SpotCheck

by
Ryan Bunker, Bradley Bruce, Will Watson

Submitted to
the Faculty of the School of Information Technology
in Partial Fulfillment of the Requirements for
the Degree of Bachelor of Science
in Information Technology

Ryan Bunker Brad Bruce Will Watson 4/27/20

Ryan Bunker, Bradley Bruce, Will Watson — Date

Abdou Fall — Date

University of Cincinnati
College of
Education, Criminal Justice, and Human Services
April 2020

TABLE OF CONTENTS

IMAGES AND FIGURES

ABSTRACT

In 2017, US motorists spent an average of 17 hours a year attempting to find parking after reaching their destination. In wasted time, fuel, and emissions this costed an estimated $345 per driver over the course of the year.[1] The amount of time spent looking for parking is expected to increase as more Americans begin driving. SpotCheck is an end-to-end solution to provide mobile users the ability to quickly and reliably find parking wherever they are. Using machine learning to process images from cameras arrays stationed around local parking lots, our application suite determines where open parking is, so you don't have to. Our applications allow users to quickly identify and get directions to the closest open parking spaces without the hassle. Parking lot information can be easily managed using our admin web application, allowing parking lot owners to easily make changes that are immediately available to SpotCheck users.

---

[1]https://www.usatoday.com/story/money/2017/07/12/parking-pain-causes-financial-and-personal-strain/467637001/

1.0 PROBLEM STATEMENT

1.1 Problem

In heavily populated areas with a limited amount of space, finding a parking spot can be very difficult. There are many cases where you are unsure of where an open parking spot may be near your destination (college classes, events, work, etc). People will often find themselves going out of their way, wasting time to find parking and when they do find one, it may not be ideal. According to a 2015 Los Angeles Times Article, parking is one of the largest issues facing colleges in America today. They state that a college of 30,000 students is unlikely to have even a fraction of that many parking spaces available around campus for student use (Rivera 2015).

According the Medium.com, Zona Azul Digital is one company located in the city of Sao Paulo that is working to fix the issue of parking in major cities. Using an extensive array of sensors around the city, they can navigate users to open parking saving time and money while also freeing up roads more effectively during periods of high traffic (Morelli 2017). While this is a great step, their implementation of using thousands of sensors around the city is expensive, intrusive to residents, and requires extensive maintenance. Overall, parking is an area that is really lacking an inexpensive and effective software implementation.

1.2 Solution

Spot Check will be a mobile application that provides drivers with the ability to locate nearby parking garages and easily see the number of open parking spots at that location. This application will also predict how many spots will either become open or closed over the next hour using historical data so that drivers can determine if there will be an open parking spot by the time they arrive at the garage. Drivers can also use the app to open their favorite navigation app and receive directions to the specified garage. Spot Check has a target audience of drivers trying to locate available parking in high density areas such as city centers or large venues. The application will be available on iOS and Android and will determine available parking by using computer vision to count the number of empty spots in a garage.

Spot Check will implement a series of small computers equipped with one or more cameras that will pointed at parking garages. Using predefined images of available and unavailable parking spots, a neural network algorithm with be able to determine if a spot is available and this data will be available for the user to view from the app on their phone. This system will be more cost effective and more efficient than other parking systems currently in use that use a single proximity sensor per parking space. Our camera systems will be able to view and update availability of an estimated 20-30 parking spaces per camera which is vastly reduce costs and maintenance.

1.3 User Profile

1.3.1 Potential Users

- Mobile users proficient in Android and iOS devices
- Mobile users not proficient in Android and iOS devices
- Admin user managing the application for their associated lot(s)

1.3.2 Software and Interface Experience

This project will be targeted at all users regardless of proficiency with mobile devices. The application will be easy to use, and users of all levels should be able to use the application without difficulty.

This device will ensure that users can easily determine the best parking lot to travel to and reduce frustrations when visiting densely populated areas. The admin will manage their associated parking lot(s) with the use of a web interface to update parking lot information and ensure that the application is functioning correctly. If there are issues, they will be able to determine cause and resolve issues accordingly.

1.3.3 Experience with Similar Applications

- Regardless of proficiency with technology most mobile users have experiences with android and iOS applications and this application should be familiar to users that have used applications feature maps in the past.
- Most users will be familiar with applications such as Yelp, or (another APP) where users use a map to find and select locations.

- Admins will be required to initially set up their raspberry pi and use the web application to set where parking spots are. This will be a simple click and drag with the mouse much like the desktop experience for all modern operating systems. It will also be familiar to them because of similarities to other admin software.

1.3.4 Task Experience

- Most users will be familiar with the experience of searching an area, selecting a location, and viewing the location information, and selecting a location from a list of locations.
- Admins will be familiar to clicking and dragging to initially setup parking spots and other functionality of the web application.

1.3.5 Frequency of Use

- Application will be installed once and then used as frequently as a user has a need to travel to an area where parking can be difficult. Majority of users will be infrequent however there is an expected minority of users that will use the application extensively.
- Admins will not be required to use the web application after initial setup and when issues arise from the system.

1.3.6 Key Interface Design Requirements That the Profile Suggests

- Easy to navigate UI
- Ability to see a number of open spots in parking lot
- Easy Start Up
- Easy Search Functions

1.4 Use Case Diagram

Below is an image of the use case diagram showing both the standard user and admin user along with their interfaces.

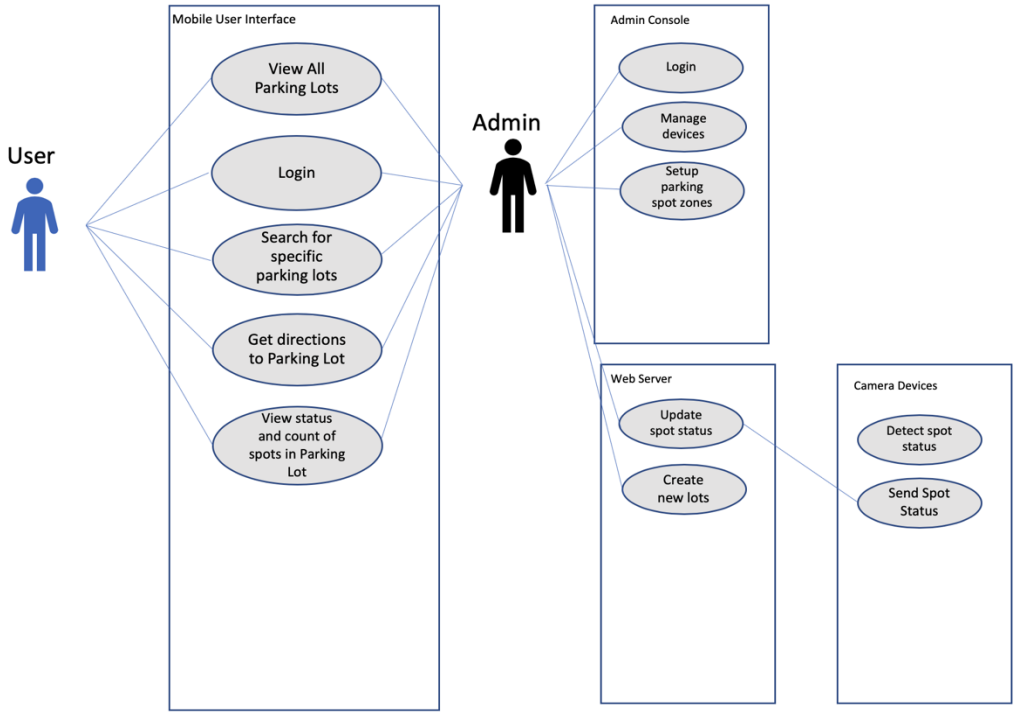


Figure 1: Use Case Diagram

1.5 Design Objectives

1.5.1 Project Goals

- Raspberry Pi and camera detect spots
- Raspberry Pi sends information to database
- Server routes information from raspberry pi to database
- Server routes information from database to mobile apps
- Mobile apps show user available spots in area
- Admin portal allows admins to setup raspberry pi and spots

1.5.2 Dropped Goals

- Data analytics – scope of this project would be too large for time constraint
- Event integration – scope of this project would be too large for time constraint

1.6 Methodology / Technical Approach

By breaking our project down into five separate applications, it has allowed us to streamline our development process and develop features for all five simultaneously. The heart of our project is our API server which handles all requests from the raspberry pi's to the database and from the database to our mobile applications. Our second project is the SQL Server running on a separate machine. The raspberry pi project handles all image processing and determining if a spot is available or not. The mobile application project is what our end user will see. It retrieves information from the API server and displays it in a user-friendly way. The last project is the admin web project

which is used for initial setup of the raspberry pi’s and maintenance and overview of them.

2.0 PROJECT MANAGEMENT

2.1 Project Budget

SpotCheck is expected to cost a total of 132,900.00 for a breakdown of individual expenses please see figure 2.

SPOTCHECK

Expenses

OPERATING EXPENSES	COST
Linux Test Server	500.00
Raspberry Pi	160.00
Windows Test Enviroment	300.00
Suits	900.00
Wages @ $32.5 per hour	131,040.00
Total Operating Expenses	132,900.00

Figure 2

2.2 Project Objectives / Deliverables

2.2.1 Project Milestones

Fall Semester Milestones		Spring Semester Milestones	
09/03/2019	Initiation and Research Phase	01/06/2020	Alpha Test Phase
09/10/2019	Design Phase	01/21/2020	Deployment Phase
09/23/2019	Team Contract Created	02/03/2020	Beta Security Testing
10/01/2019	User Profile Drafted	03/02/2020	Redesign / Update Phase
10/13/2019	Project Abstract Created	CANCELLED	Spring Presentation
10/21/2019	Elevator Speech	04/14/2020	2020 IT Expo
11/22/2019	Implementation Phase		
12/02/2019	Fall Presentation		
12/13/2019	Alpha Security Testing Phase		

2.3 Project Schedule

2.3.1 Gantt Chart

Below is an image of the project schedule of SpotCheck for the entire year, covering both fall and spring semester.

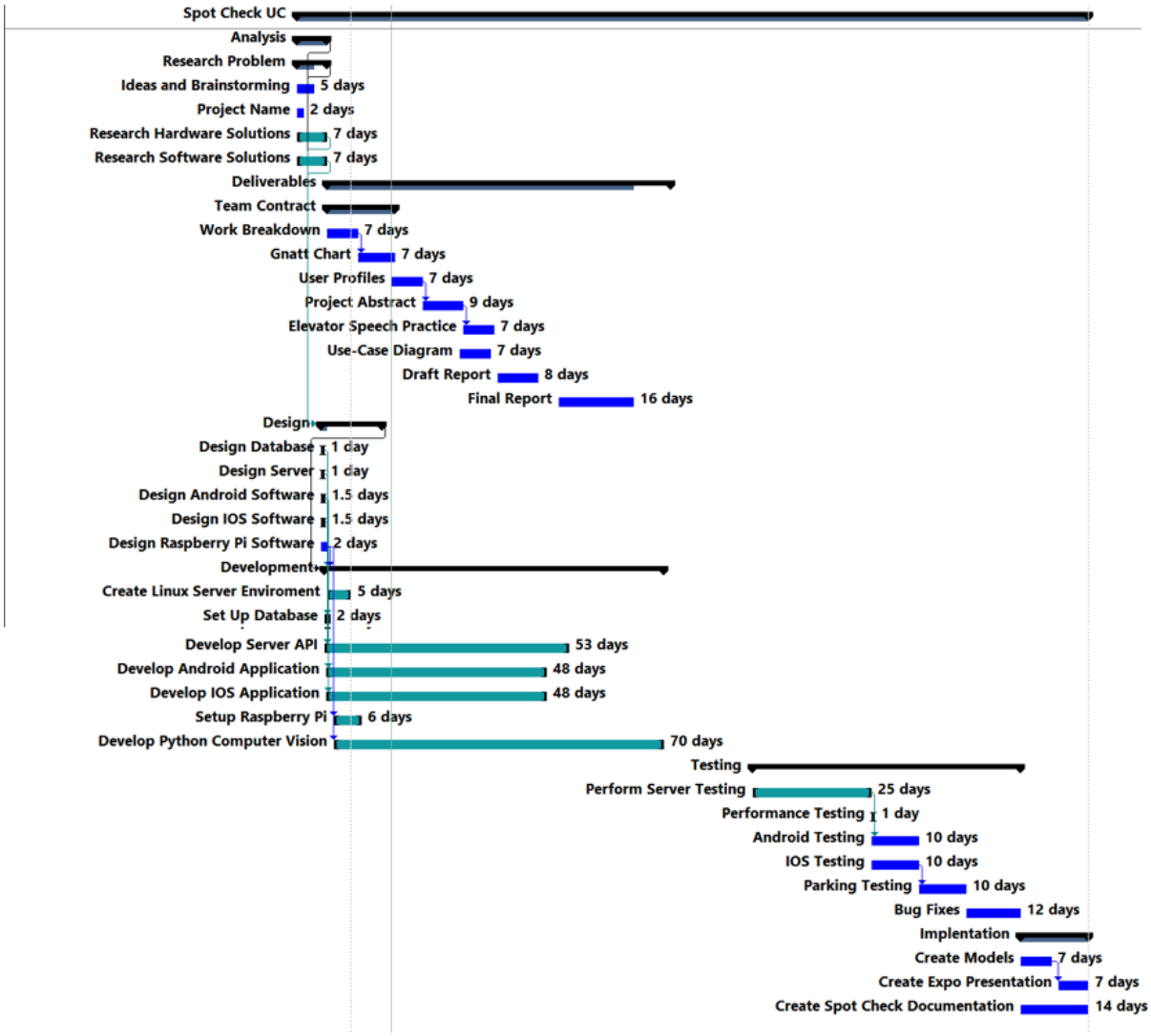


Figure 3: Gantt Chart

2.3.2 Work Breakdown Structure

Below is the work breakdown structure of SpotCheck showing how and when work will be completed throughout the duration of the project.

1	**Spot Check UC**	**168 days?**	**Mon 8/26/19**	**Tue 4/14/20**
1.1	**Analysis**	**7 days?**	**Mon 8/26/19**	**Tue 9/3/19**
1.1.1	**Research Problem**	**7 days**	**Mon 8/26/19**	**Tue 9/3/19**
1.1.1.1	Ideas and Brainstorming	5 days	Mon 8/26/19	Fri 8/30/19
1.1.1.2	Project Name	2 days	Mon 8/26/19	Tue 8/27/19
1.1.2	Research Hardware Solutions	7 days	Mon 8/26/19	Tue 9/3/19
1.1.3	Research Software Solutions	7 days	Mon 8/26/19	Tue 9/3/19
1.2	**Deliverables**	**74 days**	**Wed 9/4/19**	**Fri 12/13/19**
1.2.1	**Team Contract**	**14 days**	**Wed 9/4/19**	**Mon 9/23/19**
1.2.1.1	Work Breakdown	7 days	Wed 9/4/19	Thu 9/12/19
1.2.1.2	Gnatt Chart	7 days	Fri 9/13/19	Mon 9/23/19
1.2.2	User Profiles	7 days	Mon 9/23/19	Tue 10/1/19
1.2.3	Project Abstract	9 days	Wed 10/2/19	Sun 10/13/19
1.2.4	Elevator Speech Practice	7 days	Mon 10/14/19	Tue 10/22/19
1.2.5	Use-Case Diagram	7 days	Sun 10/13/19	Mon 10/21/19
1.2.6	Draft Report	8 days	Thu 10/24/19	Mon 11/4/19
1.2.7	Final Report	16 days	Mon 11/11/19	Mon 12/2/19
1.3	**Design**	**14 days**	**Mon 9/2/19**	**Thu 9/19/19**
1.3.1	Design Database	1 day	Mon 9/2/19	Mon 9/2/19
1.3.2	Design Server	1 day	Mon 9/2/19	Mon 9/2/19
1.3.3	Design Android Software	1.5 days	Mon 9/2/19	Tue 9/3/19
1.3.4	Design IOS Software	1.5 days	Mon 9/2/19	Tue 9/3/19
1.3.5	Design Raspberry Pi Software	2 days	Mon 9/2/19	Tue 9/3/19
1.4	**Development**	**73 days**	**Tue 9/3/19**	**Wed 12/11/19**
1.4.1	Create Linux Server Enviroment	5 days	Wed 9/4/19	Tue 9/10/19
1.4.2	Set Up Database	2 days	Tue 9/3/19	Wed 9/4/19
1.4.3	Develop Server API	53 days	Tue 9/3/19	Wed 11/13/19
1.4.4	Develop Android Application	48 days	Tue 9/3/19	Thu 11/7/19
1.4.5	Develop IOS Application	48 days	Tue 9/3/19	Thu 11/7/19
1.4.6	Setup Raspberry Pi	6 days	Fri 9/6/19	Fri 9/13/19
1.4.7	Develop Python Computer Vision	70 days	Fri 9/6/19	Wed 12/11/19
1.5	**Testing**	**57 days**	**Tue 1/7/20**	**Wed 3/25/20**
1.5.1	Perform Server Testing	25 days	Tue 1/7/20	Mon 2/10/20
1.5.2	Performance Testing	1 day	Tue 2/11/20	Tue 2/11/20
1.5.3	Android Testing	10 days	Tue 2/11/20	Mon 2/24/20
1.5.4	IOS Testing	10 days	Tue 2/11/20	Mon 2/24/20
1.5.5	Parking Testing	10 days	Tue 2/25/20	Mon 3/9/20
1.5.6	Bug Fixes	12 days	Tue 3/10/20	Wed 3/25/20
1.6	**Implentation**	**14 days**	**Thu 3/26/20**	**Tue 4/14/20**
1.6.1	Create Models	7 days	Thu 3/26/20	Fri 4/3/20
1.6.2	Create Expo Presentation	7 days	Mon 4/6/20	Tue 4/14/20
1.6.3	Create Spot Check Documentation	14 days	Thu 3/26/20	Tue 4/14/20

Figure 4: Work Breakdown Structure

3.0 TECHNICAL ELEMENTS

3.1 Network

This application will utilize http rest calls to communicate to and from the central server environment. The mobile and python applications will make use of HTTP requests to the central server to send and retrieve information. This server will then communicate as necessary with the SQL server to create, read, update or destroy data as necessary.

3.2 Application

The SpotCheck Server API is running on an Ubuntu Server and hosted by a Tomcat Instance. The team has utilized a physical server maintained and operated by Ryan Bunker. The server will act as a central repository for all requests for data from the mobile application and the Raspberry Pi Cameras. It will run on Java 1.8 utilizing the Spring Boot Framework with dependencies consolidated using Maven.
The mobile application will be written using Xamarin. Due to the constantly changing nature of the data a mobile noSQL database will not be used for mobile data.

3.3 Database

The entire application will use a SQL server database hosted on a Windows 10 environment. The server will handle all CRUD requests and make the necessary calls to the server for these functions. SQL was chosen because of the high number of possible calls from mobile and Camera applications.

3.4 Security

- API adds additional layer to accessing data
- No data is sent from the mobile user to the database
- Images taken from the Raspberry Pi never leave the Pi

3.5 Other

Our admin portal webpage will use ASP.Net as its backend server control. This will allow for dynamic data to be shown and sent via the API. On the front end, modern webpage technologies including Javascript, Jquery, HTML and Bootstrap will be deployed.

4.0 TECHNICAL DIAGRAM

4.1 Application Architecture

Below is a quick breakdown of the various technologies that the project will use. (List is subject to change as requirements evolve).

Server Code:

- Java/Spring Boot
- Apache Maven

Linux Server Environment

- Apache Tomcat
- Ubuntu

Database Server

- Windows Server hosted on Microsoft Azure
- SQL Server

Mobile Application

- Xamarin
- Google Maps API

Hardware

- Raspberry Pi
- RaspberryPi Camera
    - Python
    - Open CV

4.2 Application Diagram

Below is the application diagram for SpotCheck showcasing how the different parts of the system interacts with each other.

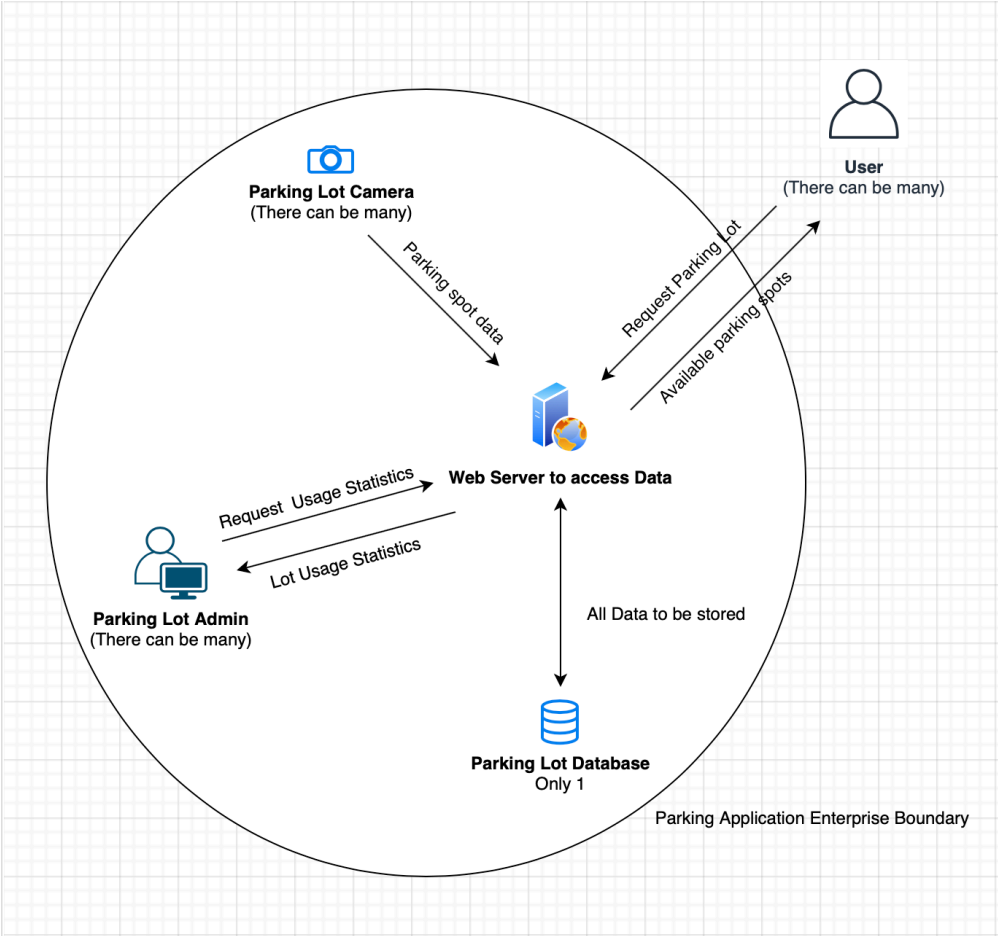


Figure 5: Application Diagram

4.3 Screenshots / Visuals

Android Application:
This screenshot shows what the user sees when searching for parking on the Android application.

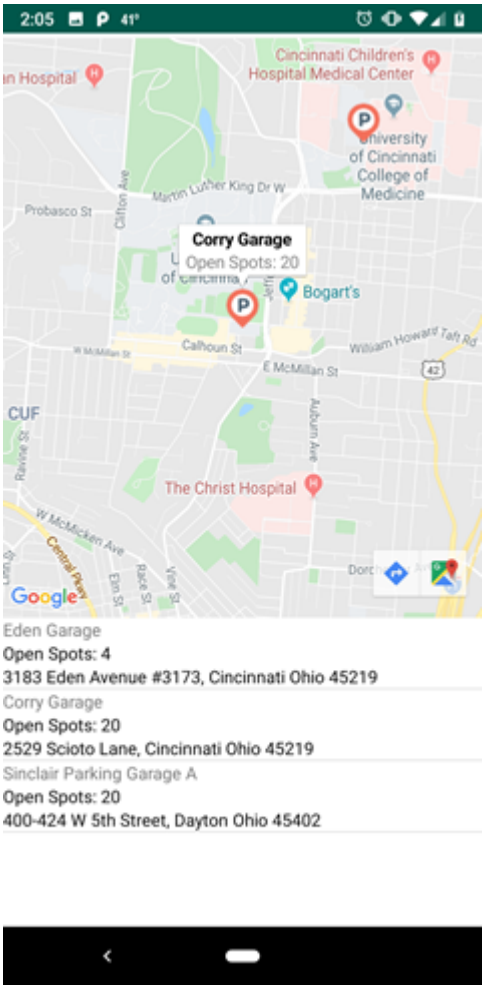


Figure 6: Android Application Screen

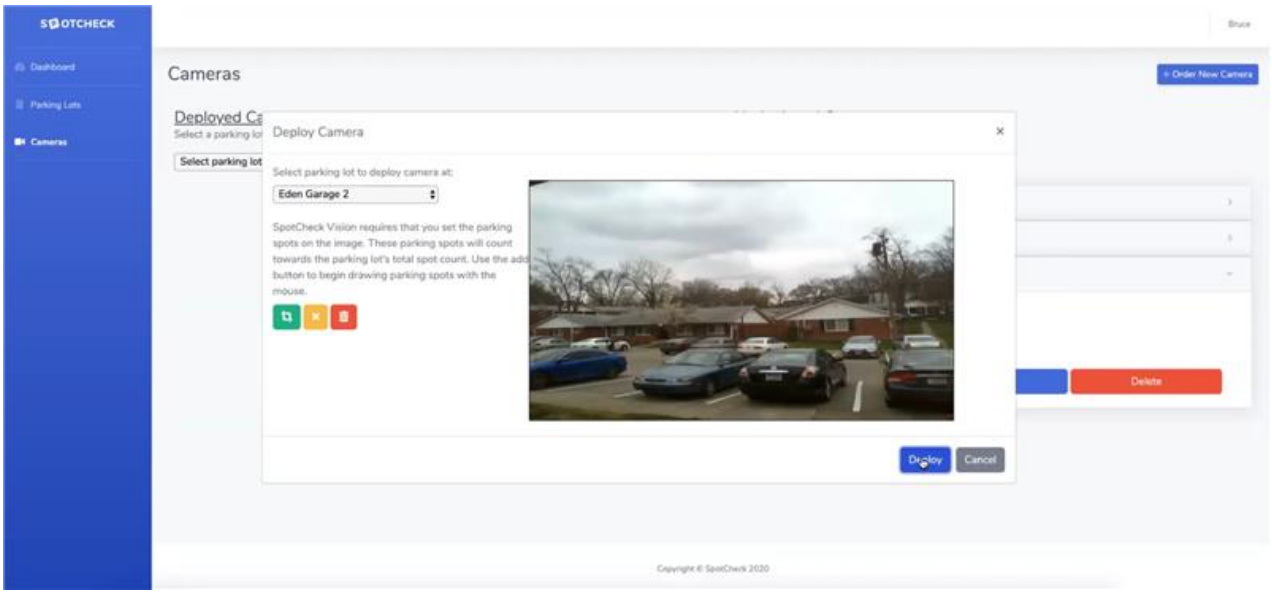
SpotCheck AI Vision:

This screenshot shows how the SpotCheck Vision AI software is able to distinguish between available and closed parking spots using its trained model set of vehicles.

Figure 7: SpotCheck AI Vision Screen

Admin Webpage:

This admin webpage screenshot illustrates how a user is able to add parking spots to a parking lot using an image from the SpotCheck AI Vision software.



5.1 Overview

For our testing methodology, we used two approaches as our overall testing strategy:

1. Manual quality assurance testing with documented test cases
2. Team-based roundtable ad-hoc “exploratory” testing
3. Junit Tests written into server code

For our first approach, we used a manual quality assurance (i.e., QA) testing approach. When we created a feature in our system, we also created one or more junit tests verifying that the changes were functioning properly

QA Testing workflow:

1. All team members shared in testing the application
2. The team members will run the QA Tests as they build the feature, to help capture issues during design/coding time.
3. Once the team member releases the code that pertains to a feature, that team member marks the relevant GitHub story as ready for test
4. Another team member will test the feature for errors
5. If the QA Test fails, the tester will open a new Bug work item and inform the team.

Developers who developed a story were not to test their own code as they are unlikely to find issues as quickly as someone who does not understand the code the way the developer does. This allows bugs to be caught and fixed faster.

We made sure to test for the happy path. Making sure that the code is working as expected for most of the code base. We make sure that for normal use the users are not running into any unexpected errors or crashes that would cause headaches for the users.

We are also making sure to test for edge cases that the original developer didn't think of. This way we try to stay ahead of issues that might be found when the application is released to the users.

Part of the testing will include forcing errors to happen in the code and seeing how the code handles those errors. This will allow us to see crashes and problems before they are released to the end users.

5.2 Objective

Our Test Strategy had to accomplish the following:

1. All major features must have QA Test Cases covering every logical End User step of the "happy path" workflow, and some of the error conditions as well.
2. The entire collection of QA Test Cases must account for all End User roles (e.g., Program Manager, Chapter Leader, Employee, Report Reader).
    a. Each *individual* QA Test Case does *not* have to account for all roles, but the overall *collection* of QA Test Cases must. For example, one Test Case can deal with the Program Manager role only.
3. All failed QA Test Cases and issues found during Exploratory Testing must result in a new Bug work item, or a plan in writing to address the defect.
4. All Bug work items are reviewed, fixed, and closed before IT Expo.
5. All QA Test Cases must eventually pass before the final release and IT Expo.

5.3 Test Cases

Test Cases:

Shown in Figure 9 are the results a successful run of the junit tests in the SpotCheck Server.

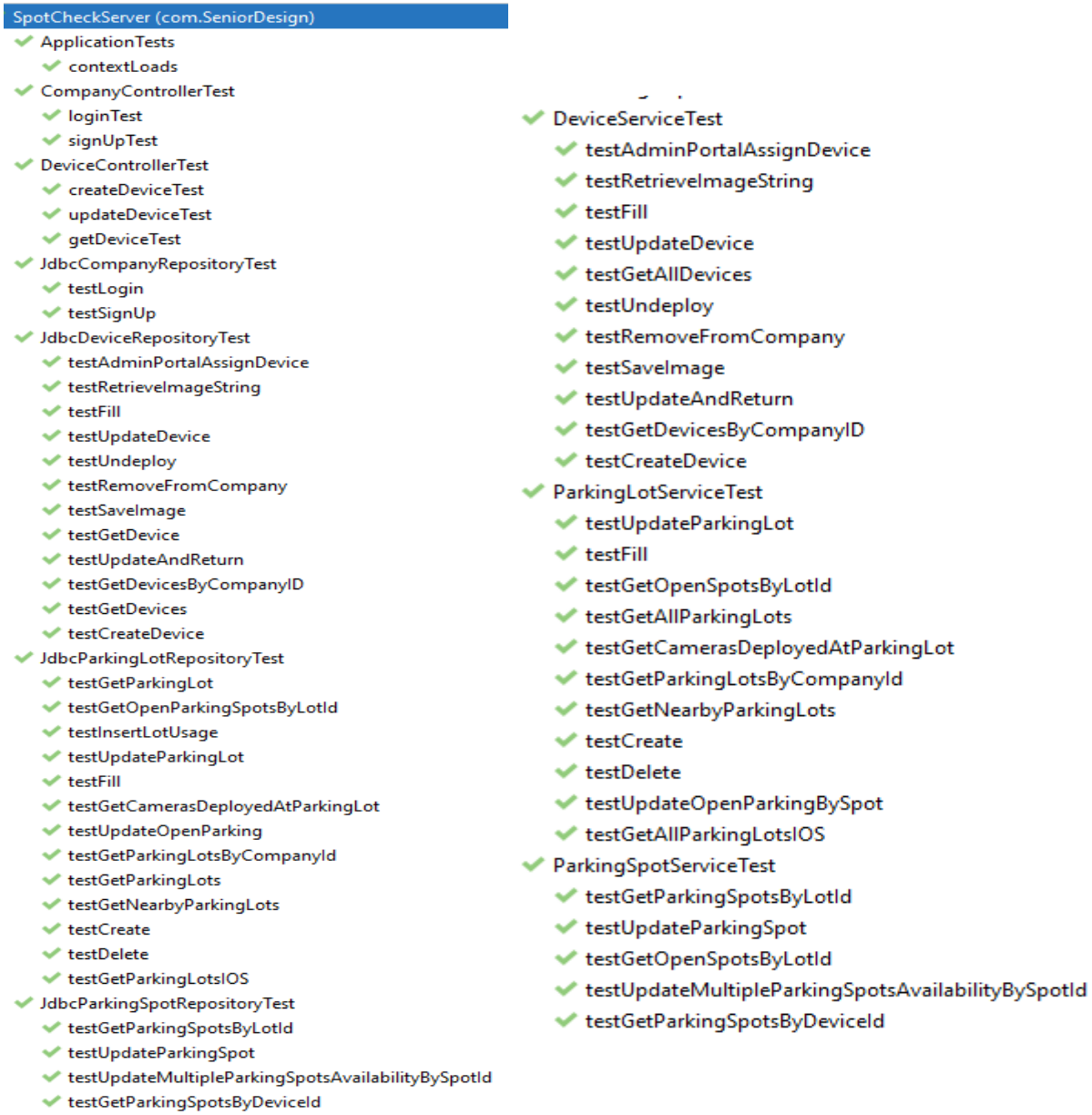


Figure 9: Test Cases

Another approach was ad-hoc testing when the team members were developing the code. As team members used the code and wrote the code and used the application, they would find errors and see problems that would exist in the code base. The team members were expected to fix the problems that they found this way so that they would not be present when released to the users.

We as a team would also get together and try and break the code, We would introduce null errors or enter letters into values that expected only numbers to make sure that the code handled these errors in a graceful way. The team meets every week and would discuss what has been developed over the last week and we would spend a little time trying to break the code. Any errors found were documented on the Github issue board so that they could be addressed.

5.4 Scope of Testing

Our unit Tests cover all major features however for the admin console we did not include any automated testing and instead testing the code manually. Our testing strategy included testing the admin console for any errors, testing the backend using Postman, and testing the camera using objects designed to trick the system.

5.5 Logging Tests and Procedures

As stated above, we took two approaches, manual QA Test Cases, and team-based roundtable Exploratory Testing.

Regardless of which of the two approaches we worked on, if we found an issue or a bug, we opened a Bug work item in TFS. Then one of the Developers assigned themselves the Bug work item, reviewed it, and fixed the bug.

We kept our QA Test Cases, Test Run data (e.g., pass or fail), and Bug work Items in GitHub so that they could be properly documented and taken care of before they become major problems.

Each team member was required to perform some QA Testing each week with the goal of getting every QA Test eventually executed. We did not assign required times for each person to perform QA Testing, as we wanted to leave it up to each person to decide what time would work best for each of us.

For Team meetings the team would meet every Monday either after class or at 6 in Teachers College at University of Cincinnati.

5.6 Test Results

Testing in agile projects is different than in waterfall projects. We don't have "exit criteria" because there is no exiting from one stage in a project to another in a waterfall sense.

Instead we simply write the tests, code, run the tests, fix bugs, and then run the tests again until they pass. At the end of our project, 100% of our tests passed.

5.7 Lessons Learned During Testing

The team that developed the code is experienced and we all have experience working as developers. The code has some bugs buy nothing that has caused major

slowdowns. As a result, we have not had to refactor due to poor design or poor coding. We have had plenty of time to develop new features.

Due to the team members experience as full-time developers we had a lot of experience dealing with software and any problems that might occur when developing code. We have been able to use those experiences to quickly develop code and stand up environments. Bugs have been easy to solve as we have experience debugging enterprise software. We are confident in our environment and our codebase to be relatively error free or at least handle errors gracefully.

6.0 CONCLUSION

6.1 Fall Semester 2019

In the implementation of this solution we have learned about structuring backends of applications in developing our API running on the server, in addition we have learned about the implementation of computer vision in modern applications, and Kotlin development for mobile applications. We have created many of the required project deliverables including the work breakdown structure, Gantt chart, and project budget.

6.2 Spring Semester 2020

Over the course of the spring semester, we were able to refactor much of the existing code to perform much more efficiently. This included making changes to our API to allow us more control over the information being sent to us by our camera devices and the mobile application. We also decided as a group to transition to using Xamarin for our mobile application development over XCode and Android Studio. This change would allow us to make simultaneous changes to both applications at the same time. As the semester finished, the Covid-19 epidemic began to affect much of our workflow, but we were able to present at the digital 2020 IT Expo without issue.

6.3 Problems Encountered

The middle of the semester brought unique challenges as the group was forced to deal with the effects of the covid-19 pandemic. At the time of writing two members of

the team were employed full time as software developers and the amount of time available to dedicate to the project was significantly decreased.

The team also decided to switch from kotlin/xcode to a unified code base with Xamarin however this switched proved harder then anticipated and it delayed completion of the mobile application and the team was unable to produce a working iOS app.

6.4 Future Recommendations

If we were to start over from the beginning and do this project all over again, there are many things that we could do differently to make ourselves more efficient. The main issue that we ran into was switching from XCode and Android Studio to Xamarin at the halfway point of the year. At that point we had put hours of work into both the iOS and Android versions of the mobile app, but we were having issues with the design being different on both apps. By switching to Xamarin, we were able to write the application in one language as opposed to two and any new additions we created, would be created for both of the apps. Going back, we would have simply started with Xamarin to begin with, which may of gave us additional time to implement more features.

REFERENCES

Rivera, Carla. “College Campuses Are Working to Lessen Parking Pains.” Los Angeles Times, LosAngeles Times, 13 Dec. 2015, www.latimes.com/local/education/la-me-college-parking-20151213-story.html.

Morelli, Marcos. “Benefits of Using a Digital Parking Solution.” Medium, Medium, 24 July 2017, medium.com/@imprensadigipare/benefits-of-using-a-digital-parking-solution-1722c6cde37f.