

BILKENT UNIVERSITY

DEPARTMENT OF COMPUTER ENGINEERING

CS492 -SENIOR DESIGN PROJECT II

Parkhound: Parking Spot Detection and Navigation System

FINAL REPORT

Group Members

Arda Türkoğlu

Ege Turan

Berkin Inan

Görkem Yılmaz

Ata Coşkun

Supervisor

Varol AKMAN

Jury Members

Uğur DOĞRUSÖZ

Ercüment ÇIÇEK



Contents

1	Introduction	3
2	Requirements Details	3
2.1	Functional Requirements	3
2.2	Non-Functional Requirements	4
2.2.1	Usability	4
2.2.2	Reliability	4
2.2.3	Security	4
2.2.4	Performance	5
2.2.5	Extendibility	5
2.2.6	Portability	5
2.3	Pseudo-Requirements	5
2.3.1	Implementation Constraints	5
2.3.2	Economic Constraints	5
2.3.3	Professional and Ethical Constraints	6
2.3.4	Time Constraints	6
2.3.5	User Experience Constraints	6
3	Final Architecture and Design Details	6
3.1	Overview	6
3.1.1	Client Module	6
3.1.2	Server Module	7
3.1.3	Vision Module	7
4	Development/Implementation Details	8
4.1	Client	8
4.2	Server	8
4.3	Computer Vision	9
4.3.1	CNN Model Decision and Model Details	9
4.3.2	Transfer Learning Implementation	9
4.3.3	Training Dataset and Training	10
4.3.4	Display Manager and Loading Trained Model for Classification	10
5	Testing Details	10
6	Maintenance Plan and Details	12
7	Other Project Elements	14
7.1	Consideration of Various Factors in Engineering Design	14
7.1.1	Public Health	14
7.1.2	Public Welfare	14
7.1.3	Public Safety	14
7.1.4	Public Information Security	14
7.1.5	Global Factors	14
7.1.6	Social Factors	15
7.1.7	Cultural Factors	15

7.1.8	Environmental Factors	15
7.1.9	Economical Factors	15
7.2	Ethics and Professional Responsibilities	16
7.2.1	Ethics	16
7.2.2	Environmental Responsibilities	16
7.2.3	Economical Responsibilities	16
7.2.4	Social Responsibilities	16
7.3	Judgements and Impacts to Various Contexts	16
7.4	Teamwork Details	16
7.4.1	Contributing and Functioning Effectively on the Team	16
7.4.2	Helping Creating a Collaborative and Inclusive Environment	18
7.4.3	Taking Lead Role and Sharing Leadership on the Team	18
7.4.4	Meeting Objectives	19
7.5	New Knowledge Acquired and Applied	21
8	Conclusion and Future Work	22
9	Glossary	22
10	User Manual	23
11	References	37
12	Appendix	39

1 Introduction

The purpose of the project is to help people to find a parking area and guide them showing empty and occupied parking slots.

The project idea comes from searching for a parking problem. This problem is especially valid in crowded cities or locations which have an intensive population. We see that some people are tired of finding a parking space in Ankara and Bilkent and we try to solve this problem. Parkhound shows the near parking areas and its available parking lots using the camera's input which are located in those parking areas. In addition to parking availability, there are two other aspects of outdoor vehicle parks. One of them is the security of the parking area which includes car security and protecting the area of the property. It is not always easy to control a large area separated for cars and this uncontrolled area can be a place for illegal activities. Secondly, some outdoor vehicle parks demand some regulations about parking, such as placing the car between the white lines. Normally, this kind of problem is handled by the security personnel in the foundation. They are walking around and making controls but again, this is not easy and has a high percentage of error-prone in the large outdoor vehicle parks. For almost a decade, many foundations have used sensor-based control parking slot maintenance systems in closed car parking spaces. However, the management of the outdoor car parking spaces is not easy for sensory systems due to the safety of the devices and the placement difficulties faced by the producers. Also, these sensor systems cannot handle many security issues and regulations about parking. Therefore, we came up with an idea of creating a computer vision system named Parkhound to maintain the parking spots in outdoor vehicle cars.

This project requires careful planning, efficient design and analysis stages because there are 2 fundamental parts which are client and server and these should work in synchron and fast. The project requires video and image data for training the system and testing. Tests were planned to be done in Bilkent Park. However, testing could not be in Bilkent Park due to COVID-19. In addition, real world testing using cameras could not be in anywhere because of COVID-19. However, PKlot[3] and CNR Parking[4] datasets are used to train parts and testing parts. These datasets are meeting the needs of the project.

All fundamental features such as parking lot detection, login and registration of users, mapping the parking area and showing the available and occupied parking lots to the users are implemented. In the next iteration, we will add these futures: favorite parking areas, drawer menu. The accuracy of the computer vision model for parking lot detection is around %98 which is very successful.

Users who use the Parkhound app will be able to find the nearest parking areas and see the number of available slots of these parks and monitor the map of these parks and their empty and occupied parking lots. It provides a satisfying experience to users while they are driving and searching parking space.

This report provides information about the final version of the Parkhound application. It includes information on the Requirements Details, Final Architecture and Design Details, Development/Implementation Details, Testing Details, Maintenance Plan and Details, Other Project Elements followed by Conclusion and Future Work.

2 Requirements Details

2.1 Functional Requirements

- There are 2 types of users. One of them is maintainers such as park managers, securities and line drawers to standardize the park features. The other type of users are regular users such

as drivers.

- Users are able to register to our application by providing their personal email addresses or company email addresses.
- Regular users should not see information about other cars. Only they can see available slots in the parks.
- Users should be able to see which park slots are available and occupied for that park.
- Park spot detection systems should consider the weather and time of the day.
- Users should be able to register to our application's system by filling the required information.
- Users should be able to update their user information.
- Users should be able to find the parks around while they are driving.
- Users can add park areas to their own favorite park list.
- Admins should be able to add parks to the system.
- If there is a problem in google maps, the system should continue to show registered parks' locations at least.
- Low-cost cameras should be enough to detect parking slots and whether they are available or not.

2.2 Non-Functional Requirements

2.2.1 Usability

- Parkhound should provide two types of interface and for the regular users it should have a user-friendly interface that only specify necessary information for parking to empty slots.
- Parkhound should provide more information offering interfaces for the admins.
- The application should include an explicit user manual that demonstrates how to use Parkhound.

2.2.2 Reliability

- The application needs to be stable and avoid any interruptions/crashes.
- The application needs to provide high accuracy on determining parking status of the vehicles.
- The application needs to provide real-time data of the available parking spots.

2.2.3 Security

- The application needs to secure user information from any possible threats.
- Any information about users and their parking lots and cars will not be shared.

2.2.4 Performance

- Load time of the application should be low.
- The image recognition and connecting to server processes should be optimized so that users can quickly and easily look for the information they can access.

2.2.5 Extendibility

- In order to improve and modify our project, the data sets that we are going to use will be extendable.
- The application should be available on multiple platforms

2.2.6 Portability

- Any personal computer with a browser and devices with an Android or iOS operating system will be able to run Parkhound.

2.3 Pseudo-Requirements

2.3.1 Implementation Constraints

- Parkhound will rely on real-time parking slot detection and recognition. Hence, it will use Computer Vision techniques.
- Open source libraries and frameworks such as Tensorflow, Keras and PyTorch will be used for development processes.
- Python will be used to develop the Neural Network.
- React will be used to build the mobile web and desktop application.
- The application will need access to the GPS location.
- AWS (Amazon Web Services) will be used to keep host-server on cloud.
- GraphQL will be used to communicate with the database.
- PostgreSQL will be used to manage all database functionalities.
- GitHub will be used for Version-Control.
- Mobile app works on Android 4.1 or higher.

2.3.2 Economic Constraints

- Libraries and frameworks will be open source.
- Low-cost cameras will be able to handle the detection and recognition processes of parking slots. Prices change cameras to cameras.
- GitHub is a free Version Control tool so there will be no expense for Version Control.
- One-time-only fee for publishing the app on Google Play is 25 USD.

2.3.3 Professional and Ethical Constraints

- User data will not be shared with any other user or company or third party software.
- No ads will be displayed.

2.3.4 Time Constraints

- Project Specifications: Monday, Feb. 24, 2020
- Analysis Report: Monday, Mar. 23, 2020
- High-Level Design Report: Monday, May 15, 2020
- Low-Level Design Report: Monday, Oct. 5, 2020
- Final Report: Thursday Dec. 17, 2020
- Presentations Demonstrations: Dec. 21 - 24, 2020

2.3.5 User Experience Constraints

- The application will require internet connection.
- The app will be launched in English since it is more universal.

3 Final Architecture and Design Details

3.1 Overview

Our application consists of 3 main modules, client, server and vision module. Client module is the front-end of the application which the users will interact with via mobile and desktop. Server module has the database and the API server that interacts with the database. Vision module analyzes the parking lot images and marks the unoccupied parking spaces on the database. A diagram for the overall architecture of the application can be found in Figure 1, in Appendix.

3.1.1 Client Module

Client module is a progressive web application written in JavaScript that can be used by both mobile and desktop users via browser. Mobile users will be able to see the parking lots and navigate to them while desktop users can take administrative actions. We decided to go with a progressive web application as the application does not require any native functionalities and to remove the need for downloading an application from an app store.

Our team members were experienced with React, a JavaScript framework which allows creating reactive and stateful web applications. As our application needed frequent changes in the displayed pages, we decided that React will help us develop much faster.

In React, we use a component based architecture that helps us construct each page from its building blocks. As we are hosting the mobile and desktop on the same files, we can re-use components between each other. Overall, React provides a better developer experience on the client side.

To show the map on the mobile, we use the Google Maps API and Google Directions API to get the driving directions between two points.

3.1.2 Server Module

On the server module, there are two main submodules, the database and the GraphQL server. The database is a PostgreSQL database which we define its schema with JavaScript using a library called Knex.js. The second submodule is the GraphQL server, which is generated by Hasura, a middleware that sits on top of the database and generates all the required endpoints for reading and writing data as GraphQL endpoints.

Both of these submodules are run on Docker containers to ensure encapsulation and easy deployment. Separate Docker containers are ran with docker-compose to enable communication between the containers. On Docker, we also run Adminer, which is a lightweight database management tool that allows us to view the tables quickly.

3.1.3 Vision Module

Classical image processing methods may give insufficient results for image classification. On the otherhand, Deep Learning provides power-full classifier models. Thus, we trained deep learning model to identify the parking lots. The networks that we use is described as convolutional neural networks which are very useful machine learning algorithms to classify the images. In our task, we needed to identify the vehicle in the parking lot. But training a network from scratch is a bit complicated and very time consuming. Thus we used Transfer Learning for development process. Further explanation about Transfer Learning will be given in the Development/Implementation Details section. In this part, we will explain architecture and design details about Deep Learning, Computer Vision task and arrangements about the parking area.

- **Deep Learning, and Computer Vision** A Convolutional Neural Network (CNN) is a Deep Learning algorithm that can take in an input image, assign importance to various aspects/objects in the image and be able to differentiate one from the other. The preprocessing required in CNN is much lower as compared to other classification algorithms. While in primitive methods filters are hand-engineered, with enough training, CNN can learn these filters[8]. Filters are very crucial in the identification process because they reduce the size of the data acquired from the image. But while doing that, they keep the important features.

As we explain the reasons for selecting the CNN, it is the main algorithm for that job. However, we also needed cost efficiency. For that reason, we have selected Transfer Learning which preserves the high-level layers of the network for the feature selection and changes the classifier layers of the pre-trained model. With that method, training becomes cost efficient and result efficiently. Therefore, we decided to use a Convolutional Neural Network as target architecture and Transfer Learning as the main training method for our task.

The model decision is very important in deep learning because of the complexity of the feature acquisition over the images. Also, layer count is very crucial in the classification task because of the computation cost. We selected the Resnet50 model for training. Its architecture provides good computation cost and good classification results.

- **Parking Area and Parking Lot** For the classification task, we decided to take each parking lot and use our model to identify is this parking lot filled with a vehicle or not. With this design approach, our system is easily adaptable for the different parking areas. We are labeling the legal parking areas like parking lots. And our program gives all of the parking lots to our trained model and the model mainly makes classification over these parking lots. We designed our parking area structure line by line. This gives us a chance to describe roads among the different lines of the parking area.

4 Development/Implementation Details

4.1 Client

While creating the mobile and desktop web applications, we used React with Typescript. Typescript is a strongly typed super-language of JavaScript which minimizes the type errors during development preemptively and provides auto-completion for a better developer experience. We used Chakra UI [25], a UI component library to reduce the time writing basic components from scratch.

For the communication between the client and server, we used GraphQL [26], a data query and manipulation language for APIs. GraphQL makes developing the API and requesting data from the client simpler and less time-consuming. To send GraphQL queries from client side and also enable caching of the queries we used a library called Apollo Client. React employs a model called hooks to extract component logic into reusable functions and while we use hooks throughout the client side, we also use it for GraphQL queries. This allows us to use the same queries in different components easily. To create hooks from GraphQL queries, we use a library called GraphQL Code Generator which generates hooks from the GraphQL schema with types.

Three of team members worked on creating the pages as close to the designs in the Analysis Report using Chakra UI. Initially, only the visual elements were added to each page. After the visuals are completed, using React and Apollo, pages were hydrated with data from the server by adding the required queries for each page.

For the map, we used Google Maps API for JavaScript. By getting the user's current location from the Geolocation API of the browser, we can show the current location of the user on the map and also the nearby parking lots. For the directions to a parking lot, we use the Google Maps Directions API and send a request with the user's current location and the location of the parking lot. The steps to the parking lot are given as an array which then displayed on the screen.

For the version control, we used Git and GitHub [22] for the remote repository hosting. While using Git, we employed a feature branch workflow for a better developer experience and encourage developer independence.

4.2 Server

For the back-end, we used a PostgreSQL as the database. To initialize the database schema with migrations and dummy data for testing we used KnexJS [28], a JavaScript library for writing SQL queries. To provide a GraphQL API for the client, we used Hasura [29], which is a GraphQL server which creates a GraphQL API by connecting to the database.

In Hasura, we configured the one-to-many and many-to-many relationships which are already defined by the database schema for GraphQL queries to fetch data from each table. Hasura provides filters, sorting and limiting for each query and for each table which we then use in the client application.

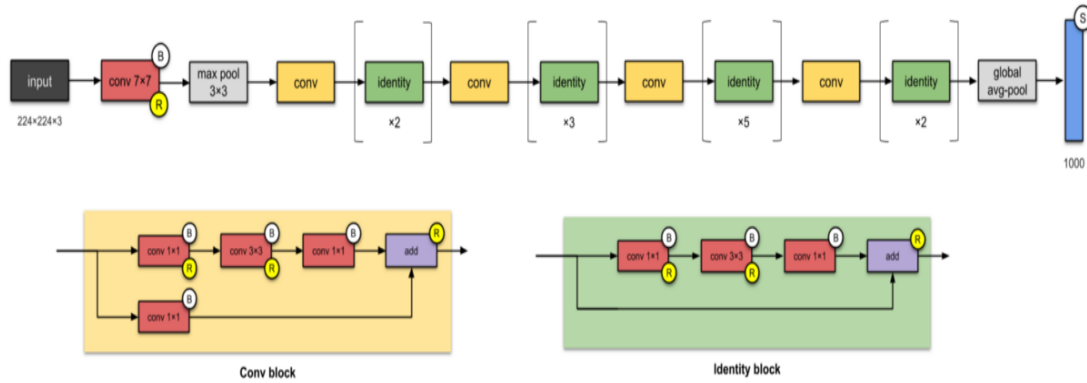
To add custom business logic to queries, we used Hasura's Actions feature. Actions allowed us to create custom queries and responses. For Actions to run, we also deployed a NodeJS server which contains the functions for the custom queries.

Our team works with three different operating systems, Windows, Linux and MacOS. To ensure consistent development between developers on different machines and operating systems, we used Docker [30] to run the back-end server. With Docker, we can run the required processes inside an isolated container. For the front-end, OS was not an issue during development. NodeJS, which is required for React development environment can be run on any OS.

4.3 Computer Vision

4.3.1 CNN Model Decision and Model Details

ResNet is a powerful CNN model that is used very frequently in many computer vision tasks. and it uses skip connection to add the output from early later to latter layers. This mitigates the solves losing gradient problem. We have selected Resnet50 because it is very frequently used in Transfer Learning. Resnet50 has 26M parameters and it consists of convolution and classification parts. Thus, training 26 M parameters for thousands of images requires a lot of computation power and time. For solving this issue we have used Transfer Learning over the Resnet50.



4.3.2 Transfer Learning Implementation

As mentioned Transfer Learning is the approach of using the pre-trained parameters for feature extraction and rearranging fully connected layers for the object classification. This is very useful when facing large datasets. We have added new classifier layers and only trained our model parameters for these modified layers. Because we made requires_grad = False for all the parameters. But our classifier layers were added after that and its parameters are not frozen. As we can see from our transfer learning code, we have frozen the parameters of the layers which are not modified, and during the training with no grad command, we have ignored gradient calculation in these parameters.

```
train_on_gpu = cuda.is_available()
trainiter = iter(dataloaders['train'])
features, labels = next(trainiter)
features.shape, labels.shape

model = models.resnet50(pretrained=True)

for param in model.parameters():
    param.requires_grad = False

n_inputs = model.fc.in_features
model.fc = nn.Sequential(
    nn.Linear(n_inputs, 256),
    nn.ReLU(),
    nn.Dropout(0.2),
    nn.Linear(256, 2),
    nn.LogSoftmax(dim=1))

if train_on_gpu:
    model = model.to('cuda')

model.epochs += 1

# Don't need to keep track of gradients
with torch.no_grad():
    # Set to evaluation mode
    model.eval()

# Validation loop
for data, target in valid_loader:
    # Tensors to gpu
    if train_on_gpu:
        data, target = data.cuda(), target.cuda()

    # Forward pass
    output = model(data)

    # Validation loss
    loss = criterion(output, target)
    # Multiply average loss times the number of examples in batch
    valid_loss += loss.item() * data.size(0)
```

4.3.3 Training Dataset and Training

We have used the PKLot database for acquiring our training, validation, and test sets. This database contains 12,417 images (1280X720) captured from two different parking lots (parking1 and parking2) on sunny, cloudy, and rainy days[3] . In addition, as we mentioned before our approach was segmenting the parking lots and training our model for identifying the vehicle in the parking lot. Thus, we have divided these 12,417 images over 80 thousand sample parking lot and used 48,244 images for the training set and necessary number of validation, test images. At the end of the training, our Resnet50 model gave 98.81% accuracy. We have controlled the chance in validation accuracy to prevent overfitting. We have used GPU with 1520 Cuda cores and training time took 1 hour 32 minutes. At the end of epoch 8, we faced a decrease in the validation accuracy and for the sake of preventing overfitting, we loaded our trained model at the end of epoch 8 and saved it for further usage.



4.3.4 Display Manager and Loading Trained Model for Classification

We processed the video stream as frames and for the frame processing, we are calculation changes in the frame or the change in the time. In the design process, we decided to update our system every 30 seconds but for the testing and the implementation parts. We faced with lack of video stream due to covid-19. And our sample videos were very short. Thus, we have made our tests for every 10 seconds due to short video streams. When new parking lot control time has come our system finds the lots using park architecture object which is predetermined for each line of parking space. After our system crops each parking area and passes these crops over our trained network. According to the results of these crops, our system sends free or occupied labels to our server.

5 Testing Details

Throughout out the implementation, we often need to test our application and its components. We used Agile methodology[2] in our project and used Scrum[1] as a application development framework. Hence we developed our application in parts. We divide our team into small pieces of 2-3 members and give small tasks to each team. We called that small works, Sprint. At the end of each sprint we met as a full team and discuss our implementation details. As a team we decided to not use any automated test to tool for verification and validation rather then IDEs. Thus, we do not have any test module to show but we did multiple test approaches. Due to Covid19 pandemic we could not release our application to the public and could not get test results from the beta test users. Therefore,

we decided to remove beta test. We did not use any testing tool to test our application but we tested Parkhound manually after each sprint. By verifying each task at the end of each sprint, we avoid testing a final and complex version of the application over and over. Therefore, we fixed smaller bugs immediately after each sprint before moving to next task. We did our verification in 3 parts. Following activities and methodologies were performed for the verification of Parkhound:

- **Code functionality review:** We tested and reviewed the code and the functionality after each step of implementation completed. We discussed the defects for each committed code part and try to solve them before starting a new sprint. However, we are a small development team and some bugs can be missed.
- **Automated Code review:** We used many IDEs (Integrated Development Environment) such as IntelliJ[6], Visual Studio[7], PyCharm[8]. These IDEs have a built-in system that checks code automatically and analyses the code before run. Also, they detect the code for the run-time errors and reports them to the developer. Reported bugs can easily be solved by the developer.
- **Software Walk-through:** Team performed software walk-throughs when a task assigned to someone else or when a team member takes off on a task. Previous responsible of the task explains the details of his implementation to the new member before new responsible of the task starts to work on task. By using that method, we aim to achieve dynamically coding without losing time and information of implementation.

We tested Parkhound by validation of the application. As a difference from the verification part, we discuss the requirements and the functionalities of the components after each sprint. Also, before the demo and publishing the first version we will do a release and customer tests with the team and customer to make sure everything implemented as customer's desire. Validation is examined under 3 parts:

- **Development Testing:** This is the first part of the validation part. In this part only development team is responsible for testing. Team tests individual components of the Parkhound after implementation of each class, interface and function. After each test, problems and bugs are discussed between team members and these issues are reported to solve. Since Parkhound has 3 main components, we tested each of the component separately and do Integration test between these components.
 - **Computer Vision Model Testing:** Since Parkhound based on Computer Vision model and detecting occupied parking spaces, this part is Parkhound's main and the most important testing part. To train and test model after each epoch, we split our dataset into 3 pieces: training, validation and test. Parkhound's implemented neural network model of the project simultaneously tests the trained data and gives an output in terms of loss and accuracy. Although model tests the data and gives information, we had to check trained model with different test sets. Before the release, team will collect new test data from different parking lots and give them to the trained model to test. Also, we improved our model and its parameters, according to the test results. Finally, developer team is satisfied with the accuracy results for occupied/empty parking spaces, team decided to move to the release testing for this part.
 - **UI Testing:** Parkhound runs both on desktop and mobile. Therefore, there may be incompatibility between two versions. Team tested over and over for each UI component after implementation of each UI functionality and until every seen bug is solved. Team decided that UI is bug-free, team moved to the release testing part of the UI.

- Backend Testing: Parkhound has a database system to store user information and system data. Therefore this component tested after each table created and backend functionality. We used dummy data to fill tables for testing.
- Integration Testing: In this part, we tried to integrate components and test them together as a component rather than separated testing. Aim of this testing was, to see how does components work if they are integrated. We tried to solve bugs relations between components rather than component specific bugs.
- **Release Testing:** This part of the testing has done after the development stage and just before the first release. Main purposes of these tests are, interacting with the customer and users rather than developers of the application and making application ready for the public release for all users. We did scenario testing to make sure implemented parts of the application satisfies user stories that we mentioned in previous reports [5]. As a final test before release, we did an optimization test to make sure we can run Parkhound on min-specified phones and desktops.
 - Scenario Testing: We manually tested the system with test scenarios specifically designed according to the user stories to check if the main use cases for the user are performed correctly since we developed with Agile methodology. Also, we tested requirements with specific cases per each requirement.
 - Optimization Testing: We tested Parkhound on multiple OS's and systems with different specifications to make sure Parkhound works stable. We integrated all parts and tested them in terms of performance. Team has faced some optimization and performance problems during the test. Most of the problems solved but Parkhound still has some performance issue because of the server's performance on trained model. Also, we are using ip cameras to update parking lot dynamically which causes a decrease in performance. Since we could not use our cameras and wanted camera angles, we moved that problems to after release part in order to fix them.
 - Alpha Testing: After all tests are done, we planned to publish Parkhound to limited user. Team will publish the application v0.1 to 20 users and will test the capacity of the servers. All alpha test users will send feedbacks about the application and after one week server will enter maintenance mode until demo. During this time, team will fix the bugs of the Parkhound and check every feedback from the alpha test users.
- **Customer Testing:** Customer will download and install the Parkhound demo to the desired environment. Then customer will test the demo. Customer will check the fully integrated version of the Parkhound to make sure whether application is ready for public publish or not.
 - Open Beta Testing: As final test, we decided to make an open beta test. Firstly, we will hard reset our database. Then, we will publish Parkhound publicly to all users of the current parking lot. Also, they will give feedback about system and new improvements. Open beta will take approximately 3 months if enough amount of feedback collected. During beta test all members of the team are responsible for bug fixing and communicating with customer.

6 Maintenance Plan and Details

Maintenance is a must for Parkhound because application is heavily server and model dependent. We are dynamically updating the server and database according the data that comes from the

model. Thus, we have to provide maintenance often to keep Parkhound stable. Model may face with unexpected frames such as bad quality videos, camera issues, bad weather and light conditions etc. Therefore, we will provide a maintenance plan for each company/customer that use Parkhound, after the publish. Main parts of the maintenance will be server and vision model. As a team, we decided to divide maintenance plan into 3 parts:

- **Computer Vision Model Maintenance:** Vision model is Parkhound's main part. It uses a neural network to train model and live captured parking lot data to give results about spaces' status. Therefore, if model face with an unexpected issue, we have to deal with it manually. We will monitor the status of the model and save the errors to model's error log. Every week one member of the team will check issues and report a fix plan to the others. Then, team will close the application for maintenance and start to applying fix plan to the model. Planned maintenance duration is 2 hours and it will be announced before. In the earlier times of the application, there may be several consequent maintenance depends on model's performance.

Since, computer vision and neural networks are popular topics on industry, every day new improvements and new technological issues come out. So, we have another issue about model which is updating the model to the recent developments and technologies. Our team will follow conferences such as CVPR and ICCV which are most important conferences on computer vision field. Regarding to the developments on our model and network, team will try to adapt Parkhound to improvements and new technologies. On the other hand, programming languages are updating too. Therefore, there will be yearly update patches. We are planning to do a yearly maintenance to update model to the new one.

- **Server Maintenance:** Parkhound stores the status of the parking lots, maps of the parking lots, trained model, classifier for parking spaces and currently processing frame of the video in a server. We divide server maintenance into two parts, first one is database maintenance and second one is server machine maintenance.
 - Database Maintenance: For the first release we will hard reset our database to make a fresh start. Users and companies will create and submit their data to the database after the release. There may be some wrong data in database tables. Team will monitor the database from the admin panel and will fix found problem immediately. However, some fixes on database may change other related data and damage stability of the application. As a team, we decided to do weekly database maintenance which will take approximately 1 hour. This maintenance will fix found bugs, remove dummy data and update collapsed data in database. Also, if customer needs an emergency maintenance on database's data, team may close server for emergency maintenance.
 - Server Machine Maintenance: Server maintenance will be done to fix problems that occurs on server machine. For the first release, we made a configuration to our server. Every month, we will close the server and do a maintenance on server's configurations and bugs of the system. This regular maintenance will not hard reset the database. Unfortunately, server machine may closed or crashed due to high population of current users and power issues. In that case, we will provide emergency maintenance.
- **Application Maintenance:** Parkhound works on multiple platforms. There may be some bugs regarding to UI compatibility. Bugs and feedbacks about UI and application logic will collected from the the customers and users. Team will review that bugs and fix request case by case and announce a maintenance every 3 months.

- **Camera Systems Maintenance:** Parkhound designed to use its own cameras to acquire data to train model and update status of the parking lot according live camera data. However, due to Covid19 pandemic we could not setup our camera systems and connect them to system. Before pandemic conditions, we decided to provide regular maintenance to camera systems every 3 months. For the first release, we can not provide any own camera systems and maintenance to the parking lots but after the pandemic we will setup our camera configuration with the assistance of the customer. We will hire a technical team for that purpose.

7 Other Project Elements

7.1 Consideration of Various Factors in Engineering Design

In the implementation process of the Parkhound, there were some factors that needed to be considered and in this part of the report, these factors and their effects will be discussed.

7.1.1 Public Health

There are no physical harm that could be directly caused by our application. But one possible scenario that could do some harm indirectly could be misleading a potential user by directing them to a parking lot that is full and causing them to crash into some car. In this case, user can experience some of the various car crash effects. To prevent this issue to some extend, we tried to train the model so that it can detect the emptiness of parking lots with highest possible accuracy.

7.1.2 Public Welfare

This factor has no effect on our solution.

7.1.3 Public Safety

Since our application is for seeing available parking slots and navigating towards them there is no public safety issue related to it. Therefore this factor also has no effect on our solution.

7.1.4 Public Information Security

Many people can use a company parking lot when they come to do any kind of business with the company and maybe these people may choose to not reveal their identity publicly. In our application, only company security officers can reach the identity of people in the parking lot as a list of names and regular users cannot reach this information. It is due to secure the user information. We believe public information security is an important factor on design and we also encrypted user information in our database to reduce the negative effect of this factor.

7.1.5 Global Factors

In the discussion of global factors in our application, we consider various rules and regulations based on the context of our application as well as the platforms we plan to publish our application. There are some about car parks that we must obey and also we need to accept the terms and conditions of Google Play Store[10] and App Store[11]. We limit ourselves with the conditions that mentioned. Also, there is a language barricade between countries and since we want to publish and application that reaches global standards, there is multi language support in Parkhound.

7.1.6 Social Factors

Social factors can include wealth, religion, buying habits, education level, family size and structure and population density[12]. When a user signs up to our application, we only collect name and email addresses in our database. We don't use any personal information publicly in order to avoid having any social issues with our users. Cameras that will be set up may catch car brands and appearance of those cars and that could give clues about social status of users. But we do not keep any of these recordings. We only used some prerecorded video frames to train a deep learning model. Lastly, we need to consider the social construction of technology behind parking lot designs and theories.

7.1.7 Cultural Factors

Parkhound tries to achieve to generate a universal car parking system and generally people use similar sized car parks and similar sized cars all around the globe. However, there can be companies that may mostly use vehicles that are not very common in other places. In this case, our deep learning model should also be trained for recognising those vehicles in order to reduce the cultural differences in car parking.

7.1.8 Environmental Factors

Our computer vision based model can be affected by some of the weather conditions such as fog, rain or snow. Also, depending on the hour, dark shadows on parking lots can also mislead our model to think that there is a car parked there even though that specific parking space is empty. Therefore, we will focus on training our model for those extreme conditions in order to get better accuracy in all possible weather conditions.

7.1.9 Economical Factors

Our system need cameras to process images and we are planning to use low cost IP cameras to record parking areas. And also we need to create a server and make connections with each installed cameras and company computers. Cameras and server specifications are costly because of the quantity and quality of images. To reduce cost, we will merge common parts of captured images to reduce image size and we will process each parking space once by giving each space an ID.

In a table below, we indicate the effect for each mentioned factors most important issue and gave them levels to show their significance. Levels are between 0 (None) and 10 (Maximum).

	Level of Effect	Effect
Public Health	2	Model with low accuracy can mislead user into a car crash.
Public Welfare	0	This factor has no effect on our solution.
Public Safety	0	This factor has no effect on our solution.
Public Information Security	6	Public information can be leaked and jeopardize people.
Global Factors	3	Country based adaptations of features like language. Not fitting into global rules and regulations.
Social Factors	3	Fitting socially excepted parking lot designs.
Cultural Factors	5	Possibility of different kind of vehicles and parking lots.
Environmental Factors	6	Weather conditions can mislead our system.
Economical Factors	4	Less available memory and cost of high-quality video recordings.

7.2 Ethics and Professional Responsibilities

In this section we inspect the ethics and professional responsibilities that we have as a team of engineers. We divide these into the following subsections and some of them were studied in our analysis report [5].

7.2.1 Ethics

Ethical concerns are including privacy and data sharing. Parkhound tower over camera records and those data records contains information about personal cars and special parking areas. As mentioned in section 7.1, all of those data are protected by the security personal of the company that owns the car park which the data is recorded in. Our project does not interfere with the methods used by security personal to protect sensitive information in data records. In addition to that, we are not sharing any collected data with third-party apps.

7.2.2 Environmental Responsibilities

In the environmental context, Parkhound is really environment friendly because it only uses electricity as a source and if it comes from a renewable energy source, then Parkhound has no environmental cost at all. Besides electricity, Parkhound uses cameras as mentioned before and we specifically choose cameras that can be recycled so it also has no effect on Parkhound's environmental cost.

7.2.3 Economical Responsibilities

As explained in the former parts of this report, Parkhound is an application that processes parking lot images to automatically determine empty and full spaces and alert user. In order to achieve this task, it requires a camera system and a server setup and they have some cost. But other than the installation costs of camera and server setups and the cost of these material, Parkhound does not create any other fee requirements.

7.2.4 Social Responsibilities

Parkhound aims to be used by all kind parking lot owners from all kind of societal and economical backgrounds. One of the Parkhound's main goals is to give service to every type of parking lots. It also fits into socially constructed parking lot designs and physical requirements. Lastly, there is not much of a social platform inside Parkhound therefore users cannot have much on an interaction and this results to not having social conflicts or any kinds of problems that involves people. Hence, it can be stated about Parkhound that it is very society friendly application.

7.3 Judgements and Impacts to Various Contexts

In this section, we examine the impacts of various context according to our responsibilities and the judgements we had to make in order to achieve our goals with Parkhound. Levels of impacts are low, medium and high.

7.4 Teamwork Details

7.4.1 Contributing and Functioning Effectively on the Team

In the Parkhound team, every member contributed the overall project implementation process from the very beginning to the end. We were all aware that in order to create a successful and solid

	Level of Impact	Impact
Impact in Global Context	High	Parkhound has a high impact in Global Context because it is a unique system that makes finding parking lots much easier.
Impact in Economic Context	Medium	Cost of Parkhound is really low compared to its effectiveness because it only new one time of installment and it saves a lot of time in return which can be spend by being productive in other works than searching for parking spaces.
Impact in Environmental Context	Low	Parkhound does not have much impact in the context of environment.
Impact in Societal Context	Low	Parkhound is an applications that meets the requests and requirements of every part of the society but is does not have an impact on the society on a considerable level.

project, every member needed to function effectively on the team for both meeting every objective on the way and having motivation. That being said, members of the project Parkhound and their main contributions can be seen as following:

Berkin İnan: He took part in the front-end work, contributed mainly to drawing the sketch of the parking lot and showing occupancy on the sketch. He also contributed on the admin's panel and helped with generating database queries. He connected Google Maps API[13] to our project and implemented an interface that gives directions and shows available parking lots. Lastly, he was also our project manager.

Görkem Yılmaz: He mainly worked in front-end development, generated screens other than the main page. He made database connections to those screens by generating GraphQL[26] queries from the Hasura[29] and saving those queries into our project folders. Helped Berkin with the design and decision making processes of extracting the necessary data from the computer vision department and processing it to generate parking lot sketches that are fully functional.

Ata Coşkun: He mainly took part in back-end development. Implemented our server with the usage of Docker[30] and Hasura[29] and connected their interfaces into our project. He helped with our configurations because we worked on Linux, MacOS and Windows systems. He also helped with the design of the screens. He helped with creating tables in our dataset. He was also responsible for software testing and reporting bugs.

Ege Turan: From the beginning of the project he took part in creating the model and training the model. He spent his time in first semester to understand how neural networks work. He wrote multiple Python scripts to train and test the model. Also, he lead computer vision model implementation and direct other teams members to how will model used in Parkhound. He was also worked on server-side of the application to get and post requests to the model. He helped to the front-end part and implemented transactions and interactions between scenes after the training of the vision model finished.

Arda Türkoğlu: He mainly took part in computer vision part and functionalities part. He worked with Ege on the vision part and discussed possible solution and he created a road map for the computer vision tasks. With Ege he did a literature survey about computer vision and classification tasks. Afterwards, he moved to the dataset acquiring part and processing dataset to make it compatible for model training. He and Ege implemented model from scratch and debug it multiple times. After the training he created captured parking lot videos and labeled each parking space to test it. He was responsible for the stability and testing of the model and filling parking space tables for the backend after training for the model is finished.

Note: Due to pandemic[18] and conversion of meetings to remote system, we had to change our project plans. Thus, we decide to work as a whole team on every functionality. All team members helped to solve problems of team members, bug solving and writing reports. We specified different work focuses for everyone but every team member also has labor in others' work for the most of the time.

7.4.2 Helping Creating a Collaborative and Inclusive Environment

As a team, we believed that working in a collaborative and inclusive environment was essential because being a team means helping each other with our challenges by collaborating and having a comprehensive work organization feeds and supports the team.

From the beginning of this project, we had already agreed on having meetings regularly and discussing each others problems and ideas. We were specifying meeting places and meeting hours but due to Covid-19, we all closed at our homes and switched to social distance lifestyle. During the pandemic, we used several platforms to have meetings, making presentations to each other and sharing files, code pieces and ideas. These platforms and why we used them can be listed as follows:

- **Zoom:** Zoom[16] was our primary way of making meetings especially when team members needed to share screen to present their code or any other related works. It was also our way of joining classes and other meetings therefore as a team, we used it almost everyday. Only problem was when there were more than two team member in the meeting, Zoom allowed only 30 minutes of meeting time. Essentially, we find that problem a solution by using Discord.
- **Discord:** When we made meeting that have more than two members, we used Discord[17]. Screen share feature was not as quality as Zoom's but we could meet and share everything that needed to be share without any problem. We can say that we all created a collaborative environment in Discord and it supported our process of project implementation.
- **Overleaf:** Throughout the project, we wrote six different reports including this one and we used Overleaf[14] to write our reports in LaTeX.
- **GitHub:** GitHub[22] was also a collaborative and inclusive environment that we used because each of us could commit our code, see and list the things that needs to be done, in the process of getting done and already done. We used different branches to collaborate to project. We used two repository and one of them was for computer vision related work and the other one was for the front-end and back-end development of the project. Mostly Arda and Ege committed to computer vision repository and Berkin, Görkem and Ata committed to other repository.

7.4.3 Taking Lead Role and Sharing Leadership on the Team

As a team, We wanted everyone to share and be part of the leading process therefore we assigned different leaders to different parts of the project implementation. Hence, throughout the development

process of the Parkhound, all of our teammates led different subjects. We all learned and tested the importance of leadership and its methods in a big project such as Parkhound. In the table below, the leading roles and their definitions for each team members can be seen.

	Leadership Position	Definiiton
Berkin İnan	Parking lot sketches, directions and APIs.	He lead the process of generating parking lot sketches from the data we got from predictions of our deep learning model, lead the design and implementations of directions and usage of Google Maps API.
Görkem Yılmaz	Design and implementation of screens	He lead the process of designing and implementing general screens of Parkhound, made decisions regarding to scalability and functionality of buttons and links on screens.
Arda Türkoğlu	Labels and identifications of parking lots and spaces.	He lead the process of identifying individual cars in parking lots, labeling them and deciding the order of starting and ending positions. Also, the process of merging intersecting video frames.
Ege Turan	Computer vision model	He lead the process of training our deep learning model and made decisions about the hyperparameters and other restrictions to use to increase the accuracy of the model.
Ata Coşkun	Databases and queries	He lead the process of generating tables and making their connections. He made decisions about creating queries based on efficiency regarding joining tables.

7.4.4 Meeting Objectives

In this section of the reports, we state and discuss the objectives that we initially set in the project-plan and whether they are met and if so, at which level. We refer to our analysis report[5] for our initial objectives.

7.4.4.1 Functional Objectives In this part we list the functional objectives that we have met. They will be listed based on Admin, User and System.

Admin Objectives:

- Admin is now able to see who entered the parking lot.
- Admin is now able to close some part of the parking lot or whole parking lot temporarily.

User Objectives:

- Users can now able to register to Parkhound by filling their information.
- Users can sign in to the application by filling their information on the login page.
- Users can now get directions to parking lots.
- Users can now see available parking lots.
- Users can now see the occupancy status of individual parking spaces.
- Users can now see their favourite parking lots and save parking lots to their favourites list.
- Users can now see the address of parking lots.
- Users can now see the locations of the parking lots on the map.

System Objectives:

- System now detects the occupancy status of parking spaces and shows in on the application.
- System now detects the occupancy status of parking spaces in various weather conditions.

7.4.4.2 Non-functional Objectives In this part we list the non-functional objectives that we have met. They will be listed based on reliability, security, performance, extendibility and portability.

Reliability Objectives:

- Application is now working without any interruption or crashes.
- Application is now provies %98 of accuracy on determining the parking status of the vehicles.
- Application is now provides real-time data of the available parking spots.

Security Objectives:

- Application is now secures the user information from possible threats by using encryption.

Performance Objectives:

- Application is now has a very low loading time.
- Application is now has a fast image recognition and server connection so that users can quickly and easily look for the information they need to access.

Extendibility Objectives:

- Now the datasets that we use are extendable because we use an automated process.
- Application is now available on multiple platforms.

Portability Objectives:

- Parkhound can now run on Android devices, iOS devices and computers with web browsers.

Due to Covid-19 pandemic[18], we could not meet every objective that we initially set. We were planning on installing a real camera and record parking lots with somewhere we agreed the terms with. Instead, we had to find old recordings of parking lots online. Some problems we had with those datasets were when cars park backwards, system has a difficulty of detecting and if both sides of camera has not have similar lighting, system also has difficulties.

7.5 New Knowledge Acquired and Applied

Before starting to implement Parkhound, we had not have the necessary knowledge and skills. Because of that, we acquired plenty of knowledge, enhanced our skills and applied new software development methods. Some of the fields and technologies that we worked on and improved ourselves can be listed as follows:

- Deep Learning
- Computer Vision
- React Development
- Project Management
- Application Design

Thanks to Bilkent's CS 464 - Introduction to Machine Learning and CS 484 - Introduction to Computer Vision courses which many of us took, we had a little information about deep learning and computer vision on fundamental basis. But we had no experience training and designing a big and complex model such as Parkhound's. Also many of us had almost zero experience on React Development and a very little experience on overall front-end development. Unfortunately, Bilkent University does not provide any course with a main focus on front-end development or just JavaScript development. Again, many of us took CS 413 - Software Engineering Project Management course that Bilkent University offers and learn the basics about the field of Project Management. For different departments of Parkhound we had a different team members as a leader and we applied the knowledge we get to balance cost, time and our budget. Of course the knowledge we got from the courses in school were not sufficient in order to develop an industry level project as Parkhound. Hence, we all trained and educated ourselves on these topics using various methods. Some of them are as follows:

- MOOC (Massive Open Online Courses)
- Literature Review
- Experimenting

We finished MOOC courses in order to gain more knowledge about Parkhound's implementation and literature review was also helpful especially in the field of deep learning and computer vision. There were many CVPR papers that taught us different computer vision architectures and methodologies for developing a successful one with least cost. We tried many different things before finalising the implementation of a part of Parkhound and it was possible by making as many experiments as possible. We mostly learned the subject we were working on from our mistakes. Results of these painful processes were getting lots of software development experience and a successful project that is ready to be published.

8 Conclusion and Future Work

After all, as a team of 5 developers, we think that we succeeded to create a solution and an application. It was a great journey that took almost one year. We learned too much about parking problems, possible solutions, new technical knowledge and applying new technology and solutions to our system. However, world faced with Covid-19 pandemic and things did not go as we expected. We were thinking about working together all the time but after the first part of the course all of the team members went another cities and we had to work remotely. None of us have any previous remote working experience. Thus, we have to create another plan on the mid time of the project. We decided to make new design, new solutions and work breakdown schedule. We add, update and remove some requirements, learned new technologies such as neural networks, machine learning, front end technologies, back end technologies and creating a remote working server. Also, we learned a lot about writing a technical report on our proposed solution and our road map. To sum up, we are happy with our progress under these circumstances. We know that we could do some things differently and better but we all gain experience about making a huge software project and we will use that knowledge on our career lives.

As a future work, all members of the team will work remotely and will be a part of different companies but we will improving the Parkhound because we have some unfinished functionalities. Firstly, we will get a constant server to run application 7/24. Secondly, we will add admins, traffic members and other types of users to the system. Lastly, we will try to automate our parking lot labelling system. Also, we are open to any feedback from all users and other stakeholders.

9 Glossary

- **Parking Space:** A parking space is a location that is designated for parking, either paved or unpaved. It can be in a parking lot. The space may be delineated by road surface markings. The automobile fits inside the space, either by parallel parking, perpendicular parking or angled parking.
- **Parking Lot:** A parking lot, also known as a car lot, is a cleared area that is intended for parking vehicles. It contains many parking spaces. Usually, the term refers to a dedicated area that has been provided with a durable or semi-durable surface.
- **Convolutional Neural Network:** In deep learning, a convolutional neural network (CNN, or ConvNet) is a class of deep neural networks, most commonly applied to analyzing visual imagery [20].
- **Resnet50:** ResNet-50 is a convolutional neural network that is 50 layers deep. You can load a pretrained version of the network trained on more than a million images from the ImageNet database [21].
- **Transfer Learning:** Transfer learning is a machine learning method where a model developed for a task is reused as the starting point for a model on a second task [19].

10 User Manual



Find Your Parking Space

Find parking areas nearby and save time



Find Space →

Figure 1: Start Screen

Our start screen welcomes the users to Parkhound app. Users cannot interact with this page until application starts.

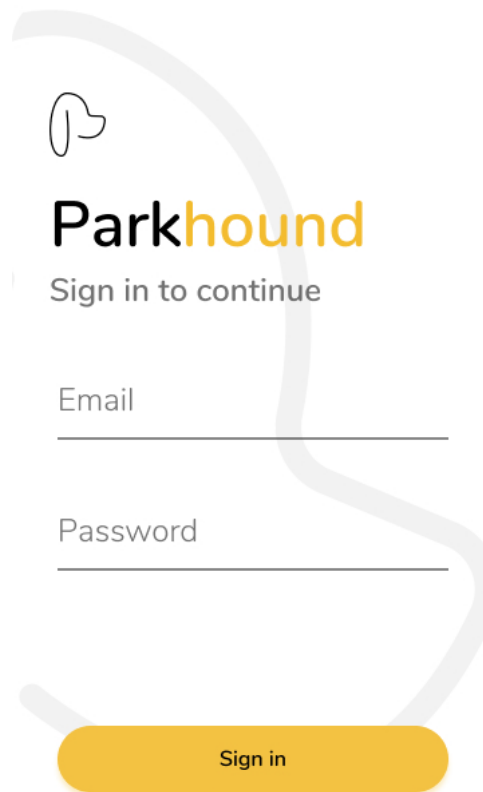
The image shows a login page for 'Parkhound'. At the top, there is a small icon of a dog's head. Below it, the word 'Parkhound' is written in a bold, sans-serif font, with 'Park' in black and 'hound' in orange. Underneath the logo, the text 'Sign in to continue' is displayed in a smaller, grey font. There are two input fields: one labeled 'Email' and another labeled 'Password', both with horizontal lines indicating where to enter text. At the bottom of the form, there is a yellow, rounded rectangular button with the text 'Sign in' in black. The entire form is set against a light grey background with a faint, abstract wavy line design.

Figure 2: Login Page

User enters his/her registered email to the Email section and password to the Password section then clicks on the Sign In button in order to sign in to the Parkhound.

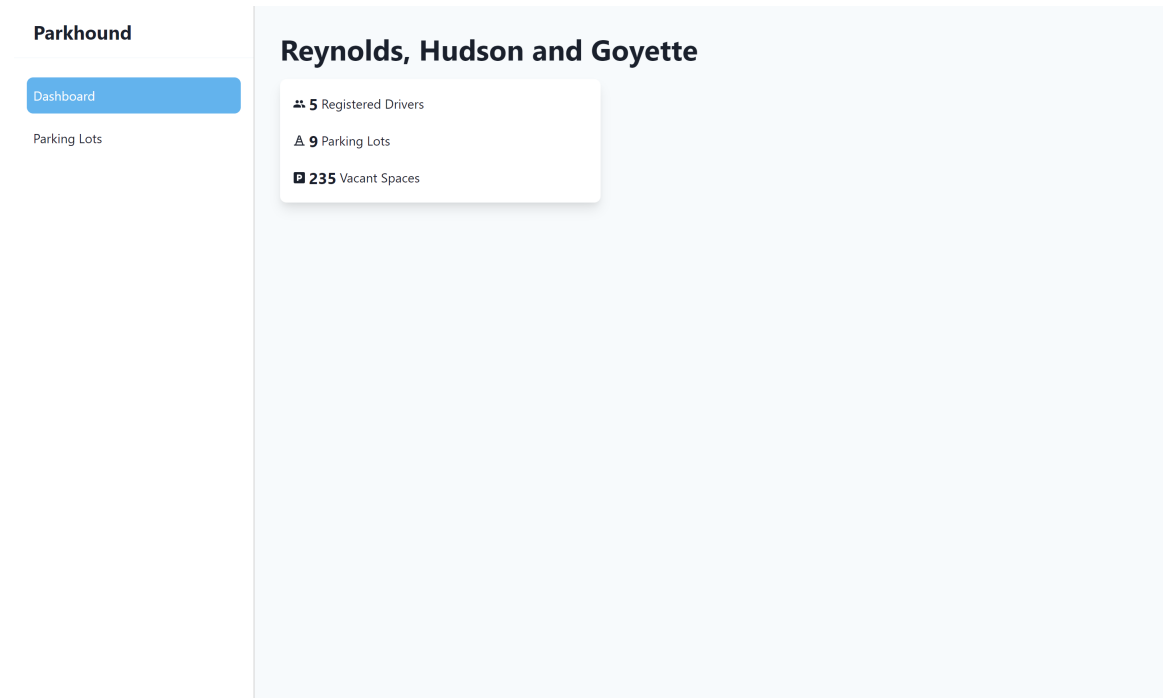


Figure 3: Admin Dashboard

In the Admin Dashboard, admin can interact with the Dashboard button to see the parking lots and their information in the listed form. Or admin can click on the Parking Lots button to do transaction with the parking lots.

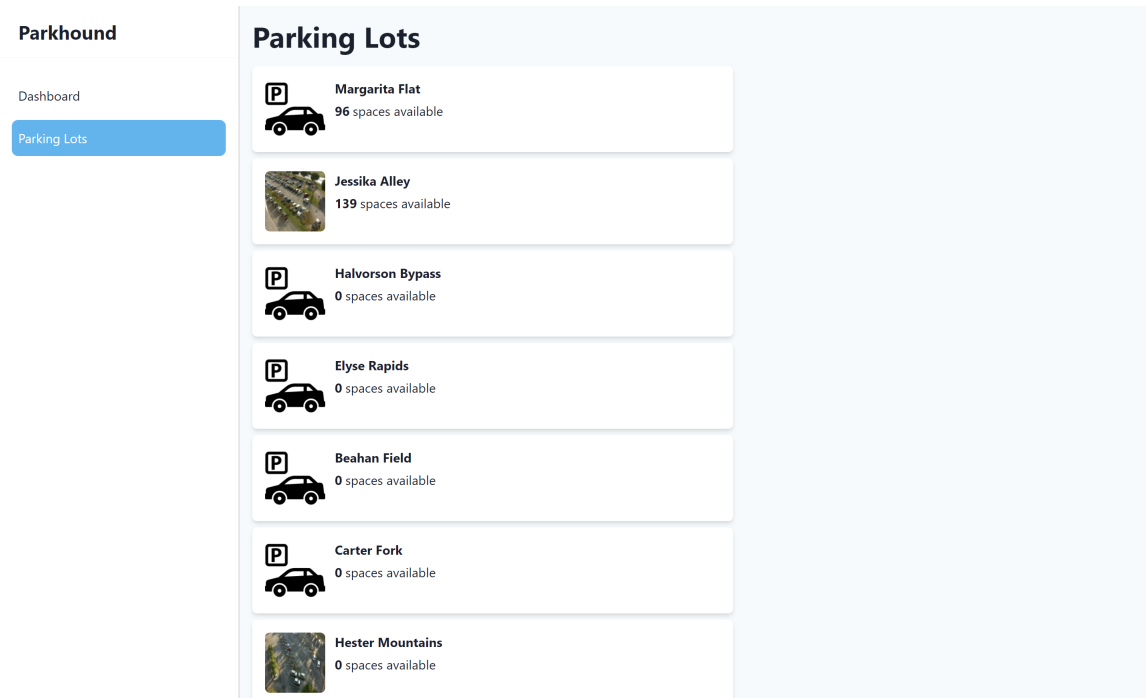


Figure 4: Admin Lot List

In the Admin Dashboard page, admin can see the parking lots in a listed form by clicking the Parking Lots button. Then, admin can select a specific parking lot from the listed lots by clicking on it.

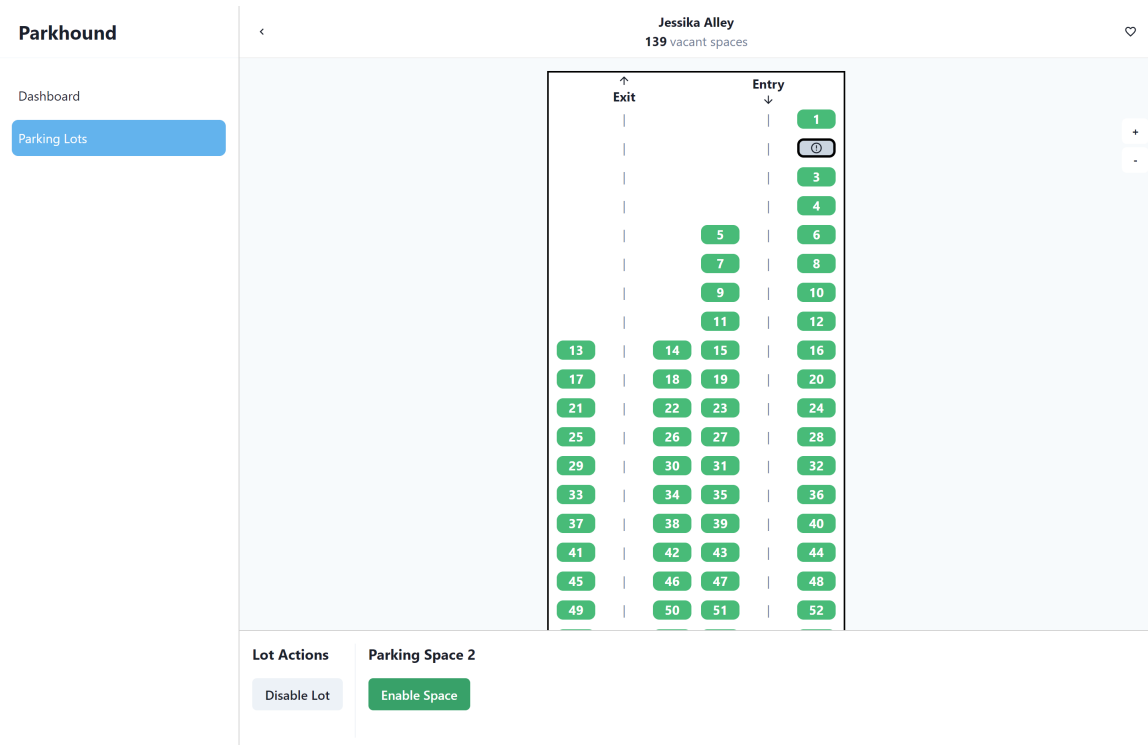


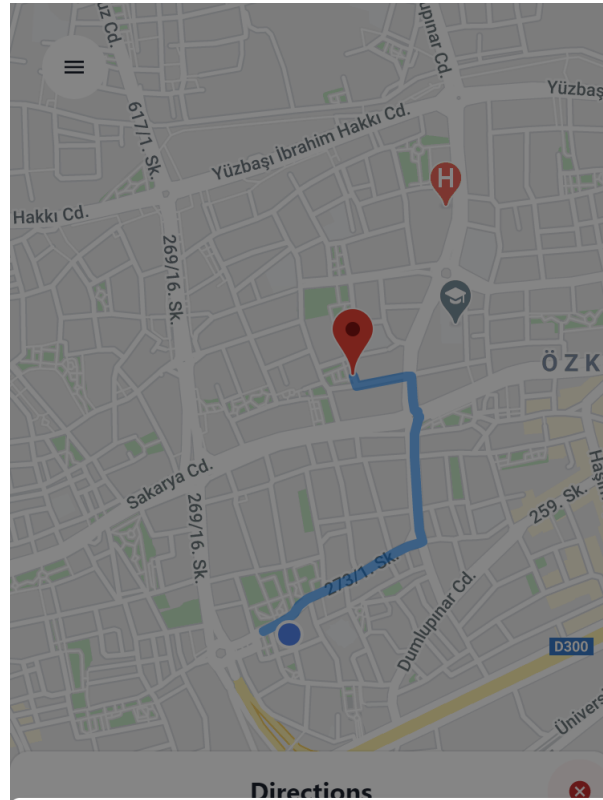
Figure 5: Admin Lot

In the Parking Lots section of the Admin Lot, admin first selects a parking lot from the listed parking lots then admin can enable or disable a specific parking space by clicking on the Enable Space/Disable Space button. Admin can also enable or disable the entire parking lot by clicking on the Enable Lot/Disable Lot buttons. Admin can see the status of the parking lot and its spaces from the mini map above the mentioned buttons.



Figure 6: Admin Lot Disabled

If admin disabled the parking lot, the mini map looks like this and it has a label saying "DISABLED" until admin enables the parking lot again.



Cancel Route

Do you want to cancel the current route?

Close

Cancel Route

Figure 7: Cancel Route

If user created a route and he/she wants to cancel it, Cancel Route button can be used. User clicks on this button and the route is cancelled. If user does not want to cancel the route, he/she can close this page by clicking the close button.

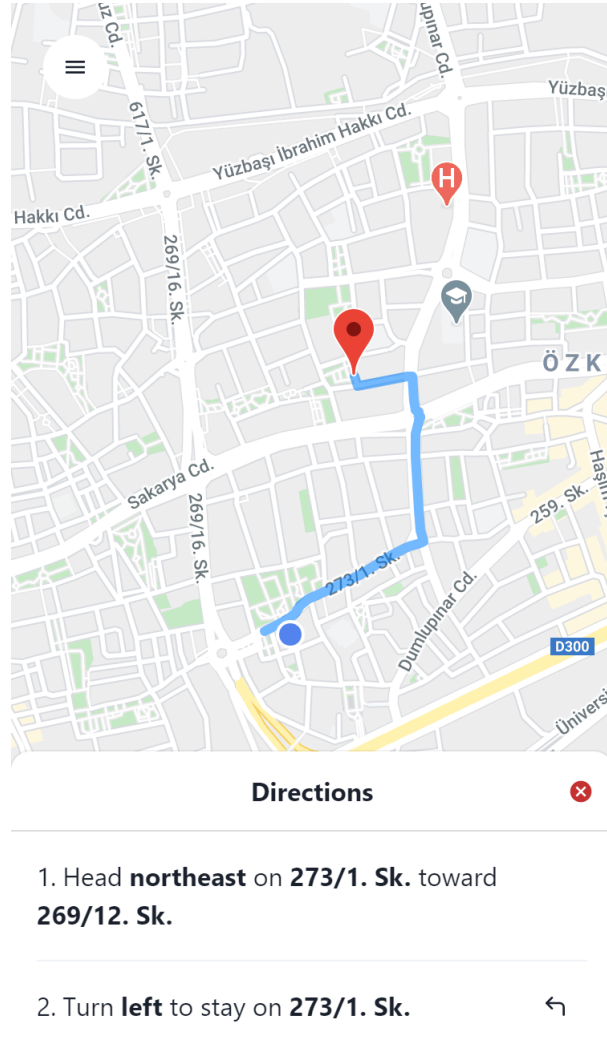


Figure 8: Directions

Once the user is selected a parking lot from the map interface, he/she can see the directions to that parking lot in the bottom of the screen as shown in the figure. User can swipe this bar up to see more of the directions. User can also click on the small red “x” symbol on the upper right part of the bottom bar to close the directions.

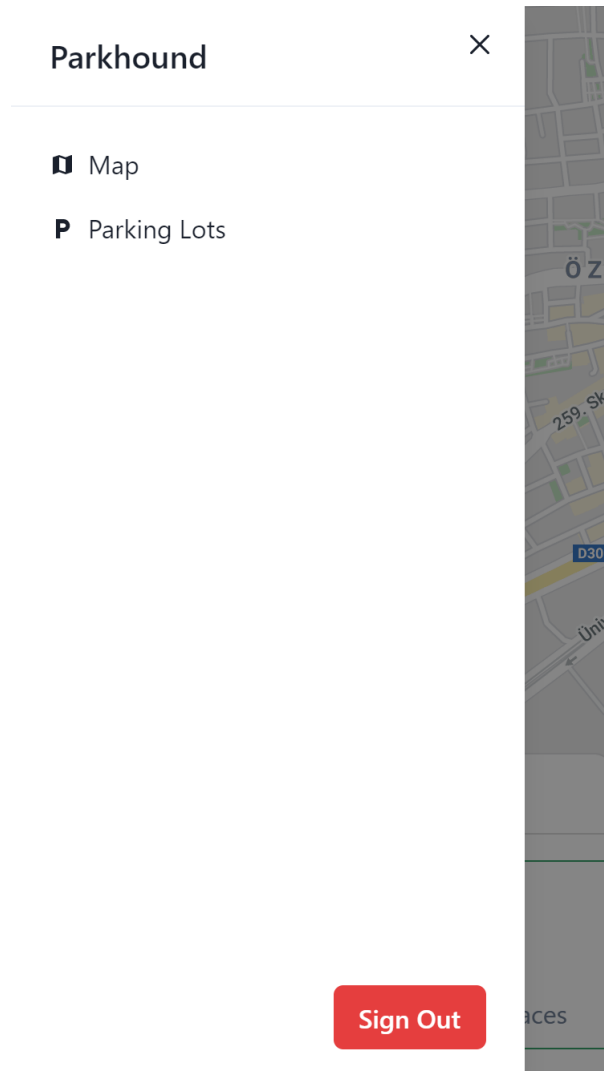


Figure 9: Drawer Menu

Drawer menu is Parkhound's sidebar navigation. Users can go to the other screens via this sidebar. It is always present in application. User can click map button to see current location and available lots. Also, user can click parking lots button to see every parking lots near him in a list. User can click sign out button at the right bottom corner of the sidebar to sign out from the application.

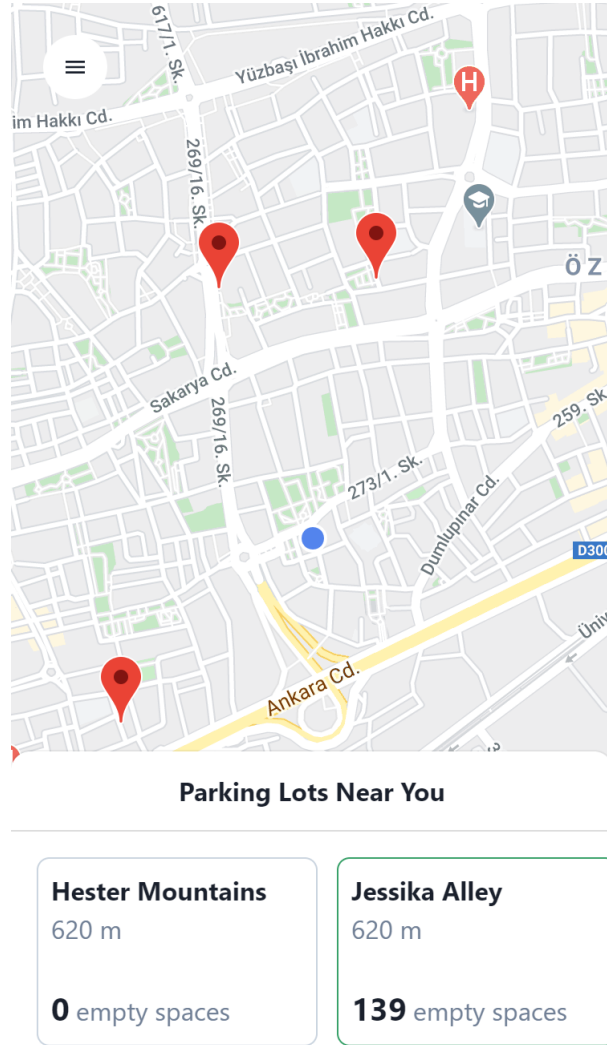


Figure 10: Home

In the home page user can see his/her location and parking lots near his/her. User can click on the listed parking lots below to create a route to them and see directions. User can click on the button on the upper left part of the screen to open up the drawer menu.

< Parking Lots




	Hester Mountains 0 spaces available 620 m
	Jessika Alley 139 spaces available 620 m
	Margarita Flat 96 spaces available 620 m

Figure 11: Lot List

User can see parking lots that is defined in system. Each parking lot shows distances and available spaces.



Figure 12: Lot View

User can see the parking lot and its spaces. User can see the parking spaces' status and their locations. Also, he/she can zoom and zoom out the screen to see all slots. When user select a parking space, they can click the navigate button to get route information. User can favorite parking lot by clicking heart button on the right.



Figure 13: Selected Lot

User can choose parking lot from the map. After user selection information about parking lots will be shown and user can click view button to view location and status of the parking lot. Also, user can open sidebar to move between screens.

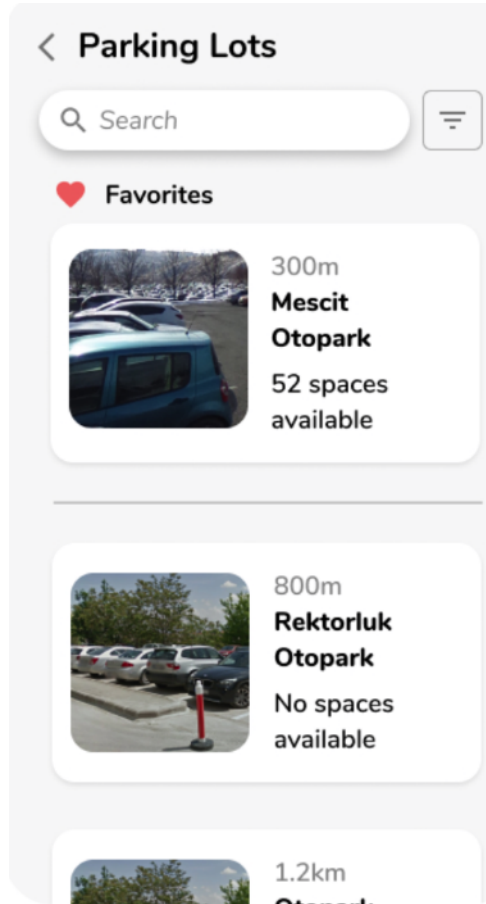


Figure 14: Favorites

User can choose his/her favorite parking space in the parking lots. This parking space will be in different color and light when user open his favorite parking spaces and parking lots. Also, user can sort parking spaces and lots by clicking upper right sort button. Therefore, user does not need to find his favorite parking lot each time.

11 References

- [1] "What is Scrum?" Scrum. Available: <https://www.scrum.org/resources/what-is-scrum>
- [2] "Agile Methodology" Available: https://en.wikipedia.org/wiki/Agile_software_development
- [3] "PKLot - A Robust Dataset for Parking Lot Classification" Paulo Almeida. Available: <http://www.inf.ufpr.br/lesoliveira/download/pklot-readme.pdf>
- [4] "A Dataset for Visual Occupancy Detection of Parking Lots" CNR Parking Dataset. Available: <http://cnrpark.it/>
- [5] "ParkHound Analysis Report" ParkHound. Available: <https://inanberkin.github.io/ParkHound/docs/Analysis>
- [6] "IntelliJ Idea" JetBrains. Available: <https://www.jetbrains.com/idea/>
- [7] "Visual Studio IDE" Microsoft. Available: <https://visualstudio.microsoft.com/>
- [8] "PyCharm IDE" JetBrains. Available: <https://www.jetbrains.com/pycharm/>
- [9] "IEEE International Standard for Software Engineering - Software Life Cycle Processes - Maintenance" IEEE. Available: <https://standards.ieee.org/standard/14764-2006.html>
<https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53>
- [10] "Google Play Terms of Service" Google. Available: https://play.google.com/intl/en-us_ua/about/play-terms/index.html
- [11] "Terms and Conditions" Apple. Available: <https://developer.apple.com/terms/>
- [12] "Social and Economic Factors". Available: <https://www.countyhealthrankings.org/explore-health-rankings>
- [13] "Google Maps API" Google. Available: <https://developer.here.com/products/maps>
- [14] "Overleaf, Online LaTeX Editor" Overleaf. Available: www.overleaf.com
- [15] "GitHub: Where the world builds software" GitHub. Available: github.com
- [16] "Zoom: Video Conferencing, Web Conferencing, Webinars" Zoom. Available: <https://zoom.us/>
- [17] "Discord | Your Place to Talk and Hang Out" Discord. Available: discord.com
- [18] "Coronavirus disease (COVID-19) pandemic" Covid-19. Available: <https://www.who.int/emergencies/diseases/novel-coronavirus-2019>
- [19] "Transfer Learning for deep learning" . Available: <https://machinelearningmastery.com/transfer-learning-f>
- [20] "Application of the residue number system to reduce hardware costs of the convolutional neural network implementation" Author links open overlay panel" M.V.Valueva. Available: <https://doi.org/10.1016/j.matcom.2020.04.0319>
- [21] "ResNet50" MathWorks. Available: <https://www.mathworks.com/help/deeplearning/ref/resnet50.html>
- [22] "Where the world builds software," GitHub. Available: <https://github.com/>

- [23] "React – A JavaScript library for building user interfaces," Available: <https://reactjs.org/>
- [24] "Typed JavaScript at Any Scale.," TypeScript. Available: <https://www.typescriptlang.org/>
- [25] "Chakra UI," Chakra UI: Simple, Modular and Accessible UI Components for your React Applications. Available: <https://chakra-ui.com/>
- [26] GraphQL, Query language for APIs. Available: <https://graphql.org/>
- [27] P.S.Q.L.G.D. Group, PostgreSQL. Available: <https://www.postgresql.org/>
- [28] Knex.js - A SQL Query Builder for Javascript Available: <https://knexjs.org/>
- [29] "Instant GraphQL APIs for your data: Join data across databases, GraphQL and; REST services to build powerful modern applications," Hasura. Available: <https://hasura.io/>
- [30] "Empowering App Development for Developers," Docker. Available: <https://www.docker.com/>
- [31] Cloud Application Platform Available: <https://www.heroku.com/>
- [32] "IEEE Standard for Software Project Management Plans" IEEE. Available: <https://ieeexplore.ieee.org/document/741937>

12 Appendix

Figure 15: Overview of the Application Modules

