Title Page

# Executive Summary

# Table of Contents

[Executive Summary i](#_Toc26130522)

[Table of Contents ii](#_Toc26130523)

[List of Figures and Tables iii](#_Toc26130524)

[Introduction and Statement of Problem 1](#_Toc26130525)

[Background Information 2](#_Toc26130526)

[Method of Work 3](#_Toc26130527)

[Management Plan 4](#_Toc26130528)

[Conclusion 5](#_Toc26130529)

[References 6](#_Toc26130530)

[Appendix 7](#_Toc26130531)

# List of Figures and Tables

# Introduction and Statement of Problem

# Background Information

The mobility of universities has become more problematic due to the increasing number of students that attend each year. The *mobility* of a university refers to the convenience and ability to travel around and into the campus. This immobility causes 67% of students and university staff to make the decision to travel by private vehicle (Dell’Olio). This directly causes the parking on campuses to become congested and troublesome even when there are other alternatives of transportation available.

Most universities’ information and student resources are scattered across various webpages that are often difficult to find and laborious to navigate. Too many institutions have either an incomplete mobile application or none at all. In a world where nearly everything is driven by technology, the majority of college students own a smartphone capable of downloading applications that can be used to benefit their education. A centralized university application is therefore needed so that students can freely navigate and reference important information in one place.

Changing environments and in some cases, moving far away from home, can be monumental for new students transitioning into a university setting. Locations of classes and other various resources can be stressful and hard to find for new students. Over the course of their college careers, only 54% of those students will graduate and receive a diploma in the United States (Hess). Reducing the confusion of how and where to find necessary information is paramount to student success.

The problem to solve is then threefold: optimize parking, assist newer students, and centralize resources into one usable mobile application. Through the already existing technology and current solutions, a university application is made possible. The software also has tremendous potential to send students push notifications on their devices. This allows university alert systems another method of distribution for the safety of students.

**Current Solutions**

Researchers have already attempted to mitigate the use of personal vehicles on campuses. A case study done at the University of Cantabria in Spain illustrates one possible attack to the problem. Through the analysis of data from preference surveys over the student body population, researchers found an optimal parking fare for the use of parking spots on campus. There exists an optimal price per parking spot that will maximize revenue for the university while influencing students to use more sustainable methods of transportation (Dell’Olio).

Installing monitoring systems for each individual parking space is inefficient and wasteful of resources. Most parking systems today require parking structures to be indoors to apply interconnected sensors, magnetic sensors, and other hardware necessary for an indoor environment. A university in Malaysia has constructed a smart parking system that utilizes computer vision to keep track of open parking spaces in real-time (Loong). This diminishes the need for individual parking space considerations as cameras can survey a large area.

*Computer vision* is the use of advanced software that tracks spatial changes in digital format using cameras or other optical sensors. The system uses very lightweight hardware that includes a Raspberry Pi model computer and a camera compatible with the Pi. The proposed system captures images that is processed by the software using the camera and computer. The results are then uploaded to a database where some mobile application can output the results to users who will have real-time access to the parking information (Loong).

The trend between bundling and unbundling features of major applications has followed both paths as seen by big technology companies in the past decade (Kapko). For example, the popular social messaging application Snapchat chooses to onload new features instead of unbundling. On the other hand, Google has chosen to offload messaging, productivity, and other aspects of their services into respective single-purpose applications. The argument can be made that the decision for combining or splitting features of an application is dependent on the customer use-case.

Multipurpose applications have everything the user needs in one place while single purpose applications are designed for one thing. Each direction of development has their respective advantages and disadvantages. For example, a multipurpose application grants a unified experience allowing data and features to work interconnectedly together. However, the user interface can become confusing and difficult to navigate. Single purpose applications allow for quick updates because of their simplicity. They also force users to download multiple applications if many components of the service are desired (Rios).

Google and many other applications have used Google’s map standard development kit. The application programming interface (API) takes care of complicated mapping features such as map gestures, the display of the map, and downloading of data (Google). An *API* is a set of computer functions that developers can use to communicate with some external resource. The interface also communicates with Google Map’s extensive database to accurately represent the layout of the real world. Developers can then make calls to the API to add markers, shapes, paths, and other graphical elements to achieve usability with the map.

**Drawbacks and Issues**

Influencing students to use public and built in transportation to navigate campuses lessens the problem. However, there remains a large portion of students who will decide to use private vehicles due to personal reasons and commuting distances. The current solution of raising parking pass prices addresses the number of students but neglects the efficiency problem of parking that busy campuses will still possess.

Live tracking of parking spots using computer vision and lightweight hardware aims to solve this very problem and does it successfully. The cost of a simple computer and camera for each general parking area is significantly less than implementing current methods of tracking parking. The minor issue that arises from this system is the use of another outside application; the goal is to centralize parking resources into one executive location.

# Method of Work

The USU Parking and Transportation App is designed to help provide students with an easy way to access information to ease the stress of starting at a new university. The different components of the app use functionality from different systems in order to bring that information to one place. How those systems are integrated, alternatives, benefits they provide, and challenges are detailed in the section.

**USU Banner Integration**

The USU Parking and Transportation App takes advantage of several of the *Application Program Interfaces (APIs)* that USU provides. These APIs gather information from USU’s servers and report back the requested USU or Banner information to the app. The APIs included in the app are as follows:

* Student account ID
* USU account messages
* Student schedule
* Classroom buildings
* Instructors
* Live bus routes

The app administrator is given permission along with a set of credentials in order to connect to each of the APIs. Each time a request is made to retrieve information these credentials must be included and verified in the request. Once the request is verified the server responds with the correct information, which the app then displays on the appropriate page [].

The student schedule request for example is sent via a GET request, which is a method of *Hypertext Transport Protocol (HTTP)* used for communicating between clients and servers. The credentials given, or API key, is sent along with the GET request. The server then verifies that the key is valid and responds by sending the student’s schedule in either *JavaScript Object Notation (JSON)* or *Extensible Markup Language (XML)* format. The information from the JSON or XML object is then parsed out by the app. It is then formatted to be viewed by the user, or in this case it is used to form a route from the user’s current location to the classes reported in the returned object.

**Class Map**

The class map utilizes Google’s map API for most of its functions. Similar to USU’s API, the app will require an API key in order to send requests to Google’s servers. The key will be added into the app’s program code so it can be given when a request is sent. Google’s API has many features which are not pertinent to this project, so restrictions will be placed on the API key to ensure only authorized requests can be made. Restrictions are made in Google’s Cloud Platform Console, which is the same area where an API key can be requested [].

The map itself is added to the application with a Fragment object, which is simply an element to attach the map element. Once the map element is placed it will automatically handle operations such as: connecting to the Google Maps service, downloading map tiles, displaying tiles on the device, displaying various controls such as pan and zoom, and responding to pan and zoom gestures my moving the map and zooming in or out [].

The app then uses the schedule and course building information provided by the USU API to apply routes to each class. Paths can be placed on the map by placing coordinates, which are then connected by the map. This is accomplished by creating a poly line object and feeding it the coordinates in the form of latitudes and longitudes []. The specified points will be plotted out before hand over intersections of USU’s walking paths for the shortest path to be calculated.

The shortest path is found by making a map of possible nodes that can be traveled to from one point to another. This is accomplished by assigning a cost or a weight to each segment between nodes based on walking time. The shortest path is then chosen by finding the combination of paths to nodes between the start and end points. The weight of each segment will also be calculated based on how populated a certain path is. For example, a longer path might be given a lower weight if the traffic is light enough to observe a faster travel time.

**Parking**

The parking portion of the app uses cameras and AI coupled with machine vision, this is used to mark if a given parking spot is occupied or empty. This is a much cheaper and more easily implemented solution for detecting open spaces than other methods. A camera attached to a Raspberry Pi is placed facing towards an area of a parking space, and the AI determines if a n area of the image contains a car or not.

This is achieved by using Canny edge detection, which is a method where a filter, such as a Gaussian filter, is applied to an image to detect edges on objects. This turns an image into a black and white image with high contrast, which can be easily detected by a computer. The edges detected can then be determined to be certain shapes, such as a car, much more easily.

Certain areas of an image can then be marked as parking or driving areas, which will help eliminate false positive errors. The parking spaces will be marked as being empty or filled in real time and sent from each Raspberry Pi to the application. The parking information will then be updated on the map allowing the user to quickly see which spots are available.

**Alternatives**

One other option for displaying a map of campus was to simply create and integrate a map from scratch. This would allow us to provide our own proprietary implementation without having to rely on an external API. The downsides to this are cost of work, maintenance, and less versatility. Google’s Map API requires very little work to get it up and running, which cuts down on development cost and time. Once it is implemented it also requires less maintenance, as Google maintains their own API. Another reason to choose Google’s API is it allows the map to be used of campus for both normal map functions and creating paths to the student’s classes.

An alternative considered for the parking portion of the app was to install a sensor in each stall to determine if it was occupied. Several different methods were explored including pressure sensors, microwave radar, and ultrasonic sensors. Pressure sensors were initially selected over microwave radar and ultrasonic sensors due to its relatively simple design. The problem with all three solutions which ultimately lead to the current selection of machine vision was cost. The rejected solutions are simply too costly to implement and maintain. The current solution requires only a single Raspberry Pi and camera for a defined area and can be easily and cheaply replaced if damaged.

**Benefits**

The USU Parking and Transportation App will help provide valuable information to new students, which will help alleviate the stress of moving to a new area. The system will achieve this by combining several resources into one area to reduce the need to search for what is needed. Each student will have one central area they know they can go to answer many different questions and problems they may have.

**Challenges**

One challenge presented by the class map is keeping the shortest path algorithm up to date so it can be as efficient as possible. The weights assigned to each segment of a path will need to be updated depending on the current foot traffic. Depending on student class schedules, how many students use the app, and possible construction or other blockages will need to be considered as these metrics change. The paths will need to be updated each semester as well as updating individual path segments to account for impediments.

# Management Plan

This section outlines the proposed schedule, budget, and personnel necessary to complete this proposal. The proposed schedule will last approximately 30 weeks and is split into 5 stages. The proposed budget totals $301,260. The team working on the project consists of four university students with adequate experience to complete the task.

**Schedule**

The project is divided into five different stages: requirement gathering, design, development, testing, and deployment. Each stage will be accounted for by a certain number of one-week sprints. In computer science, a sprint is a time management technique which distributes tasks to individuals in order to better organize the tasks. The total amount of time estimated for the entire project is around 30 weeks, or 7 months. See Figure \_\_ for a visual representation of the general outline for the schedule.

**Stage 1: Requirement Gathering.** The initial stage of the project will last three weeks and will consist of studying and researching current university applications. Doing so provides an opportunity for the team to discuss plans for the project before the team members break into smaller, specialized sub-projects.

**Stage 2.1: High Level Design.** The high-level design stage will last two weeks. This stage will provide an opportunity for each individual to transition from group research to individual research. This stage will end with a peer review process where each individual’s design plan is approved by the rest of the group.

**Stage 2.2: Low Level Design.** The low-level design of the project, lasting two weeks, will be more separate. During this stage, each individual will be working alone, with support from the rest of the group, to fine tune the details of their sub-project.

**Stage 3.1: Development.** The development stage will take the longest of all the stages. Lasting 12 to 15 weeks, this stage will consist of each individual working on their own sub-project, with peer reviews at the end of each week-long sprint.

**Stage 3.2: Integration.** The integration stage will last two weeks. This stage will consist of each sub-project being fit into the larger project, like puzzle pieces. Each individual will continue working on their sub-project, but this stage will have a larger, team-based focus.

**Stage 4: Testing.** The testing stage will last four weeks. This stage will be for the team to test the integration and development of each of the other individual’s sub-projects. When issues are found in the application, they will be reported using a ticket reporting system. Those tickets will be distributed to the individual in whose sub-project the issue was found.

**Stage 5: Deployment.** The final stage of the project will last two weeks and will focus on moving the working application to its servers. The final portion of this stage will consist of more testing to verify that everything is working as it should.

**Figure \_\_:** Timetable of each stage relating to how many weeks it will take from the start date

**Budget**

This section contains all of the proposed costs for the entire project. The costs fit into two different categories: labor and tools. The majority of the costs will come from development compensation for the software engineers. Each engineer will be paid $2,500 per week or $62.50 per hour. This rate is fixed and applies to each engineer. The tools section includes software development tools which will facilitate collaboration between the engineers. See Table \_\_ for a visual break-down of costs.

**Table \_\_:** Break-down of total costs for the application

|  |  |  |
| --- | --- | --- |
| **Stage** | **Amount (hours)** | **Total Cost ($)** |
| Stage 1 | 480 | 30,000 |
| Stage 2 | 640 | 40,000 |
| Stage 3 | 2,720 | 170,000 |
| Stage 4 | 640 | 40,000 |
| Stage 5 | 320 | 20,000 |
| Subtotal | 4,800 | $300,000 |
|  |  |  |
| **Item** | **Amount** | **Total Cost ($)** |
| Visual Studio Professional |  | 1,260 |
| Subtotal |  | $1,260 |
|  |  |  |
| **TOTAL** |  | **$301,260** |

**Personnel**

The team that is building this application consists of four capable Computer Science majors currently attending Utah State University. Each team member has acquired experience, both academic and professional, which will aid to better the college experience for students on campus. See Appendix A for the resumes of each of the members of the team.

**Jason Boyd.** Jason is currently a junior working on his B.S. in Computer Science at Utah State University. Jason has worked on a variety of personal projects from iOS development to a dynamic personal website that has pushed his understanding of the web. His experience and devotion to mobile and web application development will greatly benefit the quality and usability of the university application project.

**Ryan Egbert.** Ryan is a junior in the Computer Science department at Utah State University. He plans to graduate with his B.S. in Computer Science in December 2020. Ryan has two years of professional development experience. His current position is with Space Dynamics Laboratory, a research center affiliated with Utah State University, working on developing web applications.

**Carter McGee.** Carter is a Computer Science major attending Utah State University. He is currently in his third year of his undergraduate degree and plans to graduate in May 2021. Carter is interested in working with computer vision and machine learning techniques. He hopes to move back to his hometown in Albuquerque, New Mexico, to work for Sandia National Laboratories after he graduates.

**Benji Stewart.** Benji will graduate with a bachelor’s degree in Computer Science from Utah State University. He has gained experience working on and troubleshooting issues with both hardware and software systems at Space Dynamics Laboratory at Utah State University. His coursework also provides a foundation for development of program and application systems pertinent to this project.

# Conclusion

# References

Ng Chiu Loong, Daniel, et al. “Machine Vision Based Smart Parking System Using Internet of Things.” *Telkomnika*, vol. 17, no. 4, Aug. 2019, pp. 2098–2106. *EBSCOhost*, doi:10.12928/TELKOMNIKA.v17i4.12772.

Vaupel, S., Taentzer, G., Gerlach, R. et al. Softw Syst Model (2018) 17: 35. https://doi.org/10.1007/s10270-016-0559-4

Stehle T, Riebisch M, A porting method for coordinated multi-platform evolution. *J Softw Evol Proc.* 2019;31: e2116. https://doi.org/10.1002/smr.2116

Hess, Abigail J. “Bill Gates: US College Dropout Rates Are 'Tragic'.” *CNBC*, CNBC, 10 Oct. 2017, www.cnbc.com/2017/10/10/bill-gates-us-college-dropout-rates-are-tragic.html.

Rios, Ronnie. “Single-Purpose or Multi-Purpose App – Which Option Is Better? - DZone Mobile.” *Dzone.com*, DZone, 22 Dec. 2017, dzone.com/articles/single-purpose-or-multi-purpose-app-which-option-i.

Kapko, Matt. “Social, Messaging Apps Struggle with Decision to 'Bundle' or 'Unbundle'.” *CIO*, CIO, 12 Mar. 2015, www.cio.com/article/2895773/social-messaging-apps-struggle-with-decision-to-bundle-or-unbundle.html.

Dell’Olio, Luigi, et al. “A Methodology Based on Parking Policy to Promote Sustainable Mobility in College Campuses.” *Transport Policy*, vol. 80, 2019, pp. 148–156., doi:10.1016/j.tranpol.2018.03.012.

Swanlund, David, and Nadine Schuurman. “Resisting Geosurveillance: A Survey of Tactics and Strategies for Spatial Privacy.” Progress in Human Geography, vol. 43, no. 4, Aug. 2019, pp. 596–610, doi:10.1177/0309132518772661.

“Overview: Google Maps Platform.” *Google*, Google, 2019, developers.google.com/maps/documentation/android-sdk/intro.

# Appendix – Team Resumes







