022. *Cascade Classifiers*. [Online]  
Available at: <https://apmonitor.com/pds/index.php/Main/CascadeClassifier>  
[Accessed 24th April 2023].

Lutkevich, B., 2022. *distributed applications*. [Online]  
Available at: <https://www.techtarget.com/searchitoperations/definition/distributed-applications-distributed-apps>  
[Accessed 15 07 2023].

McCoy, K., n.d. *Drivers spend an average of 17 hours a year searching for parking spots*. [Online]  
Available at: <https://eu.usatoday.com/story/money/2017/07/12/parking-pain-causes-financial-and-personal-strain/467637001/>  
[Accessed 24 04 2023].

McKinney, D., 2022. *What is Conda*. [Online]  
Available at: <https://cloudsmith.com/blog/what-is-conda/>  
[Accessed 27 04 2023].

Michael Jones, P. V., 2001. *Rapid Object Detection using a Boosted Cascade of Simple*. [Online]

Available at: <https://www.cs.cmu.edu/~efros/courses/LBMV07/Papers/viola-cvpr-01.pdf>

[Accessed 23 May 2023].

Nolle, T., 2021. *The 5 essential HTTP methods in RESTful API development*. [Online]

Available at: <https://www.techtarget.com/searchapparchitecture/tip/The-5-essential-HTTP-methods-in-RESTful-API-development>

[Accessed 15 08 2021].

O Reilly, n.d. *Advantages of Anaconda*. [Online]

Available at: <https://www.oreilly.com/library/view/mobile-artificial-intelligence/9781789344073/9ef54367-7a36-4064-ae81-e65ce25e80bb.xhtml>

[Accessed 27 04 2023].

Patel, A., 2020. *What is Object Detection?*. [Online]

Available at: <https://medium.com/ml-research-lab/what-is-object-detection-51f9d872ece7>

[Accessed 23 May 2023].

Ponnambalam, C. T., 2020. *Searching for Street Parking: Effects on Driver Vehicle Control, Workload, Physiology, and Glances*. [Online]

Available at: <https://www.frontiersin.org/articles/10.3389/fpsyg.2020.574262/full>

[Accessed 12 12 2022].

PubNub, n.d. *What is a Map (Mapping) API?*. [Online]

Available at: <https://www.redhat.com/en/blog/role-apis-increasingly-digital-world>

[Accessed 21 08 2023].

Quellmalz, R., 2021. *6 surprising facts about parking you probably dont know*. [Online]

Available at: <https://www.spotparking.com.au/insights/facts-about-parking-you-probably-didnt-know>

Red Hat, 2020. *What is a REST API?*. [Online]

Available at: <https://www.redhat.com/en/topics/api/what-is-a-rest-api>

[Accessed 21 08 2023].

Reid, C., 2016. *GOOGLE POPULAR TIMES LIVE IS TRACKING YOUR CUSTOMERS*. [Online]

Available at: <https://8ms.com/blog/google-popular-times-is-tracking-your-customers/>  
[Accessed 08 08 2023].

Roca, C., n.d. *REST API: what it is, how it works, advantages and disadvantages*. [Online]

Available at: <https://www.thepowermba.com/en/blog/rest-api-what-it-is>  
[Accessed 21 08 2023].

Rocheeva, O., 2018. *Parking Space Detection in OpenCV*. [Online]  
Available at: <https://github.com/olgarose/ParkingLot>  
[Accessed 20 February 2023].

Root, D., 2020. *An Overview of The Anaconda Distribution*. [Online]  
Available at: <https://towardsdatascience.com/an-overview-of-the-anaconda-distribution-9479ff1859e6>  
[Accessed 27 04 2023].

S.Gillis, A., 2022. *Distrubuted computing*. [Online]  
Available at: <https://www.techtarget.com/whatis/definition/distributed-computing>  
[Accessed 15 07 2023].

Santacroe, F., 2022. *Test-Driven Development (TDD): A Time-Tested Recipe for Quality Software*. [Online]  
Available at: <https://semaphoreci.com/blog/test-driven-development>  
[Accessed 27 04 2023].

Sawer, P., 2017. *Search for parking spaces cost drivers four days a year*. [Online]  
Available at: <https://www.independent.ie/regionalsherald/news/search-for-parking-spaces-costs-drivers-four-days-a-year-35416591.html>  
[Accessed 24 04 2023].

Seal, A., n.d. Thermal Human face recognition based on Haar wavelet. In: s.l.:s.n., p. 12.

Sharma, R., 2022. *Top 6 Major Challenges of Big Data & Simple Solutions To Solve Them*. [Online]

Available at: <https://www.upgrad.com/blog/major-challenges-of-big-data>  
[Accessed 15 01 2023].

Simplilearn, 2023. *What Is Computer Vision: Applications, Benefits and How to Learn It.* [Online]

Available at: [https://www.simplilearn.com/computer-vision-article#what\\_is\\_computer\\_vision](https://www.simplilearn.com/computer-vision-article#what_is_computer_vision)  
[Accessed 23 06 2023].

Sonar Rules, n.d. *Python static code analysis.* [Online]  
Available at: <https://rules.sonarsource.com/python/RSPEC-3776>  
[Accessed 28 04 2023].

Splunk, 2021. [Online]  
Available at: [https://www.splunk.com/en\\_us/data-insider/what-is-real-time-data.html](https://www.splunk.com/en_us/data-insider/what-is-real-time-data.html)  
[Accessed 03 April 2023].

Steinfeld, G., 2020. *5 steps of test-driven development.* [Online]  
Available at: <https://developer.ibm.com/articles/5-steps-of-test-driven-development/>  
[Accessed 27 04 2023].

Tech Target, n.d. *object recognition.* [Online]  
Available at: <https://www.techtarget.com/whatis/definition/object-recognition>  
[Accessed 21st May 2023].

TechTarget, 2023. *Unit testing.* [Online]  
Available at: <https://www.techtarget.com/searchsoftwarequality/definition/unit-testing>  
[Accessed 26 04 2023].

TechVidan, 2022. *Why Big Data – Benefits and Importance of Big data.* [Online]  
Available at: <https://techvidvan.com/tutorials/why-big-data/>  
[Accessed 15 02 2023].

Testingxperts, 2023. *BDD (Behavior Driven Development) Testing: 7 Benefits that Ensure High Product Quality.* [Online]  
Available at: <https://www.testingxperts.com/blog/bdd-testing#:~:text=BDD%20testing%20uses%20a%20plain,teams%20on%20the%20same>



%20page.

[Accessed 26 04 2023].

Tricentis, 2018. *What is BDD (Behavior-Driven Development)?*. [Online]  
Available at: <https://www.tricentis.com/blog/bdd-behavior-driven-development>  
[Accessed 26 04 2023].

Turner, A., 2023. *How Many Smartphones Are In The World?*. [Online]  
Available at: <https://www.bankmycell.com/blog/how-many-phones-are-in-the-world>  
[Accessed 20 April 2023].

Tyagi, M., 2021. *Viola Jones Algorithm and Haar Cascade Classifier*. [Online]  
Available at: <https://towardsdatascience.com/viola-jones-algorithm-and-haar-cascade-classifier-ee3bfb19f7d8>  
[Accessed 24 04 2023].

Unadkat, J., 2023. *What is Test Driven Development (TDD)?*. [Online]  
Available at: <https://www.browserstack.com/guide/what-is-test-driven-development>  
[Accessed 27 04 2023].

University of Limerick, 2022. *Facts and figures*. [Online]  
Available at: <https://www.ul.ie/presidents-office/university-profile/facts-and-figures>  
[Accessed 16 01 2023].