Problem Solving and Engineering Design part 3

CW1B2

Ruben Mariën (r0883561)
Robin Martens (r0885874)
Neel Van Den Brande (r0876234)
Rik Vanhees (r0885864)
Rune Verachtert (r0884615)
Tuur Vernieuwe (r0886802)

An intelligent parking garage

PRELIMINARY REPORT

Co-titular
prof. dr. ir. Bart De Decker

Coaches
Shuaibu Musa Adam
Shirin Kalantari
Hamdi Trimech

ACADEMIC YEAR 2022-2023

Contents

List of Figures

List of Tables

List of Abbreviations

ANPR	Automatic Number Plate Recognition
API	Application Programming Interface
GDPR	General Data Protection Regulation
HTTPS	HyperText Transport Protocol Secure
HTTP	HyperText Transport Protocol
IOT	Internet of Things
JSON	JavaScript Object Notation
ORM	Object Relational Mapper
OS	Operating System
REST	Representational State Transfer
SQL	Structured Query Language
SSL	Secure Socket Layer
TLS	Transport Layer Security
UDMS	Ultrasonic Distance Measuring Sensor
WSGI	Web Server Gateway Interface
CSRF	Cross Site Request Forgery
DDOS	Distributed Denial of Server
GUI	Graphical User Interface
IDOR	Indirect Object Reference
IPS	Intelligent Parking System
JS	JavaScript
LAN	Local Area Network

1 Introduction

People these days value efficiency in life on all fronts ever higher. Time has become more and more valuable and loss of time has to be minimal. Since almost every family owns a car with most of them even owning two or three [STATBEL, 2022], and travelling by car has become increasingly popular. Parking efficiency is an element that affects everyone on a near daily basis. This requires engineers to invent and examine new solutions to improve this parking experience. The dissatisfaction with the classic ticket system in most parking garages, causes an evolution into more digital and automated systems. Often accomplished through licence plate recognition in combination with mobile apps and automated payment options [4411, 2022]. Despite the convenience of these systems, it poses a risk of a privacy breach [Verheyden, 2022]. Therefore, sufficient attention should be given to the security of these systems. The privacy of the user is the number one priority.

The main goal of the project is to make an automatic parking garage using Internet of Things (IoT) devices. This means that a client will be able to drive into the garage and park his/her car here for a certain duration of time. Then, drive away without having to pay with the use of a ticket. This is accomplished by cameras and Automatic Number Plate Recognition (ANPR) software, together with a mobile application.

This intermediate report will describe both the mechanical and software aspects of the current state of the design. Section 1.1 describes the working of the entire system in an abstract way. Section 3 gives the concrete implementation of the system. The reasons why the components of the system were chosen can be found in section 4.Then, section 5 will describe a sample user experience in the Intelligent Parking System (IPS). Section 7 will give a conclusion of the project and finally, section 8 gives an overview of the course integration with the different courses in the first and second year of the Bachelor in Engineering Science.

1.1 Problem description

The official problem description is very broad: "design a fully functional intelligent parking garage" with the following requirements [Decker and Hughes, 2022]:

1. the parking garage detects the amount of available parking lots;
2. the amount of available parking lots is displayed across multiple screens;
3. drivers can reserve a parking lot;
4. the parking garage detects entering and exiting vehicles which eliminates the need of parking tickets.

Therefore, the following infrastructure has to be designed, provided and built:

1. sensors to detect the occupancy of a parking lot;
2. a central server which stores and provides all the necessary data;
3. a frontend application which the clients can use.

The main research question this project tries to answer is "*How can we realise a safe* IoT *-infrastructure which makes parking easier, faster and foremost, safer?*" [Decker and Hughes, 2022].

2 General design description

This section describes the entire working of the system in an abstract way. The details of the concrete implementation can be found in Section 3.

The IPS consists of three main parts: the on site sensors and gateway, the frontend application and the backend.

The backend is the central entity in the system, as both the local system and the frontend application connect to it in order to query data. This is done via a Representational State Transfer (REST) Application Programming Interface (API), which transfers data in JavaScript Object Notation (JSON)-format; the standard for REST APIs [Gupta, 2022]. The backend consists out of the combination of the servers which handles the incoming requests and the database, which stores all the information about the parking garage (e.g. occupancy of parking lots, users inside the parking garage, etc.). The backend also hosts the web variant of the frontend application and thus needs proper redirecting in order to redirect requests to the right destinations (in casu the frontend application or the database). The backend resides in a separate location (e.g. a data centre), separated from the actual garage.

The local setup requires a gateway (IoT-device) and sensors installed in the physical garage. The sensors and the gateway are preferably connected through a Local Area Network (LAN), but cable wiring is also possible, although less practical.

The gateway is a micro-controller or IoT-device which is the central processing unit for the local system. It can post and request data from the backend and controls the interaction between the different sensors. It performs only simple logic tasks and delegates from complex calculation to the backend.

Two cameras detect the cars which are entering or exiting the parking garage, via ANPR. They are linked via the gateway with two servo motors, which control the barriers for entering and exiting the garage. Inside the garage, sensors detect if a parking lot is occupied or not. A LED visually indicates to the drivers if the parking lot is occupied. Furthermore, a display indicates the amount of free spots left in the garage.

The frontend application serves as a Graphical User Interface (GUI) for the user to interact with the backend database. The application comes in two formats: a mobile application which is installable on both iOS and Android and a web application in the form of a website. Both variants have the same functionality.

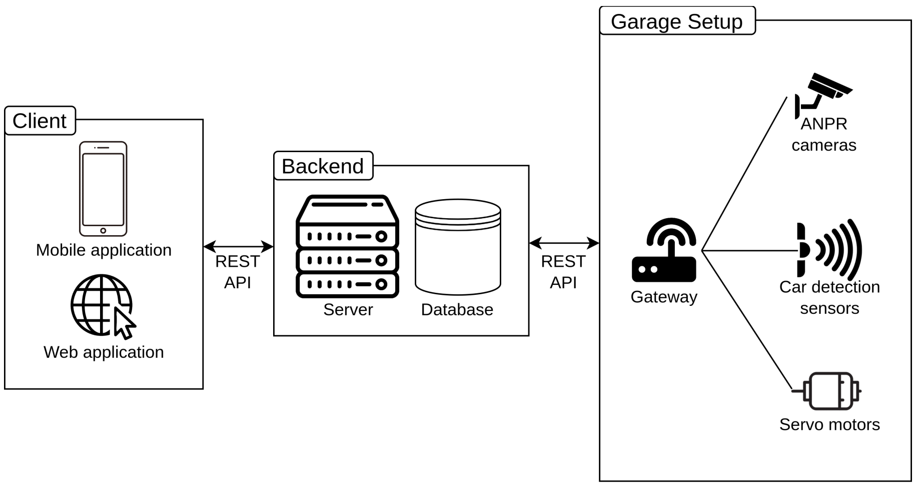

Figure 1: Abstract overview of the intelligent parking system. The backend can be deployed on one psychical machine, which can run the proxy server and the database server in parallel.

2.1 Core functionalities

The three main spearheads of the IPS described above are user privacy, security and ease of use. Table 1 gives a summary of this paragraph, listing the three core functionalities, together with the steps taken to achieve them.

The ease of use is split up in three sub-objectives.

Firstly, it is not necessary for users of the parking garage to install the mobile application or even to have a pre-existing account. It possible that users can drive to the parking garage and park without any need for prior setup, but with an almost equal user experience.

Secondly, if the user creates an account, he/she is able to reserve parking lots in a specific garage from and to specific hours, such that the availability of a free parking lot is guaranteed at arrival of the user.

Thirdly, the infrastructure supports automatic and non-automatic payments from within the frontend application, the former for user which created an account, the latter as an alternative for users without an account.

Besides the ease of use of a person parking in the garage, the frontend application also supports admin features for a garage owner. A garage owner can add and delete garages from the system, as well as configure the parking lots in a garage.

Maximum user privacy is achieved by, firstly, not storing any information that is not needed for operation of the IPS (least to know principle) and secondly, hashing all user-identifiable and/or sensitive information (e.g. passwords, licence plates, email addresses) before it is stored inside the database. This way, even if an attacker obtains the database, he/she will not be able to retrieve any useful information from it.

The two ways parking in the garage (i.e. with or without a pre-existing account) provide two levels of user privacy. If the user decides to park without an account, no user-identifiable information is stored in the database, except the licence plate, which is deleted upon exit of the garage. This way, it is not possible to retrieve any personal information, nor information about the user's whereabouts from the system. From the standpoint of the system, that user ceases to exist upon exit of the garage. In the other case, license plate information is coupled with the user's email address and a real first and last name, but history of the user's whereabouts are not stored (they are deleted upon exit of the garage).

Making an automatic payment requires payment information from the user. This information is hashed before it is stored inside the database, but the user can opt for a manual payment, in which case no payment details are stored in the database.

Furthermore, a garage owner is not able to query the users or license plates in its garage, only the amount free parking lots left are displayed.

Security is the last core functionality. This mainly focuses on securing the backend, because it stores all important user information, but also includes securing the mobile application and the local garage setup, such that the sensors, nor the ANPR-cameras can be hijacked.

Firstly, the connections between the frontend and the backend and between the backend and the local garage gateway are encrypted. Secondly, the server only accepts traffic coming from the local garage gateway or the mobile application, which prevents Distributed Denial of Server (DDOS)- and Cross Site Request Forgery (CSRF)-attacks. Thirdly, users can only query the information regarding their own account (i.e. their personal information and bookings), which prevents Indirect Object Reference (IDOR)-attacks and fourthly, the API can only be queried by authenticated users.

Table 1: An overview of the core functionalities of the entire system and how this is achieved on an abstract level.

Core functionality	**Achieved by**
Ease of use	• No obligatory account • Reserving parking lots • Automatic and non-automatic payments
User privacy	• Least to know principle • Hashing sensitive user information
Security	• Encrypted connections • Request origin validation • Authenticated API

3 Implementation

3.1 Parking garage

3.2 On site system

3.3 Frontend

This section describes the implementation of the core functionalities of the frontend as described in section 2.1. The frontend apllication is written in Dart, with the Flutter[1] framework of Google. Flutter is used, because its main benefits are that it can be runned on any operating system(Android, iOS, MacOS, Linux, Windows, etc.), and it provides type safety and null safety. Further supports flutter a hot reload feature, which makes it easy to develop an application. The next section will give a rough idea how the app will work when you open it.

3.3.1 First app design

The first page that users see when they open the app is the login screen, here they will have the option to login with either their existing account or to register a new one. If they choose to make a new account, then the register page pops up. On this page they have to fill in their first name, last name, licence plate, e-mail address and password. These credentials are used to create a secure account for a user that is linked to their licence plate for the automatic payment system. The user is required to confirm the password by entering the same password in another text field. This is required for lowering the chances of accidentally typing the wrong password. Once the necessary text fields are submitted, the user will be able to register their account.

As soon as the account is created, the user can login by hitting the 'Sign in'-button. When that's done, the homepage will pop up and the app will make a request to the backend server to load all the possible garages. While the app is connecting there will be a progress indicator on the screen and the user will still be able to access other buttons like the navigation bar. After all the garages have been loaded into the app, then the user can select one of the garages to make a reseredrvation. Next the reservation screen will pop up of the selected garage. On this screen the user can observe the busiest times of the day and how many spots are left in the garage. If the user is satisfied of this garage, than he/she can book a reservation. There will be the option to choose the time and day and an available spot.

For users who are not familiar with using the app or a mobile app in general, there will be a guide in the navigation bar. Furthermore, the user can see his/her statistics, profile, reservations and adjust their settings. A detailed schematic of the entire app can be found in Appendix B in Figure 5. At last there is also an option to sign out of your account.

3.3.2 Deployment

The frontend deployment should support two use cases: users who want to download the mobile application and users who want to access the website.

Due to Flutter's nature, it can run natively on all major platforms and operating systems. To make the mobile application accessible to the general public, it should be uploaded to the Google Play Store, the Apple App Store and the Microsoft Store. For the purpose of this project, the application will be installed on the devices of the team members.

The web application should be hosted on a web server, for the users to be able to access the site. The backend already incorporates a web server, namely Nginx[2]. Apart from being a reverse-proxy for the backend, it also hosts the static files (HTML and JavaScript (JS)). Figure 2 shows the deployment diagram for the frontend application. Note that the two client devices represent both options of the client of connection to our backend API.

[1]`https://flutter.dev/`
[2]`https://nginx.org/`

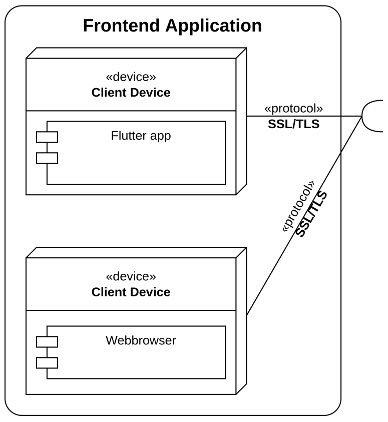

Figure 2: Deployment diagram of the frontend application.

3.4 Backend

The main functionalities of the backend are described in Section 2.1. This section describes the implementation of those core functionalities. Section 4 outlines the augments for the different software used in the backend.

The main parts of the backend described below can run on a single physical machine. The deployment of those services can happen both on a local server or on a cloud server. For the purpose of this project, a local home server is preferred, but in real-world system requires scalability which makes a cloud server indispensable.

3.4.1 Server

The server is the main entry point to the outside world of the database and the REST API and thus needs proper security measures. The encryption of traffic happens on the server, origin validation to prevent CSRF-attacks is included in the backend application (see Section 3.4.2).

Nginx is used as the main HyperText Transport Protocol (HTTP)-server in the backend. It serves as an industry standard for a fast and lightweight server. The most important feature for the backend is that it can handle Secure Socket Layer (SSL)/Transport Layer Security (TLS)-connections and redirect HTTP-requests to HyperText Transport Protocol Secure (HTTPS)-requests [Nginx, 2022]. Furthermore, the server has to redirect incoming requests to the right application, namely, the web variant of the frontend application or the backend application. This is achieved via a *proxy-pass*, which can redirect traffic from one server to another, based on certain conditions in the request (e.g. all URLs which start with `api/` are redirected to the Gunicorn Python server (see below).

Besides a web server, the backend needs a way to communicate between the web server and the actual backend application (in casu the Django application). This is achieved with a Web Server Gateway Interface (WSGI). The backend uses Gunicorn[3] as its WSGI. The main purpose of the WSGI is making the deployment more stable and faster. The former is achieved by running multiple instances of the Django application, which improves to overall availability of the system [Chakon, 2017]. Besides improving the deploy stability, the WSGI makes it possible to use an Nginx server as reverse proxy for redirecting HTTP to HTTPS-traffic. This is not possible without a WSGI.

The services above require a lot of dependencies and configuration files, which can make it tedious to set up on a remote machine. The backend is therefore deployed with Docker containers, which bundles *Docker images* with an Operating System (OS) in a so-called isolated *container*. This way, all the dependencies are packed inside the container, which eliminates the need of doing a laborious setup on the server machine. In total, there are three containers, one for the Nginx server, one for the Gunicorn gateway which runs the Django application and one for the MySQL database. The containers run as a single service with Docker Compose, which makes communication between the different containers effortless [Docker, 2022]. Figure 3 shows the deployment diagram of the entire backend.

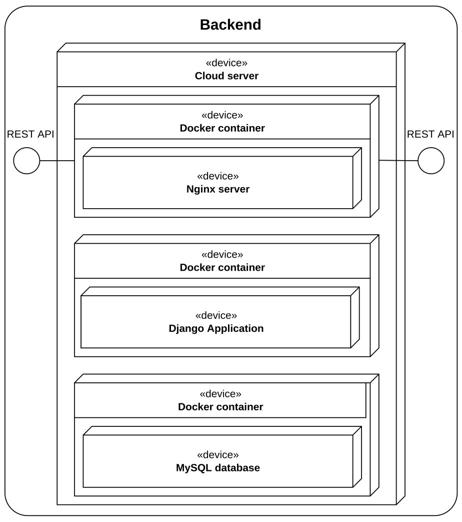

Figure 3: Deployment diagram of the backend server software.

[3]`https://gunicorn.org/`

3.4.2 Backend application

The backend server-side application is written in the Django framework. The Django framework serves as an Object Relational Mapper (ORM) for the database, which makes it much easier to query the database. The main purposes of the backend application is the retrieve the right information from the database and more importantly, validate if the user is authenticated and authorised to query that information. The database itself is a MySQL database.[4]

3.4.2.1 User authentication

User authentication is an essential part of the IPS in general, due to the fact that this is the primary defence against unauthorised querying of the API. The authentication system in the backend consists out of three *views*: registering a new user (sign up), logging in a user and logging out an existing user.

If the user creates a new account with the frontend application, their record will be added to the `users`-table, but this account is not yet functional. Upon registration an email is sent to the user with an activation link. Once the user has clicked on this link, their account will be activated and can be used to log in.

A user can log in via the frontend application which sends the entered `email` and `password` to the backend via an encrypted SSL/TLS-connection. The backend validates the credentials and if they are correct, an *authentication token* is generated and sent back to the frontend. This token has to be used in *all* API-requests which query or post data from and to the database. Furthermore, this token indicates if a user is logged in or not. If a token is present in the `knox_authtoken`-table, a user is considered logged in. To prevent attackers to log in with this token from a separate device, the number of tokens is limited to one token per user. If someone tries to log in with the credentials of an already logged in user, an error is returned. This token also makes it possible for the frontend to implement an automatic login system for the users.

From the frontend application, users are able to log themselves out. From a backend's perspective, this means deleting the authentication token from the database. In order for the logout the succeed, the authentication token has to be sent with the request. This prevents unauthorised logging out of users.

3.4.2.2 Serialization of database models

All data is transmitted in JSON-format over the API, but it cannot be directly inserted in this format in the database. One of the tasks of the backend application is *serializing* the data between the JSON- and database formats. This not only happens when data is queried from the database, but also when data is posted to the API. In this direction, the serializing function becomes more important, because it *validates* the sent data. The data has to match the requirements of the schema of the database (e.g. certain columns of the database are non-null, others have to be unique, etc.). If not, an error message is returned to the sender and the record will not be stored. This procedure is also the main defence against Structured Query Language (SQL)-injection.

3.4.2.3 User roles

All users in the database have one specific role, which is associated with a set of *permissions* for this user. There are three roles: `GENERATED_USER`, `NORMAL_USER` and `GARAGE_OWNER`. Upon registration with the frontend application, the default role is `NORMAL_USER`. In order for a user to register as a `GARAGE_OWNER`, the user has to provide the necessary legal documents which proves its ownership of a garage. The set of permissions of the `GARAGE_OWNER` is a superset of the user permission set, with the added permissions of adding and deleting garages and adding, disabling and deleting parking lots.

The `GENERATED_USER` is a special role which is created upon entering of a user without an account in the parking garage. The ANPR-cameras send registered text of the licence plate to the backend. Then the backend checks if a licence plate is already registered in the database. If not, a new dummy user, with the role `GENERATEED_USER` is created with the associated licence plate. This account only serves as a visualisation of the parameters of the user's park (e.g. duration) and has no functionalities which are found in a normal user's account. Upon exiting of the garage, the backend checks the associated role of the user; if it is a `GENERATED_USER`, the account, with the associated licence plate, is deleted.

[4]`https://www.mysql.com/`

4 Design motivation

the design discussed in this report was not chosen without careful consideration of all options. The first decision made was which micro controller to use. Common types of micro controllers such as a Raspberry Pi or an Arduino are readily available and are suitable for the project's needs. The Raspberry Pi 3B was used because the department had it available.

Secondly, the detection of entering or leaving cars can be done by cameras with motion detection software. This means that the camera is constantly running and detects whether or not there is any movement. The positive side of this method is that you don't need any sensors or other extra hardware. The downside is that it might be too much to handle for the Raspberry Pi. An alternative option is working with sensors (e.g. a distance sensor) to detect the cars. This is less heavy for the Raspberry Pi, but adds an extra cost to the garage. Another option would be to leave the cameras filming and run the algorithm directly on the video footage. But this may also be too much to handle for the Raspberry Pi. Therefore, the option with the sensors is chosen for this project.

Thirdly, the detection of available parking spaces can be done by several sensors. The most useful ones are Ultrasonic Distance Measuring Sensor (UDMS) or light sensors. The latter is more expensive and doesn't offer any extra advantages over the UDMS. Therefore the UDMSs (HC-SR04) are used in this project.

Fourthly, The backend application is written with Django, an open-source web framework, written in Python. The eventual decision was made based on the following concerns: 1) Python is known to all group members; 2) Django is a very explicit framework, which does not include a lot of magic features like Ruby on Rails does; 3) Django describes itself as the "framework for perfectionists with deadlines", which is exactly suited for the job [Django, 2022].

Lastly, The frontend application will be written in Dart, with the Flutter7 framework of Google. Other valid alter- natives were primarily JavaScript (js)-frameworks (e.g. React8 or AngularJS9). The main benefit for Flutter over the other frameworks is that it can run on any operation system (Android, iOS, MacOS, Linux, Windows, etc.), that it provides type safety and null safety (Dart is a strongly typed language) and that it supports hot reloads, which makes development much easier [Flutter, 2022]. Furthermore, two of the team members already worked with Flutter.

Of course the price of the different components played also a big part in the decision making. The prices of the individual components and the total price can be found in 2. The leftmost column gives the name of the component used in the model. The second column shows how many pieces of that component are needed. Column 3 shows the price per piece and column 4 the total price for a specific component. The budget for this project is 250 euros. Right now the design only uses a total of 153.53 euros. You can see this in the second to last row of Table 2. This means that the budget is not nearly reached with a surplus of 96.47 euros.

Table 2: Current budget state.

Component	Amount	Price/piece	Total
DORHEA Raspberry Pi Mini Camera	2	11.95	23.9
Ultrasonic Module Distance	8	3.95	31.6
MDF plates 6mm	3	2.4	7.2
Green LED lights	6	0.35	2.1
Red LED lights	6	0.33	1.98
Resistors	12	0.2	2.4
Raspberry Pi extension cable	2	4.99	9.98
Micro Servo Motor	2	7.21	14.42
Raspberry Pi 3B	1	59.95	59.95
Total Price		153.53	
Remaining		96.47	

5 Case example

Earlier, this report gave an abstract explanation of how the system should work and the more concrete implementation using the different components of both the software and the hard system. Subsequently, this section gives a real-world example of a user experience in the parking garage. Of course, in following reports, these actions are going to be made more concrete with sequence diagrams. Due to the large size of the flowcharts, they are included in Appendix D.
The process begins with the user who drives towards the entry barrier. A UDMS detects the car and sends a signal to the ANPR-camera which takes a picture of the licence plate. This picture is then analyzed by the Google Vision API. The recognised string from the licence plate is then sent to the backend with a `POST`-request to `api/plate`. The system supports two use cases: either the user has a registered account with a licence plate, or the user has not. In both cases the user should be able to use the garage. The backend checks which of the two cases the received licence plate falls in. In the former, the barrier will be opened and the `licence_plate`-table is updated to include the time of arrival. In the latter case, the backend will create a new user account and print a paper ticket with a QR-code which contains a link to the created account. With this dummy account the user can view all the information about his park. This dummy account is deleted if the users exits the garage, compliant to current General Data Protection Regulation (GDPR)-guidelines. Figure 6 in Appendix D shows a schematic overview of the entering process.

After entering the garage, the user drives to the pre-booked parking lot, in which case the occupancy of the parking lot is already set to `True` or to a parking lot of choice. In the latter case, a UDMS detects the cars, so that the Raspberry Pi can sent an API-request to backend to update the respective table. Note that only if the parking lot is booked, the parking lot is associated with the licence plate and thus with the user. Figure 7 in Appendix D shows a schematic overview of this process.

When exiting the garage, it is recommended that the user pays its ticket in advance, to make the exiting-process run smoothly, but the system also supports payments in front of the barrier for users who might have forgotten to pay.
In a similar way as when entering, a UDMS detects the car and the ANPR-camera takes a picture, which is sent to the Google Vision API for analysis. Subsequently, the recognised text is sent to the backend via the same URL. The backend can distinguish the images for entering and exiting the garage via the `licence_plates`-tables which stores whether a licence plate is currently inside the garage. This table also contains a column which indicates if the user connected to the licence plate has already paid. If this is the case, the barrier will open. In the opposite case, there are two possibilities: the user has an account which supports automatic payments, in which case the payment will happen in situ and the barrier will open consequently. In the case in which the user hasn't paid, nor has an account which supports automatic payment, a paper ticket will be printed with a QR-code which redirects the user to a payment-environment. Once the user has paid its ticket, the barrier will open. Figure 8 in Appendix D shows a schematic overview of the exiting process.

Note that the need of paper tickets isn't fully eliminated in this user flow, but is only used as a back-up system if the user doesn't have an account forgot to pay. In both cases, the user will be able to use our parking garage with almost the same features as a user who installed the application.

6 System analysis

7 Conclusion

The work of the previous weeks has led to a solid foundation on which can be built upon in the upcoming weeks. There's a concrete design of the major parts of the system, namely the frontend application, the backend server and the Raspberry Pi. Furthermore, a scale model of the parking garage has already been realised. All major functionalities of the different systems have been designed and connected to each other in theory. The main work of the forthcoming weeks is bringing the theory into practise and realising all the details of the different systems. The design will offer an answer to the problem of making a safe infrastructure that makes parking easier and faster.

8 Course integration

This project is a sequel of it's predecessors P&O 1 and P&O 2. So the most knowledge and experience that's been used for this project came from these courses. In these courses things like writing reports (in LaTeX), keeping track of a logbook, making presentations, ect. were taught. These are basic aspects that are needed for creating a good project. As mentioned earlier, the experience that's been gained from these courses is also very important. From these courses the skill of working in a team were developed, which was very important for this project and will remain important for future projects.

Furthermore, methodology of computer science was an important course for an introduction to programming and understanding complex algorithms. This course was taught in Python and this knowledge was needed for the Raspberry Pi and licence plate recognition. Along with Python, Dart was used and this language was easy to learn because of this course.

Just like methodology of computer science was used for the licence plate recognition, other courses like calculus and linear algebra were needed for neural networks. Calculus was usefull for solving the optimization problems in the neural network, to find the best solution. Linear algebra is used in the neural networks for solving large systems of linear equations.

Another course that was very useful, was technical drawing for creating our physical design in Solid Edge. In this course the skills were taught for creating a 3D-design of an object or product and understanding the 2D-drawings of it. Other knowledge that was used for the physical aspect of the project was circuit design. This was taught in P&O 2 and was used for the sensors and lights. For creating these electronic circuits, knowledge from the course electrical networks was also necessary. This course was needed for understanding how to connect different electronic components with each other.

Of course was the knowledge of all these courses not enough to make and realise this project. But it's a good basis to understand and learn new advanced topics in this field.

References

[4411, 2022] 4411 (2022). Betaal je mobiliteit met één app. [Online]. Last accessed on 11/11/2022. Retrieved from https://4411.io/nl-be/.

[Chakon, 2017] Chakon, O. (2017). Deploy Django app with Nginx, Gunicorn, PostgreSQL & Supervisor. [Online]. Last accessed on 23/10/2022. Retrieved from https://hackernoon.com/deploy-django-app-with-nginx-gunicorn-postgresql-supervisor-9c6d556a25ac.

[Decker and Hughes, 2022] Decker, B. D. and Hughes, D. (2022). Groepsopdracht voor het vak Probleemoplossen en Ontwerpen.

[Django, 2022] Django (2022). About the Django Software Foundation. [Online]. Last accessed on 22/10/2022. Retrieved from https://www.djangoproject.com/foundation/.

[Docker, 2022] Docker (2022). Key features of Docker Compose. [Online]. Last accessed on 23/10/2022. Retrieved from https://docs.docker.com/compose/features-uses/.

[Gupta, 2022] Gupta, L. (2022). What is JSON? [Online]. Last accessed on 11/11/2022. Retrieved from https://restfulapi.net/introduction-to-json/.

[Nginx, 2022] Nginx (2022). About Nginx. [Online]. Last accessed on 23/10/2022. Retrieved from https://nginx.org/.

[STATBEL, 2022] STATBEL (2022). Wagenbezit per huishouden. [Online]. Last accessed on 11/11/2022. Retrieved from https://statbel.fgov.be/nl/themas/datalab/wagenbezit-huishouden#:~:text=37%2C3%25%20heeft%20%C3%A9%C3%A9n%20wagen,koppels%20met%20een%20inwonend%20kind.

[Verheyden, 2022] Verheyden, T. (2022). Ethische hacker waarschuwt voor privacylek in parkeerapps als 4411. [Online]. Last accessed on 11/11/2022. Retrieved from https://www.vrt.be/vrtnws/nl/2022/09/08/parkeerpapps/.

Appendices

A General deployment diagram

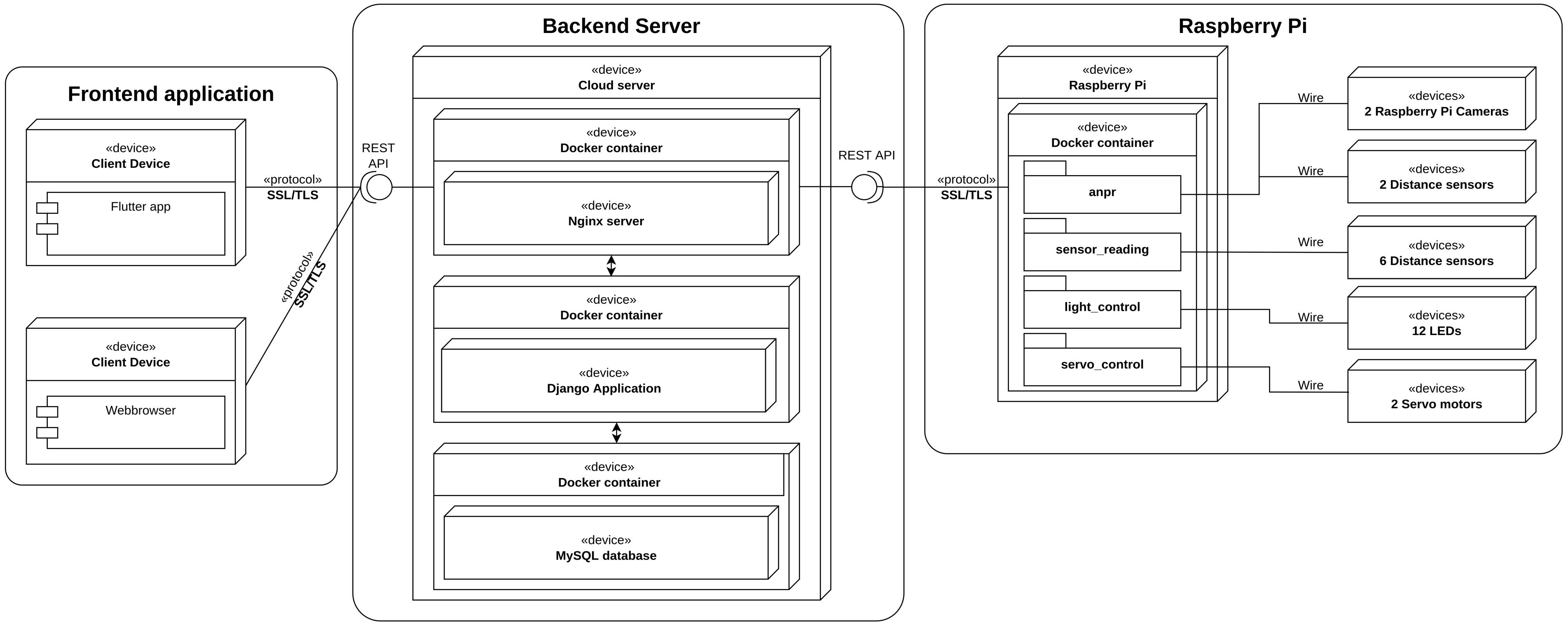

Figure 4: General deployment diagram.

B Application design

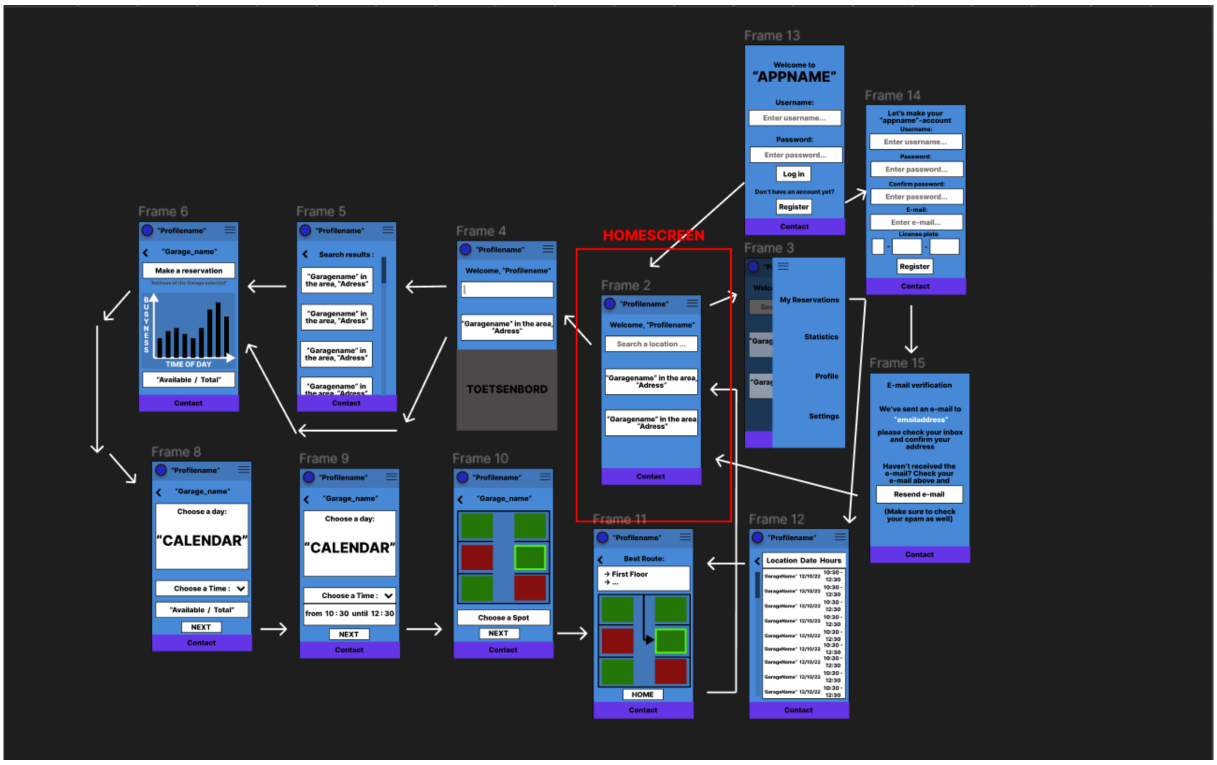

Figure 5: Current app design.

C Mechanical part list

Table 3: Overview of all used mechanical components and their model number.

Component name	Model number	Amount
Raspberry Pi	Model 3B	1
DORHEA Raspberry Pi Mini Kamera	HE0304-002	2
Ultrasonic distance measuring sensor	HC-SR04	8
MICRO SERVO MOTOR	OKY8003	2
Red LED (3 mm)	COM-00533	6
Green LED (3 mm)	COM-09560	6
Resistors (20 kΩ)	SFR2500002002FR500	12
Jumper cables	/	≈ 60
Raspberry Pi camera extension cable	B087DFJ2RP	2

D Flowcharts

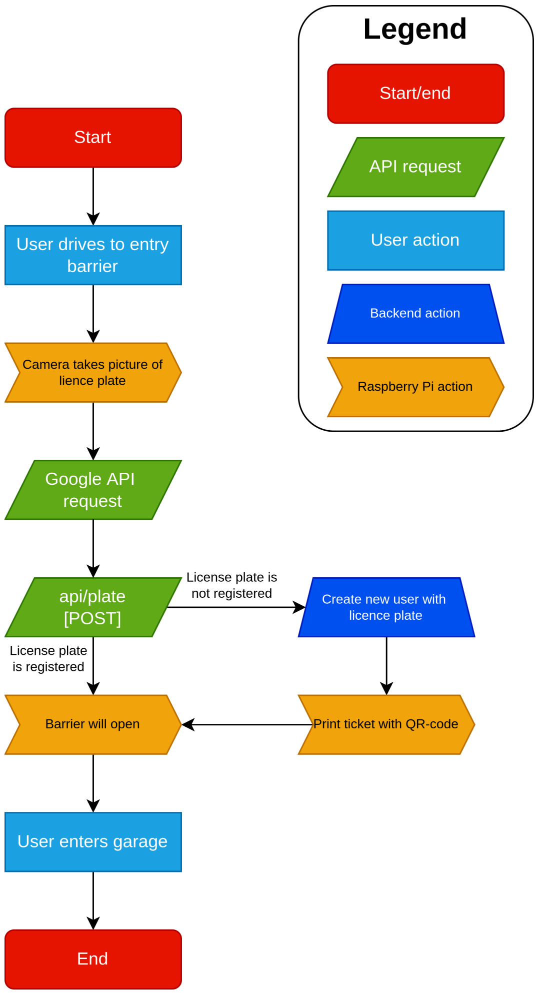
Legend
Start/end
API request
User action
Backend action
Raspberry Pi action
Start
User drives to entry barrier
Camera takes picture of lience plate
Google API request
api/plate [POST]
License plate is not registered
Create new user with licence plate
License plate is registered
Barrier will open
Print ticket with QR-code
User enters garage
End

Figure 6: Flowchart of the entering process of the garage in both hardware, software and user terms.

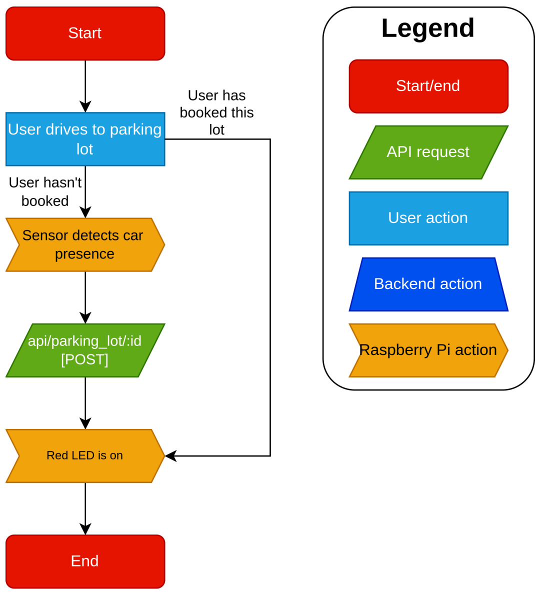

Figure 7: Flowchart of the car detection process of the garage in both hardware, software and user terms.

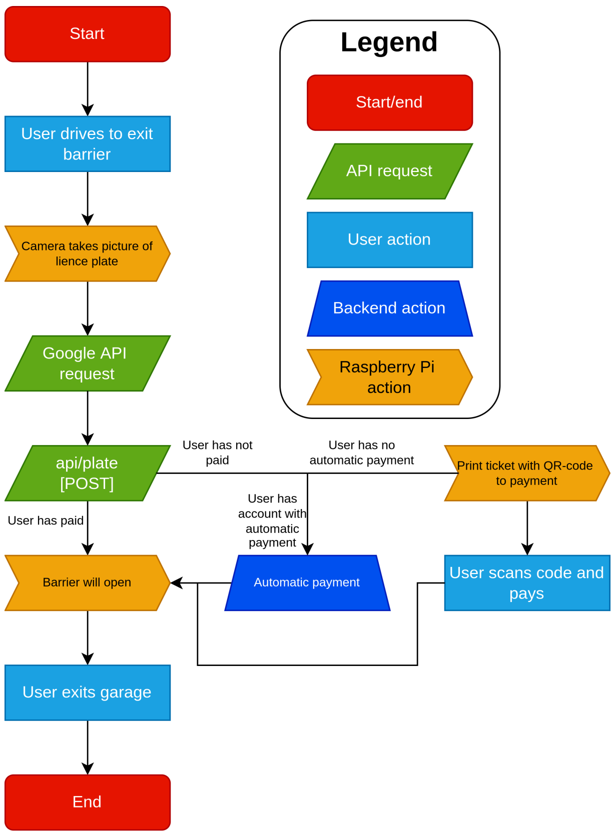

Figure 8: Flowchart of the exiting process of the garage in both hardware, software and user terms.