ABPaS: Automated Bike Parking System

Group 04: John Pravin Arockiasamy, Jayakumar Ramasamy Sundararaj, and Erisa Hoxha

Service Computing Department, IAAS, University of Stuttgart
st180270@stud.uni-stuttgart.de; st178863@stud.uni-stuttgart.de;
st181212@stud.uni-stuttgart.de

Abstract. For several years, automobiles have been the preferred choice of mobility for many commuters in the world. But this mode of transportation has been proven to contribute to various environmental problems such as air pollution, global warming, etc. As a result of various environmental awareness raised by several organizations, many people show a keen interest to move towards more sustainable transportation like biking. Also, due to the rapid growth in the city infrastructure for sustainable mobility services like biking, many people prefer biking as a primary mode of transportation. Biking is not only sustainable transportation but also a healthy way of transportation. This increase in the biking trend would eventually result in an increase in demand for bike parking garages in the near future. Thus, this project aims at automating public bike parking with a smart system that is capable of pickup/delivery of the bike from the user from a collection point and storing it in a parking garage with a bike carrier vehicle (BCV).

Keywords: Internet of Things (IoT) · Bike Carrier Vehicle (BCV) · Automated Bike Parking System (ABPaS)

1 System Introduction

1.1 Project Scope

Background Information Although people prefer to use their bikes to commute to football stadiums and public places like railway stations, parking becomes a huge problem to account for in such scenarios, thereby forcing people to consider other forms of transportation [1]. There are also external issues with safety: i.e., protecting the bike from climatic conditions, theft, and vandalism. The Automated Bike Parking System (ABPaS) aims to address this issue by automating the parking process thereby reducing the hassles of parking and making the bike commuting experience a breeze. [1]

[1] https://github.com/JohnPravin97/Automated$_{B}ike_{P}arking_{S}ystem - ABPaS$

Problem Statement To develop a system with a parking garage facility that comprises monitoring systems such as fire alarms, proximity sensors, and a smart database management system for parking and ticketing. The system also comes with a bike transporter (Bike Carrier Vehicle) that acts as a delivery/pickup vehicle. This vehicle is capable of collecting the bike from the user, storing it in the garage, and delivering it back to the user upon request. The system also offers a GUI that can be used by the facility manager that provides details on the status of parking lots, the status of BCV, and emergency scenarios like fire accidents. The entire process on the user end can be managed via a mobile app that keeps the user updated on the status of pickup/delivery and other data about their bike.

1.2 Project Goals

The goal of this project is to provide a secured automated parking system that safeguards the bikes from environmental calamities and ravaging human activities and reduces the bike parking and retrieval time.

2 System Analysis

2.1 System Outline

Nowadays, bike parking becomes increasingly important due to the more ecologically oriented traffic habits. Hence, this project is being developed to provide users with an automated parking system for their bikes. First, the user is authenticated through a mobile application where he/she is provided with the availability status of parking garages. Upon selection of a parking garage, a Bike Carrier Vehicle (BCV) inside the facility will automatically navigate towards the door and the entry/exit door is opened to allow the BCV motion. Once the user places his/her bike on the BCV and acknowledges it on the mobile application, the BCV carries the bike to the chosen parking garage and updates the occupancy of the garage in the system and the entry/exit door is closed. The above procedure is repeated if the user wishes to take back his/her bike. While withdrawing the bike, instead of choosing the parking garage, he/she needs to acknowledge the retrieval process in the mobile application. Meanwhile, any discrepancy occurring during the inward and outward movement of BCV is monitored and reported to the facility manager. In the end, based on the parking duration, the amount is charged to the user accordingly. Also, the emergency conditions like fire tragedy, obstacles in the path, etc. in the facility are continuously monitored and notified to the facility manager via buzzer. In addition to monitoring the emergency situations, these data are sent to the AI planner. The AI planner then provides commands to halt the BCV's process and alerts the facility manager and the user. It drives the actuators that are responsible to work in these emergency situations. This process runs automatically without any manual interference.

2.2 User Stories

1. The user should be able to sign up/log in to the mobile application
2. The user should be able to find the availability status of the parking garages.
3. The user should be able to choose the available parking garage for parking his/her bike.
4. The user should be able to park the bike on a bike carrier vehicle (BCV) which is automatically navigated from the chosen parking slot to the entry door.
5. Upon the user placing his/her bike on the bike carrier vehicle (BCV), it should automatically navigate back.
6. The user should be able to find the duration of the parking and the corresponding parking charge on the facility mobile application.
7. The user should be able to make payment of the parking charge via the facility mobile application.
8. The facility manager should be able to monitor the analytics of the availability status of parking slots, user details, parking charges, etc.
9. The facility manager should be able to monitor the emergency conditions like fire tragedies, and obstacles in the bike carrier vehicle (BCV) lane.
10. The facility manager should be able to monitor the temperature and humidity conditions inside the facility.
11. In case of emergency, the facility manager is alerted through a buzzer. And the necessary actions for the actuators should be automatically taken.
12. The facility entrance/exit door should be opened/closed following the motion of the bike carrier vehicle (BCV).

3 System Architecture Design

Fig. 2 shows the architectural design of the ABPaS. The physical layer, the ubiquitous layer, the reasoning layer, and the user layer are the four levels that make up the architecture of the project. The physical layer, which comprises of sensors, actuators, and a standard gateway, sends/receives data and descriptions to/from the ubiquitous layer. The ubiquitous layer gathers sensor raw data and descriptions via a standard gateway in the physical layer and compares them with the knowledge base repository which maintains descriptions of the available service types, configuration semantics for the actions, and the registry of active device instances at any given moment. The context information from the knowledge base repository is transferred to the reasoning layer, where decision-making happens. This decision-making system is basically an AI planner. The reasoning layer interacts with the user via the user layer which provides the user with information about available service activities from active devices and receives the goals. The decision-making module takes the user goals from the user layer and fulfills them by putting together acceptable compositions of existing services. When a goal is specified by the user, the decision-making module develops a plan, which comprises service operations whose execution changes the state of the environment to meet the user specifications. The developed plan is

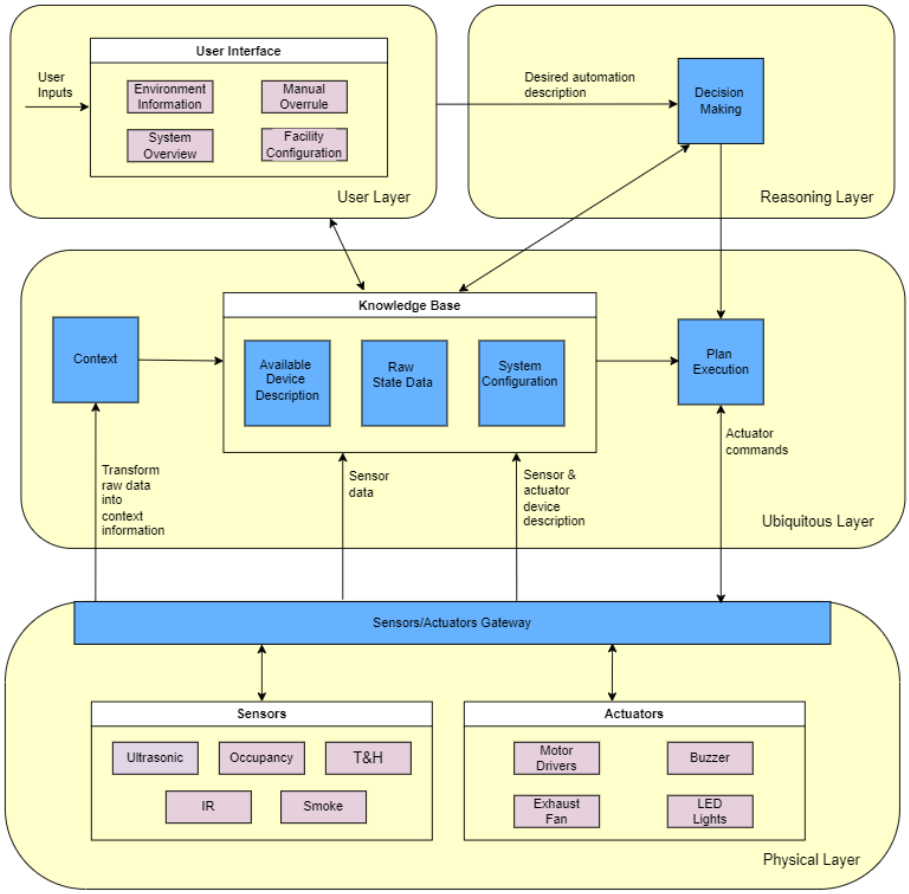

Fig. 1. System architecture diagram of ABPaS

then provided to the ubiquitous layers execution module, which subsequently sends the orders to the actuator in the physical layer over the standard gateway. In this project, the user interface (UI) is distributed into mobile applications for end-users and monitoring graphical user interface (GUI) for the facility manager.

4 System Implementation

4.1 Sensors and Actuators

IR Sensor: The IR sensor (TCRT5000) used in this project is an active reflective infrared sensor that comprises an infrared emitter and a photo-transistor. This sensor can continuously detect the presence of a stationary object by emitting an infrared beam of a short wavelength and detecting the reflected beam. This sensor is used by the bike carrier vehicle (BCV) to determine its path from the parking garage to the entry/exit door and vice versa. Basically, a black-colored path is laid from the entry/exit door to the garage which can absorb the infrared beam preventing any reflection of the beam to the photo-transistor. Thus, this sensor helps BCV to follow the black-colored path. There are two IR sensors incorporated by which forward, left, and right motions are calculated and fed to BCV.

Ultrasonic Sensor: The ultrasonic sensor (HC-SR04) used in this project is a proximity sensor that detects an object/disturbance in the BCV's path. This sensor emits ultrasonic sound waves and measures distance by calculating the time taken to receive the reflected waves. If an object is encountered within a short distance in the BCV's path, the ultrasonic sensor detects it and triggers the controller to raise an alarm.

Temperature Sensor: The sensor (DHT-22) is used in this project to measure the temperature and humidity of the facility. This sensor is built-in with a thermistor and a capacitive sensor to measure the temperature and humidity of air respectively. The output of this sensor is a voltage value describing the temperature and humidity values. This sensor is mounted inside the facility to monitor and alert the system about the rise in temperature. The humidity value is used for monitoring purposes.

Occupancy Sensor: The occupancy sensor is used in this project to detect the presence/movement of an object within its provided range and transmits the signal to the control unit. In this project, we have used two occupancy sensors (one per parking garage) to detect the availability status of the respective parking garages. And let the user know about the availability of parking garages in the mobile application.

Raspberry Pi: Raspberry Pi is a single-board computer used in this project as the main computing device. It can run multiple programs at once and can act as a server to host other systems. In comparison with other computing devices, raspberry pi has the high processing power and provides much more interfaces. In addition, it also supports python which allows applications to build on it.

NodeMCU ESP8266: NodeMCU is a self-contained System on Chip that can act as a low-cost WiFi module. In this project, NodeMCU is used as the main controller of the BCV to collect data from the sensors in BCV and publish it to the raspberry pi. An indirect communication is established between the NodeMCU and Raspberry Pi.

Motor Driver: The motor driver (L298N) is a dual H-bridge driver that allows the control of the speed and direction of two DC motors at the same time. It is possible to connect motors that have working voltages of range 5v to 35v. The module has a 5V regulator which is either enabled or disabled using a jumper and the 5v pin can be used as an output to power the controller and sensors. In this project, this motor driver is mounted on the BCV to control the speed and direction.

DC Motor: DC Motor is an electrical motor that converts direct current electrical energy into mechanical energy. In this project, two motors are used on the BCV for locomotion.

Stepper Motor: Stepper Motor is also an electrical motor that converts direct current electrical energy into mechanical energy by dividing a full rotation into a number of equal steps. In this project, a stepper motor is used to open/close the facility door.

Exhaust Fan: The exhaust fan is used to regulate the airflow in this project when the temperature exceeds the predefined threshold value.

Buzzer: A buzzer is mounted inside the facility to alert the system and surroundings in case of an emergency. The emergency situations are 1. if the temperature inside the facility increases above the predefined threshold value. 2. if the BCV encounters an obstacle in its path.

4.2 Sensing and Processing

The whole process is initiated with a user logging into a mobile application. Once a user logs in, he/she is able to view the availability of the parking garage that is sensed with the help of occupancy sensors mounted inside each garage. And the availability status is locally indicated with the help of LEDs. After the

user selects a parking garage to park his/her bike, he/she is notified to start the process of bike parking. The data from the mobile application and Raspberry Pi are hosted on an external server to be able to communicate with each other using HTTP requests. Once the user starts the process, the start command is received by the Raspberry Pi and it notifies the NodeMCU to start the movement of BCV. Then, the BCV from the respective parking garage follows the path to the entry door where the user is waiting with the bike to mount onto the BCV. Upon the user mounting the bike onto the BCV, the BCV follows the path back to the parking garage, and then the parking garage availability status is sensed by the occupancy sensor and changed to unavailable. This process is repeated once the user requests the retrieval of the bike.

This communication between Raspberry Pi and NodeMCU is indirect and is implemented using the MQ Telemetry Transport (MQTT) protocol, which works on publishing and subscribing. The MQTT Protocol is a commonly used protocol in the Internet of Things (IoT) networks and has become an OASIS standard. This protocol message format is very efficient, with each message sent containing only essential information in the couple bytes of the packet. This protocol is ideal for systems where low bandwidth is desired due to the small amount of overhead and the method of distributing messages. In the MQTT protocol, there is a broker through which the message is published/subscribed. The Mosquitto package is used as a broker for the MQTT protocol as it is open-source. The Mosquitto package has good flexibility during the configuration of the server. Hence giving a holistic experience for the user and the system.

The sensor data from the IR sensors and ultrasonic sensor from the BCV are fed into the NodeMCU which is then passed to Raspberry Pi through the MQTT publish method. Then, this data is provided to the AI planner. Similarly, the Raspberry Pi collects the DHT sensor and occupancy sensors values and provided them to the AI planner.

A GUI is developed for the facility manager to monitor. The GUI displays the temperature inside the facility, the availability of parking garages, and the status of BCV. If an emergency situation arises, the facility manager is notified through this GUI.

4.3 AI Planning

An AI planning system takes the problem formalization, or model, as input and uses problem-solving techniques, such as heuristic search, propositional suitability, or others, to work out its solution. It transforms the model into a search space or logical reasoning problem, creating heuristics to solve it efficiently. The planner does not know or need to know what the formal problem description is about. It can be applied to any problem that can be expressed in the modeling language, though, of course, not every planner will be able to solve every problem that can be given to it. This property of planners is known as domain independence. In order to make a considerable relevant interaction between the sensor value and AI planner, we have used the context entity whose output data is fed into the AI planner for further processes.

In this project, we have used Planning Domain Definition Language (PDDL) as the problem description language. A PDDL specification of a planning problem consists of two files: the domain file and the problem file. In the domain file, we have described the complete invariant rules of our world model, like object types, conditions of the world state, and possible actions to perform. Also, it contains all the actions that can possibly be occurring in this project such as BCV movement commands like forward, right, left, and stop, emergency actuation situations, entry/exit door movement, etc. Each action mentioned in the domain file is achieved only when the preconditions of the particular action are satisfied. The problem file is based on the domain file, which describes one concrete problem at a time, specifying the objects which are part of the problem, an initial state, and the goals to fulfill. In the problem file, the initial state is dynamically changing over time and all goals are mentioned as a stationary object which is outputted by the AI planner based on the initial state.

In this project, we used fast forward (FF) as an AI planner which is a domain-independent planning system. Fast forward is publicly available under the GNU license. In order to use the FF AI planner to provide a plan for a particular problem, we have provided the two above-mentioned domain and problem files. we have used the "REQUESTS" library from python to post the domain and problem files to the FF AI planner. The output from the FF AI planner is a JSON file containing the action to be taken for the problem based on the initial state mentioned in the problem file.

4.4 Actuation

With the result from the AI planner, the actuation is taken towards the expected outcome. In the case of BCV operation, the output command from the AI planner is used to drive the BCV along the path. The motor driver, mounted on the BCV, is coded based on the output from the AI planner. For example, if the command is 'forward' from the planner, then both the motors rotate forward driving the BCV to move forward. And, if the command is either 'left' or 'right', one of the motors rotates forward and another rotates backward accordingly driving the BCV to turn left or right. And, if the command is 'stop', then the BCV stops. The BCV stop motion is triggered in three instances namely when the BCV reaches the respective parking garage or when the BCV reaches the entry/exit door or in case of any emergency. The goal is to drive BCV back and forth in the black-colored path. The BCV is powered through a 9v battery. In the case of facility operations which are controlled by the Raspberry Pi, An buzzer is mounted on the facility near the door to indicate any emergency situations. With the output command from the AI planner, the exhaust fan mounted inside the garage is triggered to regulate the air inside the facility in case of a rising temperature. And, the door motion is activated to open/close based on the output from the AI planner.

5 System Overview

Below Fig.2 represents the overall system overview of this project. Each individual block in the system is integrated with the other and tested. When a new user wants to park his bike, he/she is asked to sign-up for the mobile application. Once signed up, the user receives a unique customer ID which can be used to log in later. And upon logging in, the user is displayed with the customer ID on the front page of the application screen. The application provides the user with the available parking slot where he/she can choose an available parking slot to park his/her bike. This garage availability data is fetched from each occupancy sensor mounted in parking garages via the Raspberry Pi and is posted to the server using HTTP requests. The mobile application fetches this data from the server and displays it in the user interface. Upon selecting the parking garage, the mobile application loads to another interface where the user has the command to start the process.

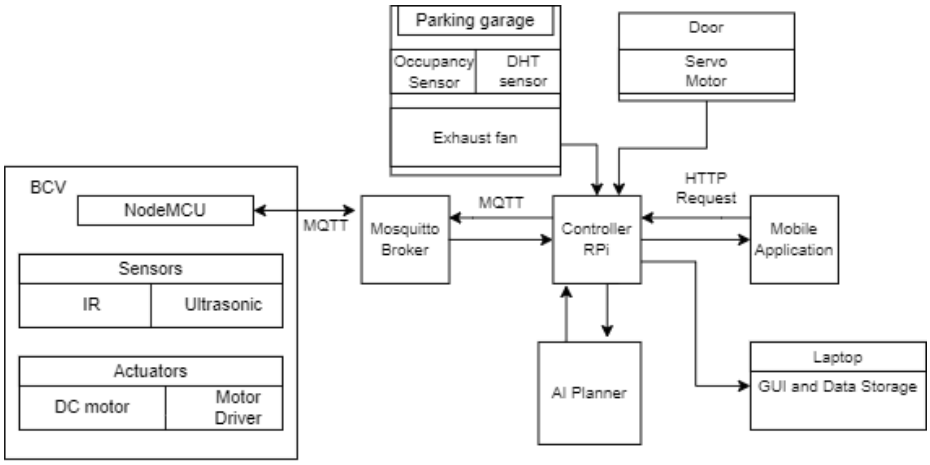

Fig. 2. System Overview diagram of ABPaS

On clicking the start button in the application, this start command is posted data to the server from where the Raspberry Pi fetches the data and commands the BCV to start the motion from the respective parking garage to the entry door. Once the BCV motion is started, the sensor data is collected in the NodeMCU and through MQTT communication protocol reaches the Raspberry Pi. These data are in turn provided to the AI planner. The AI planner provides commands to the BCV to follow the path to reach the door. The entry/exit door motion is activated to open the door for the BCV. Once the BCV reaches the door, it is notified to the user through the application. Then, the user places his/her bike on the BCV and presses the finish (loading the bike) button in the application. On receiving this finish command from the application, the BCV

makes its return journey to the selected parking garage and the entry/exit door is closed. Meanwhile, a timer starts counting the parking time. Upon BCV reaching the respective garage, the interface in the application changes into a new one displaying a return button, and the respective parking garage status is set to busy, and the parking garage LED is led to show busy status. Once the user hits the return button, the above process is repeated and the BCV drives towards the exit door, and the user can take his/her bike back from the parking garage. In the meantime, the timer stops, and the cost is calculated based on the parking time, and the interface in the mobile application takes the user to a payment page where he/she can pay the parking cost. During this process, if emergencies like fire or obstacles in the path are encountered, then the BCV comes to a halt and the Raspberry Pi with help of a planner alerts the user and facility manager through an alarm. A GUI is developed and locally hosted on a laptop to monitor the status of the facility.

5.1 Graphical User Interface

A graphical user interface (GUI) is developed to allow the facility manager to monitor the activities happening inside the facility. The GUI is developed using a python based open-source application framework called 'Streamlit'. Streamlit is an application framework wrapped around the python pandas library to help the customer build easy data analytic webapps. Streamlit is preferred over other tools because it is compatible with major python libraries. It is mainly used to help machine learning and data science teams to visualize their work easily.

Fig. 3. Sensor and Actuators Data Page

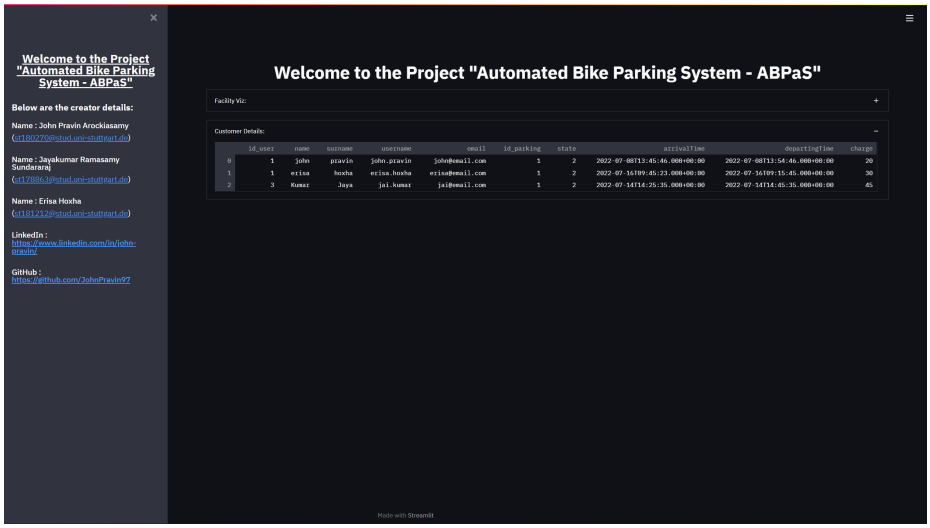

Fig. 4. Customer Information Page

In this project, we have hosted the GUI on a local host (namely a facility manager's laptop) to avoid outsiders to access the confidential data. By subscribing to the MQTT topics, we have collected the data from the BCV and the Raspberry Pi and have stored them in JSON format on the laptop which is then converted to data frames using the pandas library. This GUI displays sensor data, the commands from the AI planner that drives the BCV, the customer details like customer ID, the duration for which he/she has parked his/her bike, and the respective parking charge.

As side information, we have also provided details of the creators of this project on the GUI.

5.2 Mobile Application

A mobile application is developed as a part of this project. It is built using Android Studio and is written in Kotlin language. This mobile application has 7 different interfaces starting from the user sign-in/login page to the payment page at the end. The mobile application has its own database where the customer details are stored. Selected data is also hosted on an external server so that the communication between the Raspberry Pi and the mobile application takes place seamlessly.

The communication between the mobile app and the Raspberry Pi is done through API from a backend application. For the backend application, we have used Spring Book, and Hibernate in Kotlin. Also, we have used the MySql database on this project. The mobile application represents the user's choice with the parking space. From the phone, the user can select a parking spot to

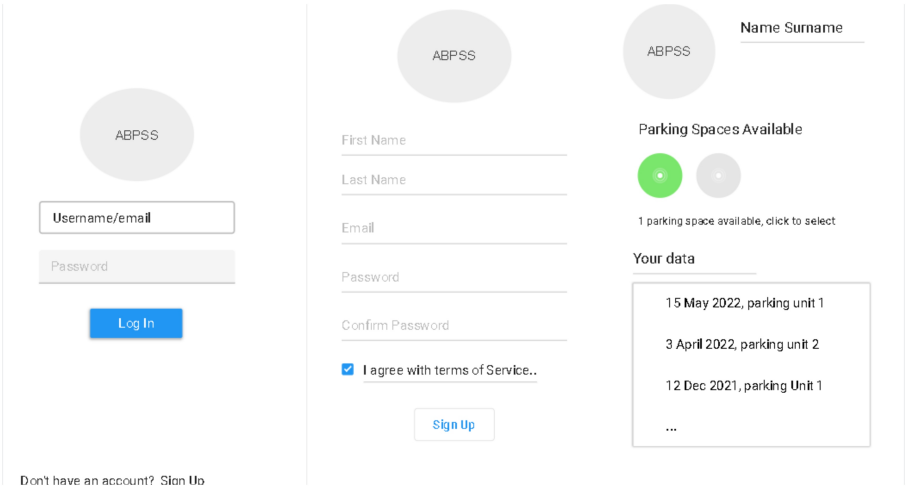

Fig. 5. App: Login Page, Sign-up Page and Main Page

park his/her bike, can get information about the parking duration, and can pay the charge for the parking.

6 Discussion

Many sections of this project are implemented in accordance with the standards established by the design, enabling proper interoperability between them. Smart Devices can execute their jobs, communicate, and update correct data to the server. They have their functionality monitored because they are connected to a central controller (the Raspberry Pi). To verify that the IoT system operates as intended, each component has undergone testing. All connection settings for the ABPaS facility that were evaluated by the MQTT server were successful. Each device linked to the controller was examined. Plans created by the AI planner and the subsequent sequence of actions to be done broadcast to the controller and it was discovered that they were activated for all devices with very little latency in just a few seconds. While building this system, we encountered some shortcomings, which were rectified.

As planned in the initial phase, we were able to build the system. But there are some improvements that can be done. In the earlier stages we tried to incorporate multiple BCVs for each parking garage so that if a parking garage is selected, the BCV allocated to that parking garage will come and collect the bike. Also in the AI planner, we tried to create a logic for multiple BCVs to run smoothly in a single path without colliding with each other. Unfortunately, we could not make multiple BCVs run simultaneously. Similarly, we tried to create separate doors for entry and exit to provide further flexibility to the user to select the door which is nearer to them along with parking garage selection. We

thought of building a big working model where multiple doors can be installed. Due to resource constraints, we were not able to erect multiple doors. Nevertheless, these were some improvements that would have increased the complexity of the project. Apart from this, the ABPaS system works as expected.

7 Conclusion

The ABPaS system is developed as a proof-of-concept that has been thoroughly tested and reviewed in its physical model version both from a technical and user perspective. This system, if implemented on a large scale, has so many advantages. Compared to conventional parking systems, ABPaS is inherently much more secure and safe because they remove drivers and pedestrians from the parking area. This means that there is no possibility of stolen bikes or accidents inside the parking area. On a large-scale implementation note, ABPaS is generally constructed underground. This saves space and protects bikes from external calamities.

8 Acknowledgements

The authors of this project would like to thank Professor Dr. Marco Aiello and tutor Dr. Ilche Georgievski and Ebaa Alnazer for their support, feedback, and valuable teachings. Thank you for guiding us through the project and helping us achieve the desired results.

References

1. Bicycle parking in dense areas. https://nationaler-radverkehrsplan.de
2. Plans to construct bicycle parking facility. https://www.givt.de/index.php
3. PDDL - Introduction and dynamic object creation. https://planning.wiki/ref/pddlplus/domain
4. Fast Forward planning system. https://planning.wiki/ref/planners/ff
5. Streamlit - Official documentation. https://docs.streamlit.io/
6. Web reference - construction of line following robot. https://create.arduino.cc/projecthub/Muhammad-sheraz/artificial-intelligence-ai-based-line-following-robot-b20235
7. NodeMCU - Official Documentation. https://nodemcu.readthedocs.io