January 2018

# Atlas: A Pathfinding Application Toolkit

Trevor J. Valcourt
*Worcester Polytechnic Institute*

# Atlas

A Pathfinding Application Toolkit

Trevor Valcourt tjvalcourt@wpi.edu

Project Advisor
Professor Wilson Wong

## Abstract

Atlas is a web-based pathfinding framework designed for finding the shortest path between two locations. Using a component based approach built upon Facebook’s flux architecture, this framework will be useful in many applications where pathfinding is needed, such as university campuses, airports, museums or hospitals. The framework provides modeling tools to turn digital maps into a routable representation of the location to assist users in navigating unfamiliar places. A toolkit allows developers to select features they would like to include, such as operation hours, handicapped routes, or a directory listing. This application is deployable on any web compatible device, such as a desktop, smartphone, tablet, or kiosk.

# Table of Contents

## List of Figures

# 1. Introduction

Pathfinding applications are most often used on mobile devices. Google Maps averaged about 102 million unique users per month, and Apple Maps averaged 54 million for a combined total of 156 million users per month in 2016 (Statista, 2016). General navigation is very relevant in our everyday lives, but these major applications are primarily useful for traveling between outdoor destinations.

Traveling inside buildings or within a large campus is not easy and often relies on static maps, which for many users can be difficult to follow. It is easier for users to use an interactive map at a kiosk or access a webpage on their mobile device. Examples of static maps can be seen at most shopping malls, airports, or even businesses worldwide. Through personal observation, most implementations of interactive maps are poorly done visually, and there is not a standard method to create these applications. Current indoor navigation companies rely on specific hardware for location detection and ultimately end up becoming mobile applications that require large time and money investments to install. See section 2 on research for more information.

The goal of this project is to develop a framework for building a pathfinding application. Using a web-based building toolkit, users are able to create a custom pathfinding application suitable for their location. An initial proof of concept for this project would be for a CS3733 Software Engineering class to be able to create their own pathfinding application. Their application would be for a single location, such as a hospital or the WPI campus like in previous years. From there, it would be extended to support many real world applications such as the ones listed above. The toolkit provides a way of creating routable paths on an existing map and gives the user the ability to add features relevant to their real world application. An example of this might be a time-table for an airport, or a business's hours of operation. The framework should be easy to use and focus on the software side of a pathfinding system. The aim is not to track a user's every movement, but instead simply get them where they need to go.

## 1.1 Project Idea Background

The idea of this project comes from the CS3733 Software Engineering class at WPI. Over the past two years, CS3733 students have been developing pathfinding applications for the WPI campus and Brigham and Women's hospital. Their applications were standalone JavaFX programs whose primary function was to create a navigation tool for each respective campus. This included traversing indoors across multiple buildings and floors. Students were able to add any additional features to make their application more useful. Atlas is framework that future CS3733 students could use to add features to, and build their own navigation applications. After a few iterations of the CS3733 course, it was apparent that there were common features every team developed. One goal of Atlas is to remove the development stage of building this repeated platform and allow students to use Atlas as a basis instead. Many of the features and design choices for Atlas have been adapted from observing student's work in the course. It is important to note that the author has taken the course and served as a team coach.

## 2. Background Research

The focus of the background research section was to identify the following information: a few of the leading companies who do pathfinding, how they do it, and how successful they have been in the industry.

After conducting preliminary research, the number of indoor navigation companies was much higher than initially expected. Many of the companies have been established within the past six years. This means that indoor navigation is still a new field and there is room for more competition.

### 2.1 Infsoft

Infsoft is a German company who specializes in indoor navigation systems. Their background research is very comprehensive, and they have posted all of their information on the company website regarding how indoor navigation works in general, as well as their product. Infsoft explains the workflow from client-based positioning and server-based positioning, and compares the accuracy between Wi-Fi, Bluetooth Beacon tracking (BLE), Ultra-wideband (UWB, a type of radio system), and Radio Frequency Identification (RFID) tracking. The process for getting a navigation system installed seems a bit too complicated for its actual functionality. There is a lengthy 8-step process to go from inspection to rollout, and this may only be for a proof of concept; not the full product.

Infsoft focuses on a hardware based implementation. They have many different methods for calculating indoor position, but they all converge to a hardware installation. On top of that, their systems are reliant on mobile ubiquitous computing, so users wishing to interact with the navigation system need to own a smart device and have Infsoft's application installed. There is also a non-mobile idea using Bluetooth or RFID tags to track location (Infsoft GmbH, 2017).

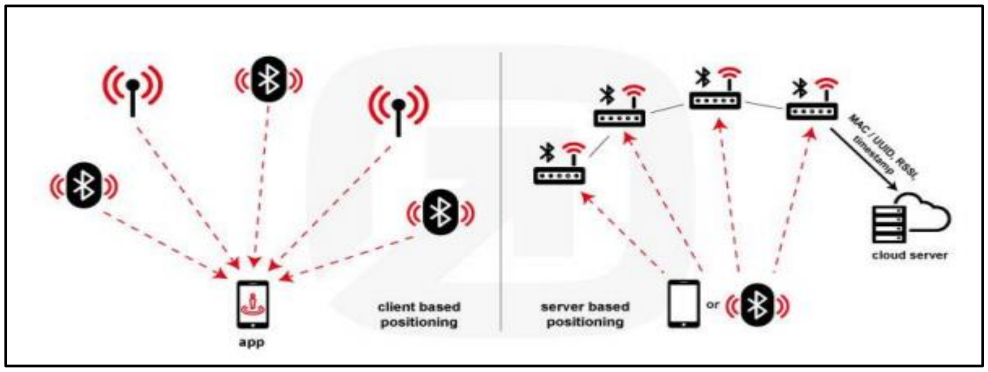


*Figure 1. Infsofts methods for designing an Indoor Positioning System.*

Some of the benefits of such a large project could be the strong analytics system provided. It has the ability to generate heat maps of where people go, or monitor what services are most used by the customers. They also have client vs server solutions as shown in Figure 1 which help cater to exactly what Infsoft customers needs.

The downsides to this scale of navigation is that it seems like there is a lot of overly complicated work between project initiation and project rollout. Also, the live demo on the company website did not work which was disappointing because it would be helpful to see what a final product might look like.

## 2.2 WiFiSlam

WiFiSlam is a startup company whose main focus was using WiFi to help determine indoor positioning where GPS falls shorts. This company was acquired by Apple in 2013 who have since created prototype indoor routing applications. Apple has been granted a few patents to pursue this field and have even claimed to create phone applications that can detect the floor a user is on based on WiFi signals and fingerprint detection (Crunchbase, 2016).

These applications require users with a smart device to be connected to WiFi to determine their location. Apple has was one of the first major companies trying to create seamless indoor and outdoor transitioning map devices and have had some success with this. Their main focus is automatic floor detection, allowing users to simply boot up the application and get going.

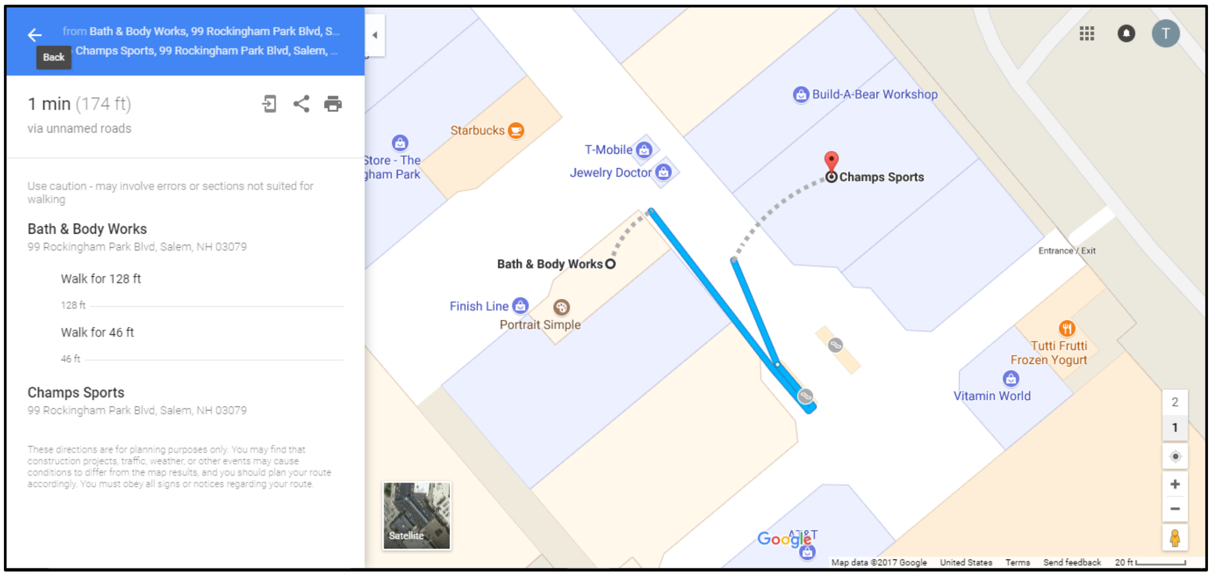
Their success came by detecting the floor of a mall a user was standing on using surrounding wireless signal sources and location fingerprinting. Apple's rationale is to make their user experience as simple as possible by updating the floor plan to the current floor automatically. In addition, this adds a third dimension to traditional static pathfinding which is only two dimensional. While Apple's proof of concept was successful, the technology still needs testing in different locations before it is ready for market (Patently Apply, 2016).

## 2.3 Google Maps

Google Maps currently has an indoor maps navigation system live on the main Google Maps application. It was implemented by showing floor plans of the insides of buildings as users zoom in closer to them. Once zoomed in far enough, floor level selection buttons will appear and upon switching floors, the indoor map changes but the main outdoor map stays the same.

Testing this feature was frustrating in trying to figure out where to go. There are currently a small number of buildings implemented, so the Rockingham Mall in Salem, New Hampshire was tested as well as the San Francisco International Airport. Upon choosing a *walking* path between two stores in the mall on the same level, Google Maps does a nice job of showing how to get there. The text directions on the other hand were not very advanced, often just telling the user to walk an arbitrary number of feet without turn directions.

The real problem comes when trying to route between two floors of the same building. Google Maps draws a confusing path that appears on **all floor levels.** This means that the path will draw itself through other stores and is extremely confusing to determine directions for the current floor. There is a dashed line sometimes, to presumably show that the path is not on the same level, but it can get really chaotic. On top of that, the text directions are not very helpful either. They skip major landmarks such as escalators or stairs and condense the directions into a single walk “x” feet. This means most of the time the directions do not even tell the user to change floors, however, in the case of the mall it will draw the path down an escalator even if the text direction does not explain it. Switching which floor level to display via the buttons does not help view the path any better and only makes navigation more confusing.


from Bath & Body Works, 99 Rockingham Park Blvd, S...
Champs Sports, 99 Rockingham Park Blvd, Salem, ...

1 min (174 ft)
via unnamed roads

Use caution - may involve errors or sections not suited for walking

Bath & Body Works
99 Rockingham Park Blvd, Salem, NH 03079

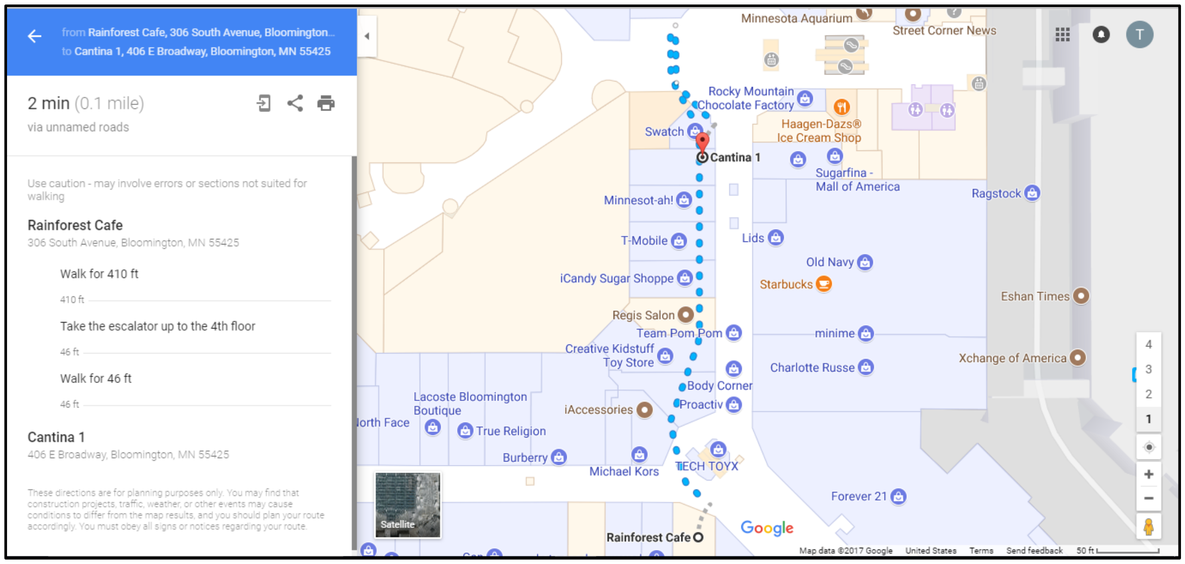
Walk for 128 ft
128 ft

Walk for 46 ft
46 ft

Champs Sports
99 Rockingham Park Blvd, Salem, NH 03079

These directions are for planning purposes only. You may find that construction projects, traffic, weather, or other events may cause conditions to differ from the map results, and you should plan your route accordingly. You must obey all signs or notices regarding your route.

*Figure 2. This shows a route going from a first floor store to a second floor store in the Rockingham County Mall. It is clear that the text directions are technically correct, but not helpful when have to use the escalator to change floors. Also, the entire route is drawn, however, as indicated in the bottom right screen, it should only be displaying the first floor.*


from Rainforest Cafe, 306 South Avenue, Bloomington...
to Cantina 1, 406 E Broadway, Bloomington, MN 55425

2 min (0.1 mile)
via unnamed roads

Use caution - may involve errors or sections not suited for walking

Rainforest Cafe
306 South Avenue, Bloomington, MN 55425

Walk for 410 ft
410 ft

Take the escalator up to the 4th floor
46 ft

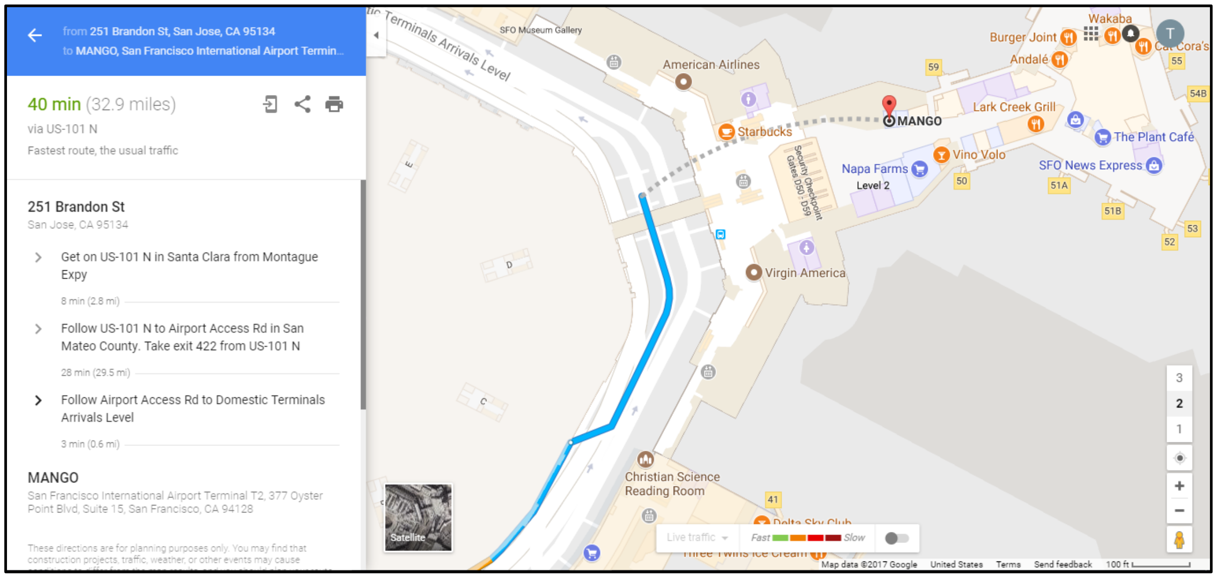
Walk for 46 ft
46 ft

Cantina 1
406 E Broadway, Bloomington, MN 55425

These directions are for planning purposes only. You may find that construction projects, traffic, weather, or other events may cause conditions to differ from the map results, and you should plan your route accordingly. You must obey all signs or notices regarding your route.

*Figure 3. This shows a route from the 1st floor to a 4th floor shop in the Mall of America. The route draws over other stores because it shows the full path even though we are only displaying the first floor. The directions are somewhat better than before as they indicate an escalator. The line drawn will turn solid if zoomed in more, but this figure is intended to capture the whole trip.*

Google Maps also suffers when trying to do outdoor to indoor transition. If a user wanted to route from their home to a store in the airport, they would have to select a route via car. The final result is a path that finds the nearest point geographically to the store reachable by car. There is no direction to park at the nearest parking lot and then transition to walking to the store. Most of the time, the route simply stops in the middle of the road and the directions end there. While technically the algorithm works correctly in that it gets the user to the closest point, it does not get the user to their destination and is simply not useful (Google, 2017).


from 251 Brandon St, San Jose, CA 95134
to MANGO, San Francisco International Airport Termin...

40 min (32.9 miles)
via US-101 N
Fastest route, the usual traffic

251 Brandon St
San Jose, CA 95134

Get on US-101 N in Santa Clara from Montague Expy
8 min (2.8 mi)

Follow US-101 N to Airport Access Rd in San Mateo County. Take exit 422 from US-101 N
28 min (29.5 mi)

Follow Airport Access Rd to Domestic Terminals Arrivals Level
3 min (0.6 mi)

MANGO
San Francisco International Airport Terminal T2, 377 Oyster Point Blvd, Suite 15, San Francisco, CA 94128

These directions are for planning purposes only. You may find that construction projects, traffic, weather, or other events may cause

*Figure 4. This figure shows a route from a residence to a store in the San Francisco Airport via car. The final point there is actually in the middle of the road and is technically the closest point to the store. The text directions, if expanded, simply ends by telling the user to turn right onto this road and then stops there.*

## 2.4 Project Justification

Indoor navigation companies tend to take a hardware approach. Companies want to be able to track a person's exact location within a few meters, or even centimeters. To do this requires invasive hardware within a company's building, and could require employees to wear technology that can display their location in real-time. Companies like Infsoft describe in great detail how great it is to find out where a person is, drop them off at a particular store on the way to their destination, or show them a certain advertisement. Their approach caters to businesses trying to profit as much as they can from customers and overlooks the user experience of a pathfinding system.

It may also be unethical in some cases to track people so accurately. Especially in the form of RFID tags in which employees are forced against their will to wear so they can be monitored. For monitoring equipment, RFID tags are a reasonable solution, but tracking humans is very situation dependent. In most instances tracking humans is a violation of privacy, but there are situations where it may be reasonable. From the advisor’s experience working with Mass General Hospital, they proposed exactly this idea of RFID tags. Tracking employees in this case might be okay due to the high cost of health care. The goal of the hospital is to monitor how much attention each patient is receiving to improve the delivery of healthcare.

The goal of Atlas is to create a very simplistic, non-invasive universal software application. The point of navigation should be getting somebody from point A to point B as quickly as possible, and this can be done without a large capital and time investment. A user interacting with a navigation system is likely going to know where they are, or at least can look around and see what they are close to. There is likely not a need to auto-detect their position. In the case of a stationary kiosk or time table, this step is not even necessary because the start location can be set automatically. One argument why auto-detection is necessary could be that people are used to it; as mentioned in the Introduction, over 150 million users use pathfinding software monthly and are used to GPS figuring out where they are and showing them where to go. However, in most indoor scenarios, the navigation tool is stationary or a nearby location is easy to identify making this concept a feature and not a necessity.

A web-based solution that does not require beacons of any kind appears to be a missing component in the indoor navigation market. This approach could still offer many of the same analytic components that hardware solutions seem to boast. Tracking common start points, destinations, time of day, facilities, etc is similar, however, it is difficult to know if the person actually traveled on the route. Perhaps a method of repeated path searches by an individual, or a confirmation message at their destination could remedy this drawback. This approach does not require an iPhone/Android application to be built for each company. One web-hosted application can be created per campus and simply added to the company website, or hosted anywhere that is convenient. It would be accessible on mobile devices, desktop computers, information kiosks, anything that has access to the Internet. While this system will not actively show a user's position in real time, it can provide verbal directions which GPS users are traditionally used to.  Considering the small scale of many of these applications, verbal and text directions are more than sufficient to help users. In addition, having an intuitive map with a clear route should provide an experience where the user does not need to rely on directions.

## 3. Methodology

Methodologies are used as guidelines for developing a project from start to finish. In software, these guidelines map out the phases of the project and ensure each step is accounted for. Currently, there are two main types of software development life cycle methodologies: the standard software development life cycle (SDLC) and Agile methodologies. This section will analyze the difference between these two categories, and propose a choice for this project based on the advantages and disadvantages listed for each one.

### 3.1 Traditional Software Development Life Cycle

The most common and well-documented methodologies that are used to develop systems is the Software Development Life Cycle. Traditional SDLC, aligned with six sigma DMAIC steps, separates development processes into six phases: Requirement Analysis, System Design, Implementation, System Testing, System Deployment, and System Maintenance. This system was not initially successful until it evolved into the waterfall system: no return is allowed to a previous phase after it was completed.

By the nature, the output of one phase becomes the input of the next phase. Following this rule, no phases are overlapped, and each phase’s process is unique. Details of each phase are defined below:

**Step 1: Requirements Gathering**
To initiate the project, the first step is planning the goals of the project and the composition of the development team. The team needs to go through a rigorous requirements gathering to make sure the project is completable and they have seriously considered every design decision along the way. Only after fully understanding all of the requirements can the team move on to the analysis step. The customer’s requirements are organized and translated into functional and nonfunctional requirements for the system, both software and hardware (Bruegge, 121).

**Step 2: Analysis**
In this phase, the requirements are transformed into an Analysis model which is a non-technical representation of the system so project stakeholders can understand. Developers interact with the clients to clarify any confusion or concerns from the requirements. Analysis results in a model of the system that “aims to be correct, complete, consistent, and unambiguous” (Bruegge, 173).

**Step 3: System Design**
System design transforms the analysis model into a system design model. Developers define design goals of the project and break the system into smaller subsystems. Each subsystem is assignable to a single development team. In this phase, developers select strategies for

handling software and hardware challenges such as: persistent data management and access control policy. In the design step, the class, state, sequence and system architecture diagrams will be created. These are the technical models of the system and are of the final product. By the end of this step, the entire application will be modeled (Bruegge, 223).

**Step 4: Implementation**
In this step, the team will code the project based on the design from step 3. They will attempt to implement as many features that were planned and this is often the most time consuming portion of the project.

**Step 5: System Testing**
Developers will perform integration testing to ensure it aligns with the project requirements. The team might opt to track the testing process by using traceability tools like ALM, and produce defect reports, testing activity reports, and updated metrics. There are many forms of testing, such as: unit testing, integration testing, and system testing (Bruegge, 435). Unit testing isolates individual components through test stubs and drivers using test cases (Bruegge, 449). Integration testing will test several components together in the same way unit testing is done (Bruegge, 449). System testing focuses on the complete system, both functional and nonfunctional requirements (Bruegge, 450).

**Step 6: Deployment**
Once the product has been thoroughly tested and complies to the requirements, the team will determine their deployment strategy for delivering the product to the end user.

**Step 7: System Maintenance**
Maintenance is a process to resolve the actual problems appeared after the deployment to the client and while users are using the system. Software maintenance involves software improvement and new requirements integration. This is the final step and is ongoing once the product has been delivered.

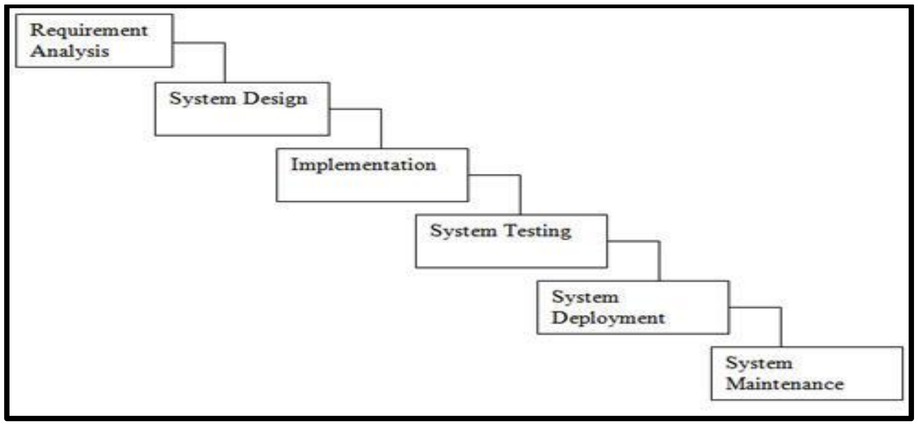


*Figure 5. A diagram depicting the Software Development Life Cycle and all of its steps.*

**Advantages of the Traditional SDLC**
SDLC benefits development teams due to its consistency between projects as each step of the process is explicitly defined. The SDLC model heavily promotes contingency planning because a lot of time is spent at each step to ensure it was done correctly.

**Disadvantages of the Traditional SDLC**
Traditional SDLC is linear by design which means a development team can not go backwards in the traditional SDLC methodology, but this is possible using the Modified Waterfall SDLC methodology which was much more common. With Modified Waterfall, development teams can move backwards, however, it would be extremely time consuming and expensive. This makes product quality difficult to maintain because project requirements will change frequently over the course of a product's lifetime, but the requirements are mostly addressed at the beginning of the project. In addition, stakeholders are not closely involved in the product’s development timeline so misinterpretation of requirements is common (Software Testing Help, 2017).

## 3.2 Agile

The Agile methodology was created to address the shortcomings of the traditional software development life cycle. The one-way development system and minimal stakeholder development caused issues with product quality, such as inconsistent requirements with the stakeholders and changing requirements. Agile focuses on the idea of flexible design, meaning each step of the SDLC should be done constantly to ensure the direction of the project is always up to date. The requirements evolve over time and require a team to be self-organizing and able to rapidly deliver a product (Agile Manifesto, 2001). Agile is a collection of methodologies, one of which being Scrum (LinkedIn, 2017).

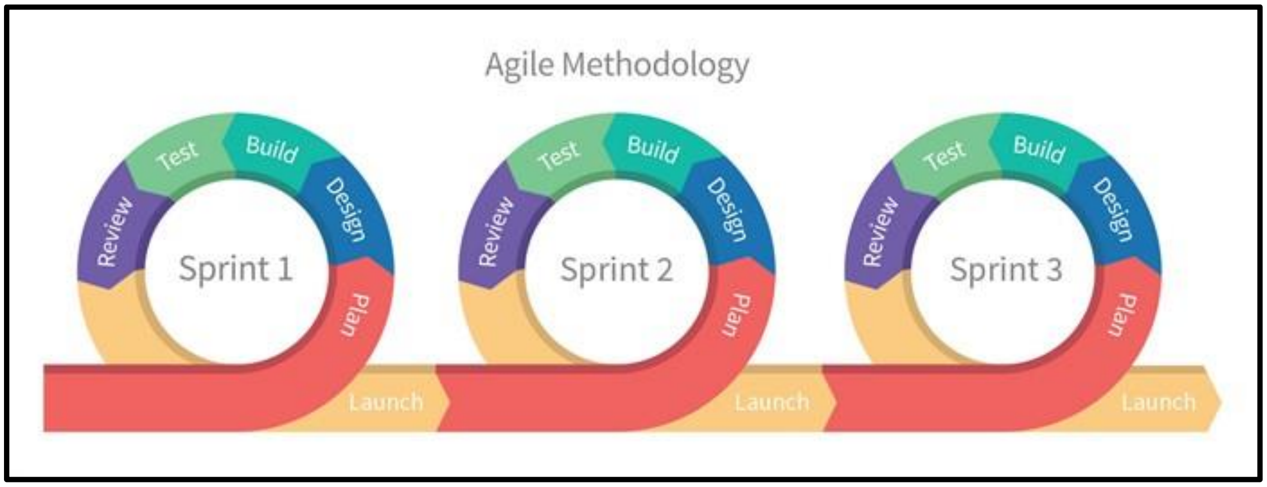


*Figure 6. A diagram depicting the Agile-Scrum Methodology and an example of the sprint process.*

### 3.2.1 Scrum

The distinguishing feature of the scrum framework is the use of sprints. A sprint is a set period of time where the development team effectively goes through most of the development life cycle over again. Scrum places a strong emphasis on project management and aims to rapidly deliver a usable product by the end of each sprint. A scrum team has two unique roles: the Scrum Master and the Product Owner. The rest of the team members will write code and do system design.

**Scrum Master**

The Scrum Master’s job is to act as an overseer of the scrum process and is tasked with making sure things are running smoothly. The scrum master is not the team manager because a proper scrum team should be self-organizing They simply ensure proper scrum practices are taking place and helps address any roadblocks team members bring up during a daily scrum meeting. A scrum master may also organize the sprint planning, review, and retrospective meetings. The daily scrum meeting occurs at the start of every work day and each team member reports their progress individually. Team member will report: their work done since the last meeting, obstacles preventing them from working, and what they intend to accomplish by the end of the day (Bruegge, 613). In the sprint planning meeting, the stakeholders select which items from the product backlog should be considered for the next sprint. The team then decides the tasks necessary to complete the selected items (Bruegge, 612). In the sprint review meeting, which takes place at the end of a sprint, the product owner may intervene and renegotiate the product backlog with the client. The product owner can then choose to re-prioritize tasks and even add new ones (Bruegge, 614). The retrospective meeting is meant for the development team to reflect on what went well and what could be improved for the next sprint.

**Product Owner**

The product owner maintains the product backlog, which is a list of all features (in the form of user stories) the team builds up. A user story is a “single functional requirement formulated by the client that is realized during a sprint” (Bruegge, 471). This person is responsible for consistent communication with any project stakeholders and will update the product backlog accordingly. The product backlog contains an organized list of all the project’s requirements (Bruegge, 612). The product owner will prioritize the product backlog for every sprint and inform the team of any requirement updates from the stakeholders. There is also the sprint backlog contains a subset of the product backlog that the team determines is completable within a single sprint (Bruegge, 612).

**Advantages of Agile**

One major advantage is the high stakeholder involvement. Stakeholders includes the client, management, and development team, so keeping them involved is important to ensure that the product turns out the way they are expecting it to. This means that an Agile team should be able to deliver constant project updates, and due to its flexible nature, be able to adjust their goals as the stakeholder changes the requirements. Requirements can change during the review at the end of each sprint to keep the project on target based on the stakeholder’s needs.

Agile-Scrum is easy to estimate time and monetary costs because of the sprint system. Each iteration will always be a fixed length, and the longer the project goes on, the better the team will be at predicting how long each feature will take to implement. While this only applies to Scrum, each Agile methodology has its own way of providing some extra advantage. For example, the XP methodology requires pair programming which improves code quality and team interaction.

**Disadvantages of Agile**
Agile is still relatively new, and therefore there is a lot to learn about the process. Many people in industry are used to SDLC already, so it can be difficult to justify spending time learning a specific software methodology like Scrum. Following a new software methodology closely can be confusing for new team members, and it is integral that each member understands and cooperates with the system (CPrime, 2013).

### 3.3 Methodology Selection

The Agile methodology will be used for this project, specifically the Scrum framework. SDLC is too linear for a project where there is one developer and would require an immense amount of contingency planning. Due to the short time constraint to develop the software, SDLC would take up the majority of the time planning and leave very little time for programming. This will cause the program to be rushed and have many bugs.

Agile works in short sprints that allow developers to focus on a few features at a time and complete them from start to finish. This also falls perfectly in line with weekly meetings with the project advisor. Agile-Scrum has been used by the author for multiple software engineering projects in the past, so there would not be any time lost in learning the process.

## 4. Software Development Environment

In this section, the software used to create the project will be listed in detail. It was important to compare and contrast a few different options for each component before deciding on what to use. This section will highlight the project management software, integrated development environment, repository, and database selections.

### 4.1 Project Management Software

Project management software is any kind of software that assists in managing a project's contents distinct from programming. Many tools exist for increasing team communication, organizing team members and their assigned tasks, as well as tools for supporting different software methodologies such as scrum. The software examined for this project was JIRA, Trello, and ScrumDo. These three were selected due to their previous use in CS3733, anecdotal experience, and basic online searching. The main criteria for selecting the actual software to use was having an intuitive user interface, personal familiarity with the software, and useful features without paywalls.

### 4.1.1 JIRA Software

JIRA software is one of the many components of the Atlassian Suite of development tools. It was originally designed a simple issue tracker, but has become a full project management tool specifically for Agile projects. JIRA makes it very easy to create user stories, break stories down into tasks, and assign them amongst team members. There is also the ability to create a custom workflow catered to your team's practices and JIRA will enforce this workflow to be followed.

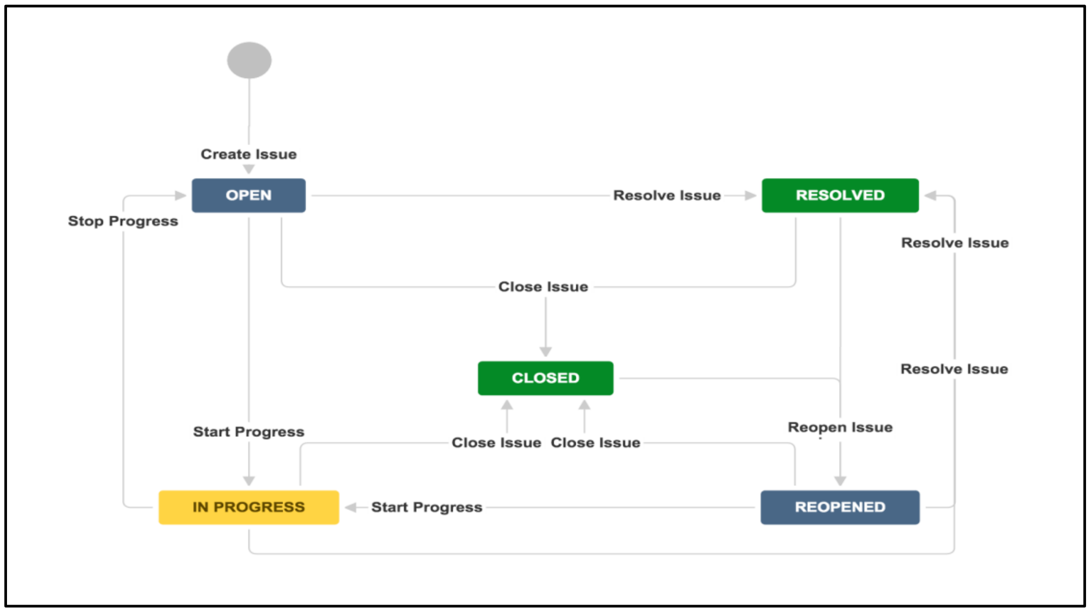
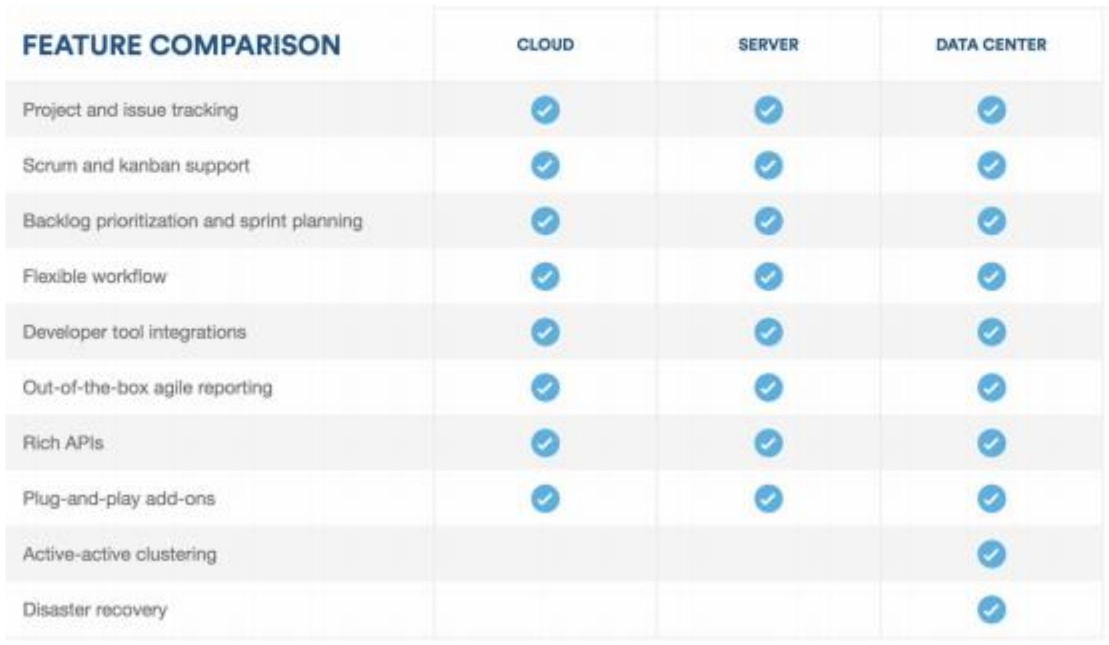


*Figure 7. A sample of a custom workflow made using JIRA.*

There are two main ways to use JIRA software: the self-hosted option and the cloud based version accessible through a web browser. The cloud based version is by far the simplest to use. It comes ready to use right away without any extra tinkering by the owner. There are also many extra plugins created by other developers and with the cloud based version it is as simple as clicking install to use them. Many of these third party plugins are to simply improve user experience over what JIRA already has to offer. However, there are not as many customization options using this approach as plugins are limited to "Cloud-only". It does also require a monthly subscription fee, and this cost scales based on the number of developers on your team.

The self-hosted approach allows a team to have complete control over the project management environment. It is up to the owners how to customize their JIRA experience, but there is the obvious drawback of having to setup and manage your own server to host JIRA. Upkeep costs can be expensive, and there is a large time investment to get the software running. Both approaches offer the same features as shown in figure 8, however self-hosting will only cost a one time fee as opposed to a monthly subscription (Atlassian, 2017).



| FEATURE COMPARISON | CLOUD | SERVER | DATA CENTER |
|---|---|---|---|
| Project and issue tracking | ✓ | ✓ | ✓ |
| Scrum and kanban support | ✓ | ✓ | ✓ |
| Backlog prioritization and sprint planning | ✓ | ✓ | ✓ |
| Flexible workflow | ✓ | ✓ | ✓ |
| Developer tool integrations | ✓ | ✓ | ✓ |
| Out-of-the-box agile reporting | ✓ | ✓ | ✓ |
| Rich APIs | ✓ | ✓ | ✓ |
| Plug-and-play add-ons | ✓ | ✓ | ✓ |
| Active-active clustering | | | ✓ |
| Disaster recovery | | | ✓ |

*Figure 8. Comparing different features between the different hosting options offered for JIRA software.*

### 4.1.2 ScrumDo

ScrumDo is a web-based project management tool for use in Agile Scrum projects. It is unfamiliar software to the author. It offers all of the necessary features for a scrum team, such as user story cards, task management and distribution, and even a task board that team members can move cards between to simulate having one in person. There is also the capability to make a custom board similar to the custom workflow feature in JIRA.

ScrumDo for the most part is straightforward to use and provides enough help if something is unclear. Some of the interfaces are a bit confusing as some menus drop down vertically while others open horizontally. Another drawback to ScrumDo is the loading screen when changing to a different part of the project. The loading screen is noticeably slow considering how often a user would be moving around their workspace. One nice feature unique to ScrumDo is a built-in planning poker game to help assign story points. ScrumDo also supports many of the popular software development tools such as Slack and GitHub (ScrumDo, 2017).

### 4.1.3 Trello

Trello is a simple web-based tool used for team collaboration. Trello was used by software engineering teams in CS3733 so it is very familiar to the author. It is generally used to keep track of which members are working on which tasks, and serves as a place to keep global deadlines, notices, and anything else that would be helpful for the whole team to see. It uses a card based system, similar to how a Scrum user story card would be made, and individual cards can be assigned to team members. There is a lot of customization to better suit your personal development team, and the interface is very intuitive.

It is very easy to categorize different types of work as there are many color coding options, as well as the ability to create tags. Trello would be useful for keeping track of which user stories are active in the current sprint, however, it was not designed to do everything a scrum team would need. The Trello board would quickly become cluttered when trying to maintain a backlog, and within each user story there is not as much detail available compared to JIRA or ScrumDo. Trello would be better suited as more of a personal todo list during the project than for scrum management but it could suffice as a sprint tracking tool in a single person team (Trello, 2017).

### 4.1.4 Management Software Selection

JIRA was decided to be the software management tool for this project. JIRA was used for previous projects and has an understandable interface. JIRA is a great match for scrum projects and provides all of the necessary management tools for user stories and sprint metrics. While ScrumDo may also be a good choice for similar reasons, given the time constraints of the project it was decided to go with a known software by the author over learning how to use something new.

Trello may be used sparingly as more of a to-do list for group sanity. Much of the task list features will be covered by JIRA so Trello may be used for components of the project not strictly related to software development, but perhaps writing parts of the report.

## 4.2 Integrated Development Environment

The development environment section was narrowed down to four options: Atom, Brackets, Webstorm, and Reactide. These were intentionally chosen as a mixture of IDEs and text editors because as a new web developer, it was important to research what kind of environment veteran web developers use and both appeared as recommendations. Developers use text editors with simplistic features such as syntax highlighting and minor autocomplete because many of the features offered by an IDE, such as debugging, can be done through a web browser like Google Chrome. The simplicity of HTML, javascript, and CSS allow for text editors to be sufficient. IDEs were included due to heavy use in past projects and their extensive list of features built-in. See section 4.2.4 on Webstorm for examples. React JS is the library being used for handling the UI elements of this project. This means that React support was one of the most important criteria for selection, but it was not the only criteria. The other criteria examined was ease of use and runtime performance.

### 4.2.1 Brackets

Brackets is a text editor that was made specifically for web development so it seemed like a logical place to start. This editor has a built-in browser preview and updates without having to run a server. It is also easy to use and get running without much time spent in configuration files and has plenty of plugins to choose from. Brackets also has syntax highlighting, selection editing, and auto-complete. The main drawback with this editor is that there is not much for React support. Brackets is very fast and performed well even while doing browser preview updates which is definitely a plus (Brackets, 2017).

### 4.2.2 Atom

Atom is an open-source text editor that was heavily inspired by the popular Sublime Text. This seemed like a great option, as the React community highly supports many of the plugins for development using Atom. The “nuclide” package for Atom was explored. This package effectively turns the simple text editor into an full IDE. It has a debugger, code outline view, syntax highlighting, and a large community supporting React development. Nuclide also provides a version control system and is overall easy to use. There are some settings that can take time to find, but in general they are where you would expect them to be. One major flaw with this system is how slow Atom runs with this package running. Launch times are very long and general program use is noticeably slower which is a definite negative for a text editor (Atom, 2017).

### 4.2.3 Reactide

Reactide is an IDE currently in development that is targeted specifically for React development. There is not a stable release available, but the project is available on GitHub and can be compiled for testing. Reactide is still very early in development and a large number of bugs made it unusable at the time of this project. While it can not be used for this project, it was worth exploring for future projects. Reactide promises to offer a built-in Node server and a browser simulator, eliminating the need to set these things up for every project. Reactide also claims to offer a state-flow visualization, allowing developers to see their project architecture and quickly edit specific components on the fly (Reactide, 2017).

### 4.2.4 Webstorm

Webstorm is an IDE created by Jetbrains, the same creators of IntelliJ. Webstorm offers all of the same features that come standard under Jetbrain IDEs such as version control, dependency management, a debugger tool, refactoring and other convenient features for writing code. In addition to these features, Webstorm also has support for React development with compatibility for .jsx format files. Even more, Webstorm also supports Meteor JS projects - allowing proper file formatting, launchability within the IDE, debugging, and package management. There are many extra features that will go unused or are not needed for this project that make an IDE more complicated to use than a text editor. Webstorm is slower than text editors like Brackets, but it did have faster launch times than Nuclide. It is also worth noting that launching the actual web server was much faster and convenient than doing it through the command line in conjunction with a text editor (JetBrains, 2017).

### 4.2.5 IDE Selection

Webstorm IDE was ultimately selected for this project. Brackets was eliminated early on due to its lack of support with React development. Originally, Atom editor was going to be used with the nuclide package as many communities seemed to recommend it and a lightweight text editor is preferred over launching an IDE. However, because of the hindered performance of Atom with nuclide installed, it really did not offer many features over Webstorm. It simply was no longer a lightweight text editor. In addition, the downside to using a text editor as a beginner is that managing packages and knowing functions, fields, etc. available is made much easier with a full IDE. Webstorm was also chosen over Atom because it has support for many web frameworks including Meteor JS; this makes overall project management much easier. Webstorms overall performance was satisfactory and the only downtimes are when initially launching the application.

## 4.3 Repositories

A repository is an essential component of any successful software project. It provides a storage location for all of the project code and any documentation or external files needed for the project. The most popular choices for version control were chosen for comparison. The choices are between GitHub and Bitbucket as code repository options, both using Git as their VCS. For documentation storage, Google Drive was selected. The rationale for this is explained in the section "Documentation Storage: Google Drive."

### 4.3.1 Version Control

Version control is the process of keeping many versions or configurations of as software project organized. A good version control system allows a user to easily revert back to old versions of the project. If version control is done correctly, it becomes very easy to identify when and where bugs occur in the project and the exact problem files can quickly be addressed. There are two ways version control is generally done, these are: centralized version control (CVCS) and decentralized version control (DVCS).

#### 4.3.1.1 Centralized Version Control System (CVCS)

A centralized version control system means that the code is stored in one centralized location. This means that if a developer wants to make changes, they need direct access to the source files if they want to record any progress. All code changes have to come directly from the centralized repository, it is not shared between users. This key concept is the main difference between CVCS and DVCS. The focus here is synchronization across developers, tracking of files, and constant backup of source files. Subversion is the most popular VCS that follows a centralized methodology.

#### 4.3.1.2 Decentralized Version Control System (DVCS)

The concept here is very simple: easy developer owns their own local copy of the project, and the goal is to share it with the rest of the team. Every change has its own unique ID so it is very easy to checkout different versions. This method of version control has many advantages, the main advantage being that every developer has their copy to play around with and make changes to. Everything can be managed from your local machine, and all of the history is pushed to the repository. This is also another advantage in that you *can* work offline, and if you do interface with the repository it is generally very fast because all of the processing was done on your machine. A decentralized model allows merging and branching very easily, creating code without duplication that is consistent between team members.

There are some drawbacks, of course, the main one being that there really is not a latest version of the application. There will always be the "master" branch which should contain the latest, *reviewed,* version of the application, but this is not necessarily up to date with what developers are working on in different branches. This is especially true when changes go through multiple code revisions which takes long periods of time (AppFusions, 2010).

#### 4.3.1.3 DVCS vs CVCS

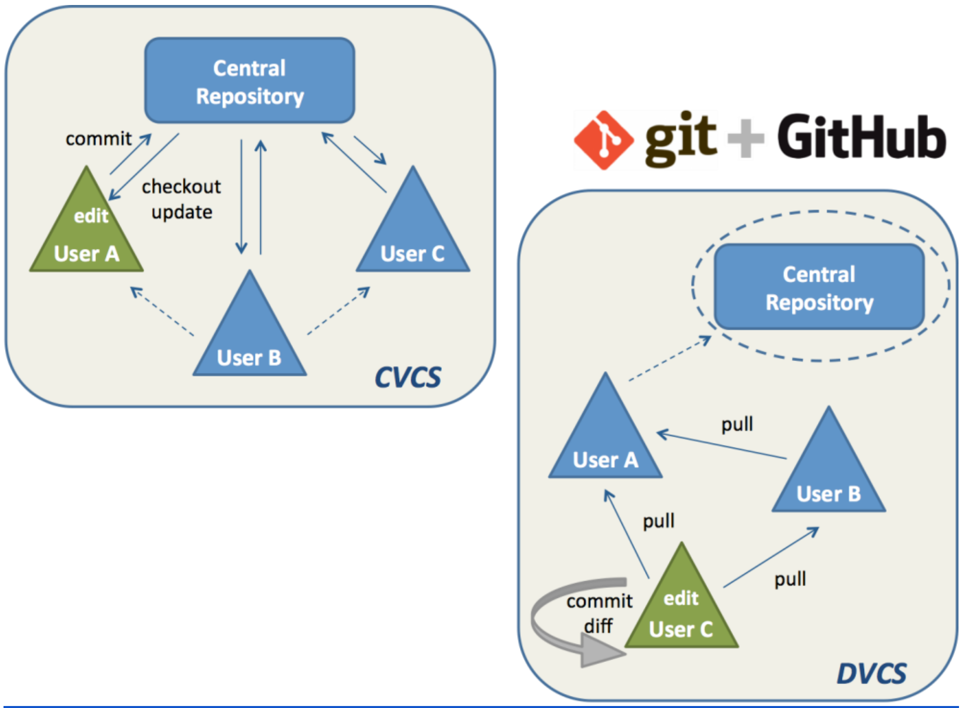


*Figure 9. A visual representation of a Centralized Version Control System (left) versus a Decentralized System (right) (AppFusions, 2010).*

#### 4.3.1.4 GitHub

GitHub is one of the most popular web-based git version control systems available. This service runs on the decentralized model, as seen in figure 9. GitHub is free to use for public repositories and costs $7/month for access to private repositories, or free for student accounts. GitHub's interface is very familiar and easy to follow. This website hosts many open-source projects and is extremely user friendly in handling things like branching, merging, and pull requests (GitHub, 2017).

#### 4.3.1.5 Bitbucket

Bitbucket is a git version control system that is part of the Atlassian Suite. It provides many of the same features as GitHub, however it does allow for private repositories for free. The interface is not as clean as the one on GitHub, but it is still fairly user friendly and all works the same since it is also based off of git (BitBucket, 2017).

### 4.3.2 Version Control Selection

For this project Bitbucket will be used as my software repository. The main reason Bitbucket was chosen over GitHub is simply for the fact that I am able to make the repository free by default. Both services provide the same features and most of the git related work will be done through the git command line. The repository is simply a storage space for the project, and a way to keep move between versions if major bugs arise. Since this project is not open source, it was decided to keep the project private during development.

### 4.3.3 Documentation Storage: Google Drive

Google Drive was selected as the service for storing all of the files for this project. Files include map images, software documentation, and diagrams. Google Drive has a huge default storage capacity of 15GB that will be more than sufficient for the purposes of my project. It is very easy to share files and access them from different computers if needed. On top of this, Google Drive has many apps already built-in such Draw.io. This application is great for creating system diagrams and UML models in a way that is compatible with Google Drive. The choice was made based on the author’s extensive experience using the service and its simple organization.

## 4.4 Databases

A database is needed in this application for storing map data information, such as nodes and edges, as well as references to images. The only prior experience was in using SQL databases, but a NoSQL approach is generally how web development is done, but is not the only way. This narrowed choices down to MySQL, MongoDB, and Firebase. The criteria used to select a database was based upon compatibility with my selected framework, ease of configuration (if necessary), ease of learning commands, and querying performance.

### 4.4.1 MySQL

MySQL databases are some of the most common databases used in industry. They have the benefit of being widely supported and very easy to set up. The command syntax is relatively straightforward, similar to how you would say the statement in English. One advantage is that prior experience exists exclusively using a MySQL database, so there would not be any time spent learning any configuration or syntax. The disadvantage to a MySQL database is the performance on tasks requiring a high read to write ratio. MySQL does have other limitations such as poor fault tolerance and data warehousing support (Mack, 2014).

### 4.4.2 MongoDB

Mongo is a type of NoSQL database that uses documents based on JSON instead of the traditional SQL table entries. Mongo has become very popular among web developers due to its ability to easily reuse data as well as simplified data analysis. Queries are very easy to index since it is effectively the same way JSON is parsed. The executable commands themselves do not have a large learning curve, and the MongoDB documentation is very thorough. There is a lot of support for Mongo in many different programming languages and web frameworks. A potential downside to using Mongo is the security background needed to configure it properly. Out of the box, a Mongo database is not configured to be very secure, and this has historically resulted in problems because of hackers. Mongo also has a set of online courses through MongoDB university to help learn Mongo and become officially certified (MongoDB, 2017).

### 4.4.3 Firebase

Firebase is a realtime database acquired by Google to be used in web and mobile applications. It has become increasingly popular as new features are added. Amongst these features is a very thorough analytics system, client-side user authentication, storage for files, and many built-in features for mobile applications such as notifications. This database boasts very good performance and is actively in development. The Firebase team welcomes open-source projects that use its services. There is a bit of a setup process to get the database going. This requires going through Google's API, getting the developer keys, and then learning all of the new commands available. The Firebase documentation provides an intuitive step-by-step guide to set this up (Firebase, 2017).

### 4.4.4 Database Selection

For this project, the Mongo database was selected. MeteorJS comes pre-built with a Mongo database and has full support for its functions. This saves the effort of configuring a database, and does not require any work with registering API keys and doing the authentication steps for that. Meteor does not provide a full Mongo build, but instead another version called "minimongo." It has most of the standard functions and should be sufficient enough for my purposes. Despite being built into MeteorJS, the option to select another kind of database in the future is still available if, after system design, this database proves to be inadequate. There was a similar pathfinding group in the CS 3733 class who had trouble using a Mongo database because their implementation of a node network was based on a table of node neighbors. Connecting nodes in this manner was difficult in Mongo because generating a network recursively with the document approach is not what Mongo was designed for. However, in hindsight many of these design flaws could be remedied through software design patterns. In the case of the node network implementation, a Composite design pattern could be used.

# 5. Software Requirements

In this section, an outline of the minimal application will be explained. The minimal application refers to the first iteration of the project, which is a barebones application that can simply route between two points. The minimal application should have the the essential components of the system architecture in place. This section will have brainstorming features that may or may not be part of the final iteration, and will not necessarily be in the minimal application.

## 5.1 Primary Audience

This section describes why an indoor navigation tool would be useful and how users might interact with it. The locations listed are considered the target audience, and employees at such locations would be using Atlas to create an indoor pathfinding version of their business.

Due to the need for indoor navigation tools, applicable locations could be any of the following:

- **Airport:** The application could be useful for visitors of an airport by having directions to gates or stores in their own language. Airports are often large and confusing, especially for new travelers so having an airport kiosk for directions would be convenient as well as mobile access. Flight schedules could also be integrated into the navigation tool.

- **Train Station:** Similar to airports, train station navigation would include train schedules and boarding locations. This would best be deployed for mobile devices, but a kiosk displaying the schedules could be useful.
- **Shopping Mall:** A navigation tool for a shopping mall would help users get to and from the parking lot, between stores, and find facilities such as restrooms. This could be deployed on a stationary kiosk, which currently consists of static maps and a huge directory that takes a long time to find the store you are looking for.
- **University Campus:** Mainly targeted at visitors to a university, a mobile application to route the user between buildings could be created down to the detail of finding a specific room in a given building. It could also show current events happening on campus. This has been proven to be a viable application as shown in the CS 3733 Software Engineering class.
- **Company Campus:** For larger companies who span multiple buildings, a navigation tool with an overview of the area encompassing all of the buildings would be useful for visitors and new hires to find their way around.
- **Hospital / Doctor Offices:** Visitors could interact with a mobile application or kiosk to find patient rooms, and know which elevators to take to get to certain departments. A directory listing would be useful to help know where doctors might be during certain hours of the day and provide information on how to contact them. This has proven to be a viable application as shown in the CS 3733 Software Engineering class.
- **Entertainment Events:** Planners for large outdoor events such as fairs, or amusement parks could create an application for helping their users find rides, food stands, and keep track of events happening in the park. This is true for locations such as zoos, aquariums, and places of that nature. For events with live performances, there could be information about the performers.

## 5.2 Initial Feature List

The initial feature list is not an exhaustive list of features for the final product of the application. These are simply brainstorming ideas that are drawn from personal experience using navigation tools, ideas from the target audience section above, and some based off of the survey data detailed in the “Survey Data” section. To be clear, features are considered extra functionality that serve to make the application better and are not core functionality, such as navigation between two points.

Features (in no particular order):

- Multiple Points along route (waypoints)
- Ability to add features specific to your location (modules)
    - Time tables
    - Staff directory
- Landmarks, such as food, bathrooms, etc.
- Ability to send directions to yourself (module)
    - Email

    - Text
- Estimated time and distance
- Heuristic based routes
    - Avoid outdoor travel
    - Avoid stairs
    - Handicap accessible
    - Areas of least traffic
- Map panning
- Map Zooming
- Database security
- Multiple supported data models
    - Ie Node Neighbors vs Node Edges
- Contact Information (module)
- Current events (module)
- Advanced search capabilities
- Multiple Routes
- Analytics system (module)

## 5.3 Survey Data

The survey data was taken from past CS 3733 courses doing a similar project. The projects were to create a pathfinding application that could be used for the WPI campus or Brigham and Women's Hospital. The surveying methodology was very simplistic given the nature of the course, so the information does not contain any form of statistical analysis, or had undergone standard surveying practices. With that being said, the feedback from the survey data will be used as a general guideline for features users might be interested in, but does not have a heavy impact on the overall design choices for this project. Since this is a single person MQP, surveying was decided to be outside the scope of this project by the project advisor and so data from the past two school years will be used in its place.

From the survey data, it is apparent that people get lost very frequently and usually have to ask for directions (Jackalopes, 2017). This means that traveling indoors is still a puzzle, and no leading technology is commonly used to help people. It is not surprising that the frequency people get lost and the frequency they ask for directions are directly proportional, based on the trend line in Figure 10. This is visually clear because most of the data is aggregated in the upper right corner.

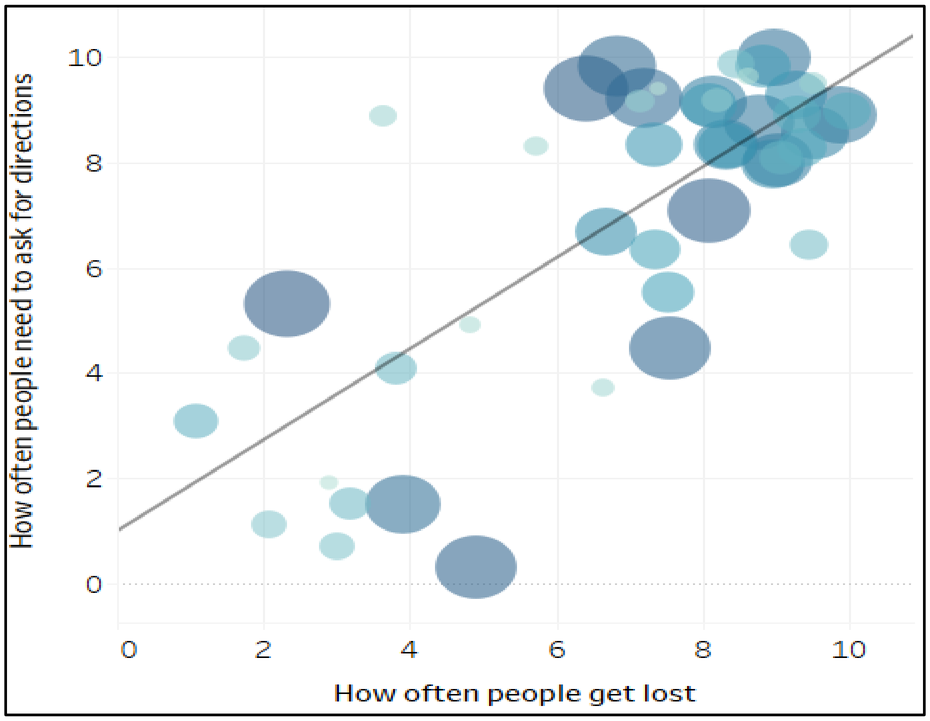


*Figure 10. A visual representation of commonly occurring survey responses. Frequency level scale from 1-10, the higher scale the higher frequency.*

The surveys attempted to determine which features end users generally want in an indoor pathfinding application. According to the data provided in the five projects' reports, with an average sample size of seventy people, these features (Figure 11) were the most requested:

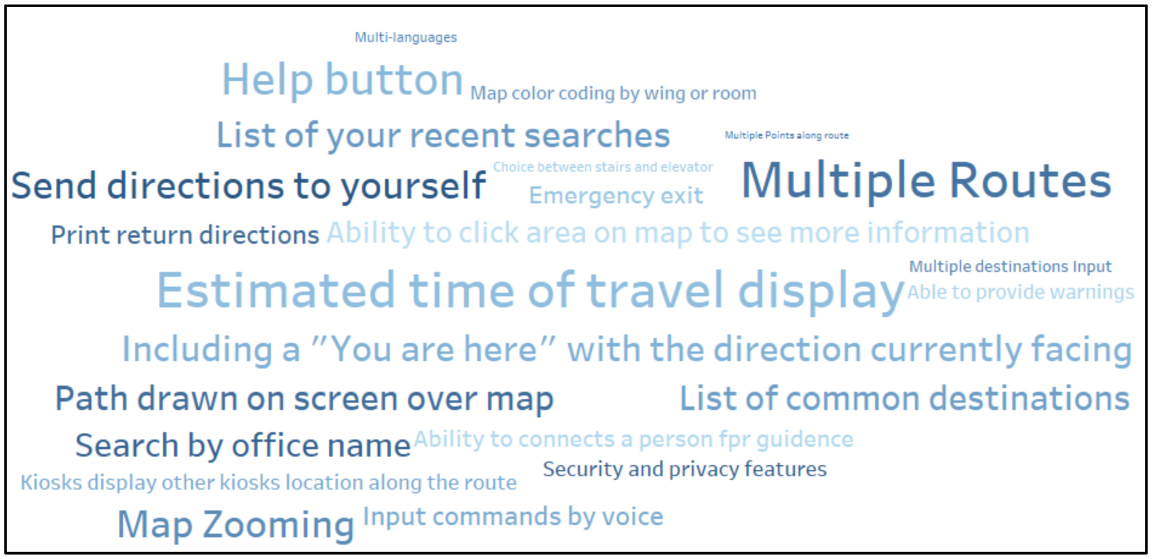


*Figure 11. A visual representation of commonly occurring survey responses. The larger the font is, the more frequent the response came up.*

These features are reorganized by the size of recorded numbers. In other words, the necessity of a feature can be measured by how frequently it appeared in the survey data.

## 5.4 Functional Requirements

For the purposes of this project, a user is a person who would interact with the final product. This is somebody trying to navigate around a location. An administrator (or admin) is a person who works at a business using this software to build an indoor navigation application. Admin is a broad term that could encompass, but is not limited to someone representing any of the locations listed in the Primary Audience section of the report. Ideally this could be broken down into specific locations such as Airport or Shopping Mall, but given the resource limitations of this project and lengthy list of users, abstracting into a single administrator role was deemed acceptable.

The format for this section will be an epic story, the user stories that fall under the epic story, a scenario or two, and a textual use case description. At the end of this section is use case diagrams for each actor.

**Epic Story 1**
As a user, I would like to move between two locations as fast as possible.

*User Stories*
As a user, I want to select any point as my start or end point to be as accurate as possible.
As a user, I want to be able to enter the name of my start location so I do not have to find it on the map.
As a user, I want to be able to enter the name of my destination so I do not have to find it on the map.
As a user, I want to select points of interest already present on the map so I can browse or choose a location I already know of.
As a user, I want the option to find a location by many names so I do not have to choose the exact search criteria.
As a user, I want to get to my destination as fast as possible so I can use my time effectively.
As a user, I want to see a visible path for my whole trip on the map so I can visualize where I need to go.
As a user, I want to only see the part of the route for the floor I am on so I do not get confused where I am on the trip.
As a user, I want to be able to pan the map so I can explore the area.
As a user, I want to be able to zoom on the map so I can see the area in more or less detail.
As a user, I want to know how much traffic there might be on this route so I can plan around it.
As a user, I want to be able to rotate the map so I can change my viewing angle.

*Scenarios*

1. David is new to the San Jose area and wants to go one of the nearby indoor shopping malls to spend the afternoon. He is unfamiliar with the area and is overwhelmed by the scale of the mall compared to the ones back at home. Since he is new to California, he

knows he wants to buy a new pair of sandals for everyday use. He arrives at the shopping mall and approaches a kiosk depicting a map of the multi-level mall. David taps on the search bar and enters “Sandals.” He is greeted with a list of locations that are tagged with sandals, and is able to select one of the stores. Upon selecting a store, he selects navigate and is shown a path from his current (static) position all the way to the next floor where the sandal store is. David is fairly tech savvy and chooses to send the text directions to his phone and takes a mental image of the path of the screen to go to his destination.

2. Mike approaches the same kiosk and sees that the navigation kiosk is also accessible from his smart phone. He goes to the mall website and launches the navigation application from the navbar. Having previous navigation experience, Mike searches for the kiosk he is standing at and sets it at his starting location. Mike wants some lunch before shopping, so he searches “food” in the destination box and is given a list of locations with the food tag. He selects one of the options and uses the directions to find his way.

*Use Case*

1. **Name:** Select Point
2. **Participating Actor:** End User
3. **Entry Condition:**
    a. Navigation application open on any device
    b. Starting and ending points known ahead of time but not selected
4. **Exit Condition:**
    a. Map of the initial floor shown with a route drawn on just that floor.
5. **Flow of Events:**
    1. User selects a starting location by clicking on the map via use case Starting Point
    2. User selects an ending location through searching via use case Ending Point
    3. User selects “Route” and a path is drawn on the screen showing the route.

    **Alternate Flow of Events:**

    [invalidPoint]

    1. If the user selects a point too far away from other locations, they will be prompted to select a new point.

    [clear]

    1. Point selection is cleared and route cannot be made until both selected
6. **Special Requirements:** None

*Use Case*

1. **Name:** Starting Point
2. **Participating Actor:** End User
3. **Entry Condition:**
    a. Navigation application open on any device
4. **Exit Condition:**
    a. User is still at map screen with a starting location chosen
5. **Flow of Events:**

    a. User taps a location, or searches a location in the search bar
    b. Selected location is set into the starting location text field
        i. NOTE: A stationary device will have this filled in by default.
6. **Special Requirements:** None

*Use Case*

1. **Name:** Ending Point
2. **Participating Actor:** End User
3. **Entry Condition:**
    a. Navigation application open on any device
4. **Exit Condition:**
    a. User is still at map screen with an ending location chosen
5. **Flow of events:**
    a. User taps a location, or searches a location in the search bar
    b. Selected location is set into the ending location text field
6. **Special Requirements:** None

## Epic Story 2

As an administrator, I want to select features relevant to my application so I only include features relevant to my implementation.

### *User Stories*

As an admin, I want to add a directory listing so I can keep track of the employees who work at my business.
As an admin, I want to add a contact listing page so users are able to easily contact the business' customer support.
As an admin, I want to add an advertisement section so I can display ads relevant to users.
As an admin, I want to add an RSS feed so I can show current events related to my business.
As an admin, I want to add the ability to email or SMS navigation so my users can get the text directions sent to their mobile device.
As an admin, I want to add time tables so I can display scheduling information relevant to my business (ie airports, train stations, etc).
As an admin, I want to include an analytic system so I can gather non-invasive data about my users.

### *Scenarios*

1. Todd is an event coordinator for a music festival happening in downtown Boston. He wants to build a navigation tool for people attending the festival. He goes through building a layout of the festival with the map editor and wants some extra features to improve the user experience. Todd goes into the feature toolkit and adds an RSS feed to display current events happening at the festival. This is pulled from an existing RSS feed his company hosts. He also selects a contact information panel so users can see emergency contact information and other points of contact. Lastly, he adds a directory listing to provide information about the musical groups who will be performing at the

festival. He plans to use the navigation capabilities to guide users to facilities such as food or restrooms.

*Use Case*

1. **Name:** Select Directory Listing
2. **Participating Actors:** System Administrator
3. **Entry Condition:** Package Manager open
4. **Exit Condition:** None
5. **Flow of Events:**
    a. Admin scrolls through the list of packages and finds one they like
    b. Admin can add a package via the AddDirectory use case
    c. Admin can uninstall a package via the Uninstall use case
6. **Special Requirements:** None

*Use Case*

1. **Name:** AddDirectory
2. **Participating Actors:** System Administrator
3. **Entry Condition:**
    a. Admin has already found the package they wish to install

4. **Exit Condition:**
    a. A new feature is added to the program through the Package Manager
5. **Flow of Events:**
    a. Admin selects install and the new package is added
6. **Special Requirements**: None

*Use Case*

1. **Name:** Uninstall
2. **Participating Actors:** System Administrator
3. **Entry Condition:**
    a. Admin has already found the package they wish to uninstall
    b. Desired package was previously installed
4. **Exit Condition:**
    a. Desired package is now uninstalled
5. **Flow of Events:**
    a. User deselects the package to remove it from the application instead of adding it.
6. **Special Requirements:** None

### Epic Story 3

As a user, I want step by step directions available so I can be guided along my route.

*User Stories*

As a user, I want to have a rough distance traveled estimate so I know how far I am going total.
As a user, I want to have a rough time estimate so I know how long the trip should take.

As a user, I want to know in more detail when to make turns so I do not miss a step in navigation.
As a user, I want to send these directions to my mobile device so I can take the directions on the go.

*Scenarios*

1. Jessica is at a large airport and needs to go from her parking garage to a specific gate. She opens the navigation tool from her phone and searches for the garage as a start location. She searches for the gate as the destination and pulls up the text directions to get there. The directions tell her when to make turns and show approximately how long it will take to walk there which is important to gauge how much time she has to check-in before the flight leaves.

*Use Case*

1. **Name:** SendDirections
2. **Participating Actors:** End User
3. **Entry Conditions:**
    a. User has already found a route and is looking at the directions
4. **Exit Condition:** None
5. **Flow of Events:**
    a. User selects “Send Directions”
    b. User enters contact information and the directions are sent
6. **Special Requirements:** None

*Use Case*

**1. Name:** SMS
**2. Participating Actors:** End User
**3. Entry Conditions:** None
**4. Exit Condition:** None
**5. Flow of Events:**
    a. Include SendDirections use case
    b. User enters an SMS number into the address box which causes the application to format and send directions via SMS.
**6. Special Requirements:** None

*Use Case*

**1. Name:** Email
**2. Participating Actors:** End User
**3. Entry Conditions:** None
**4. Exit Condition:** None
**5. Flow of Events:**
    a. Include SendDirections use case
    b. User enters an email address into the address box which causes the application to format and send directions through the mail server.

6. **Special Requirements:** None

## Epic Story 4

As a user/admin, I want to have access to the application on multiple devices so I only have to develop one application and not many applications that are OS dependent.

### *User Stories*

As a user, I want to have access to the application on my mobile device so I can access it from anywhere within the facility.
As a user, I want to have access to the application on my tablet so I can access it from anywhere within the facility.
As a user, I want to have access to the application on my laptop or personal computer so I can access it remotely to plan ahead or while within the facility.
As an admin, I want to have access on a stationary kiosk so my users do not have to use their own device.
As an admin, I want the application to run seamlessly and consistently across platforms so the user experience is the same no matter what outlet a user accesses it from.

### *Scenarios*

1. Jane is planning to go visit her friend at a nearby university which has a fairly large campus. While at home, she visits the university website and launches an interactive map of the campus. She locates the dorm her friend lives in and takes the time to search for other nearby points of interest. Jane leaves home and arrives at one of the university parking lots. She pulls up the same application on her phone and enters the locations she found while browsing at home to get from the parking lot to her friends dorm.
2. Albert is a visitor to the same university and used another navigation tool to get to the campus center. He has a meeting with a professor on the second floor of one of the buildings on campus. Albert goes to a kiosk located in the campus center and uses the search features to find the professor's office by searching their name. A route showing Albert how to get from the kiosk to the office is shown, and Albert can step through the directions to get to the office. He decides to send a quick email to himself with the directions and includes pictures of the route so he can figure out where to go from his phone.

### *Use Case*

1. **Name:** AppDeployment
2. **Participating Actor:** Business Owner
3. **Entry Condition:**
    a. Owner wants to create an indoor navigation tool to run on multiple devices
    b. Owner does not want to create standalone applications for every OS
4. **Exit Condition:** None
5. **Flow of Events:**
    a. Owner uses Atlas to create a navigation tool for their company
    b. Owner hosts their new application through their company website

   c. Clients of the business are able to access the navigation tool from their phone via the Mobile Phone use case, or computer via the Personal Computer use case

6. **Special Requirements:**
   a. Scalability: Application should be able to handle many web requests at once and should cache to reduce heavy computations
   b. Consistency: Application should deliver roughly the same experience across all devices

*Use Case*

1. **Name:** Mobile Phone
2. **Participating Actor:** End User
3. **Entry Condition:**
   a. Business owner has deployed Atlas on a web server
4. **Exit Condition:** None
5. **Flow of Events:**
   a. User is able to access Atlas from their mobile device through the hosting companies website via a web browser
6. **Special Requirements:** None

*Use Case*

1. **Name:** Personal Computer
2. **Participating Actor:** End User
3. **Entry Condition:**
   a. Business owner has deployed Atlas on a web server
4. **Exit Condition:** None
5. **Flow of Events:**
   a. User is able to access Atlas from their personal computer through the hosting companies website via a web browser
6. **Special Requirements:** None

### Epic Story 5

As an admin, I want to monitor traffic in my business so I know which areas are most frequented.

*User Stories*

As an admin, I want to keep track of common starting locations so I know which areas users generally need help navigating from.
As an admin, I want to keep track of common ending locations so I know which areas users are going to the most.
As an admin, I want to keep track of which nodes are routed through the most so I can generate a heatmap of my business.
As an admin, I want to know what time of day specific nodes are routed through the most so I can give general traffic information.

As an admin, I want to know the average walking distance so I can better plan out the layout of my business.
As an admin, I want to know the average walking time so I can better plan out the layout of my business.

### *Scenarios*

1. James is building a navigation application for his shopping mall. In the application toolkit, he selects that analytics module to monitor data about usage. James finishes his application and waits for a week to gather data from his users. He notes the most common starting and ending locations users take to see which stores or restaurants are the most popular and generate the most people coming to his mall. Using this information, he is able to better plan the layout of the mall and introduce more stores of the same products as the most popular.
2. Katie owns a fairly large set of buildings within an industrial park and created a navigation tool with the analytics module to gather metrics about user travel time and distance. She sees that the average employee is spending most of their time walking between buildings or traveling fairly long distances to get to commonly accessed areas such as laboratories. With this information, Katie can optimize the layout of the buildings (and meetings rooms / laboratories inside of them) to reduce company time spent traveling and increase time spent working.

### *Use Case*

1. **Name:** CreateHeatMap
2. **Participating Actor:** System Administrator (SA)
3. **Entry Condition:**
    a. Analytics module has been installed for this application
    b. Some initial data has been gathered from finding routes
4. **Exit Condition:** None
5. **Flow of Events:**
    a. SA goes into the analytics module via the package manager
    b. SA selects “Generate Heat Map” from the list of available actions
    c. SA selects which maps they want to create a heat map for
    d. SA is shown a gallery of heat maps and is given metrics about how much data is represented on each map
        i. SA has the option to save heat maps

    **Alternate Flow of Events**

    [Insufficient Data]

    1. User has not generated enough data to create a heat map OR
    2. User has selected a map that does not have any data
6. **Special Requirements:** None

## Epic Story 6

As an admin, I want the ability to build a graph of nodes for any floor plan image.

*User Stories*

As an admin, I want to load a new image of map so I can create a graph for a certain floor.
As an admin, I want to be able to connect nodes within a map so I can create paths.
As an admin, I want to be able to select if a node is accessible or not so I can block off parts of my business I do not want users going through.
As an admin, I want to connect nodes across multiple maps so I can mark transition points between floors or buildings.
As an admin, I want to keep track of the scale of the map so distance calculations are accurate.
As an admin, I want to calculate the distance between two points so I can give accurate distance estimates to the users.
As an admin, I want to place a point on the map so it can be traveled to.
As an admin, I want to be able to edit a point so I can define its attributes easily.
As an admin, I want to be able to remove nodes so I can get rid of nodes that no longer exist or fix errors.
As an admin, I want to be able to remove edges so I can remove paths that no longer exist or fix errors.
As an admin, I want to be able to save my changes so I can continue working later on the same map.
As an admin, I want to be able to load maps so I can work from a template or edit an existing map.
As an admin, I want to be able to tag locations so they are easily searchable.
As an admin, I want to be able to select and edit multiple points at once so I can increase my map building efficiency.
As an admin, I want to be able to equally space a group of points so I do not have to worry about precise point placement.
As an admin, I want to be able to linearize points to create straighter paths so my final path is clean and does not have many jagged edges.
As an admin, I want to able to move nodes by dragging so I can increase my map building efficiency.
As an admin, I want the application to automatically detect if I have created an invalid path so I do not accidentally create an unusable graph.

*Scenarios*

1. Kyle works for a local hospital is as tasked with creating a navigation tool to help visitors find rooms easily. He is given a set of map images to use and loads them into the map editor. Kyle quickly creates a graph of nodes that fit the first floor image by clicking on the editor. He is able to edit attributes of a node by clicking on them, and can move things around easily by dragging or selecting multiple nodes. Kyle found that many of the paths in the hospital are very linear, and found the linearize feature to be very helpful in creating a clean path. He then saves the changes to the database and proceeds to continue the process for the next loaded image.

*Use Case*

1. **Name:** Edit Node
2. **Participating Actor:** System Administrator (SA)
3. **Entry Condition:**
    a. Map is already populated with at least one node
4. **Exit Condition:** None
5. **Flow of Events:**
    a. SA selects a point that is already placed on the map
    b. SA is presented with a small tooltip-like box with node attributes
    c. SA changes the type of node to "Store"
        i. From here, SA can click away and the change will be saved via the Save Changes use case

    **Alternate Flow of Events**

    [InvalidAttribute]

    1. SA enters a field that does not exist or is malformed and is given a warning to fix it
6. **Special Requirements:**
    a. Consistency: Changes should be saved automatically when finished and propagate to all versions of the application

*Use Case*

1. **Name:** Save Changes
2. **Participating Actor:** System Administrator
3. **Entry Condition:**
    a. Changes have been made to a node via the Edit Node use case
    b. User has clicked away from the node window
4. **Exit Condition: None**
5. **Flow of Events:**
    a. Application makes an update call to the database with all of the editable information
    b. Database overwrites old information with changes made by the user
6. **Special Requirements:** None

## Epic Story 7

As an admin, I want proper database security so my data is only accessible by authorized users.

### *User Stories*

As an admin, I want user authentication so only known users can view the database.
As an admin, I want all account passwords to be encrypted so passwords can not be compromised.
As an admin, I want to be able to configure my database settings so I can have more control over my security settings.

As an admin, I want to be able to view log files so I can monitor database activity.

*Scenarios*

1. Lara is an administrator for a university who has created a navigation application for the university campus. She is experienced using Mongo databases and wants to fine tune the configuration settings by herself to adhere to the university standards. She is able to go into the database manager using a protected admin account, select configuration file, and can make changes to the database settings. She saves her changes and has to verify her credentials again before rebooting the system.
2. Kevin is a data analytics expert in for a business and wants to see how much traffic the database has been seeing. Using a verified admin account, he goes into the database manager and selects the option to view the log files. From here, he can see specific log information as defined by the configuration file, as well as other useful logged information because of the analytics module that his company installs.

*Use Case*

1. **Name:** EditConfig
2. **Participating Actors:** System Administrator (SA)
3. **Entry Condition:**
    a. Database Manager module has been installed
    b. User is authenticated as an SA
4. **Exit Condition:** None
5. **Flow of Events:**
    a. SA accesses the database manager via the package manager
    b. SA selects the Edit Config option
    c. SA (even if already logged in) is prompted to authenticate themselves
    d. SA edits the config file for MongoDB
    e. SA selects save changes and reboots the server

    **Alternate Flow of Events:**

    [InvalidLogin]

    1. User is kicked out of the application overall to prevent any tampering with the config file

    [Cancel]

    1. SA moves away from the config editing so no changes are saved to the config file (warning message before moving away)
6. **Special Requirements:**
    a. Security: Authentication must be done at every step to ensure the SA is valid. User activity with the config file will be logged.

## 8.4.1 Use Case Diagram for End Users

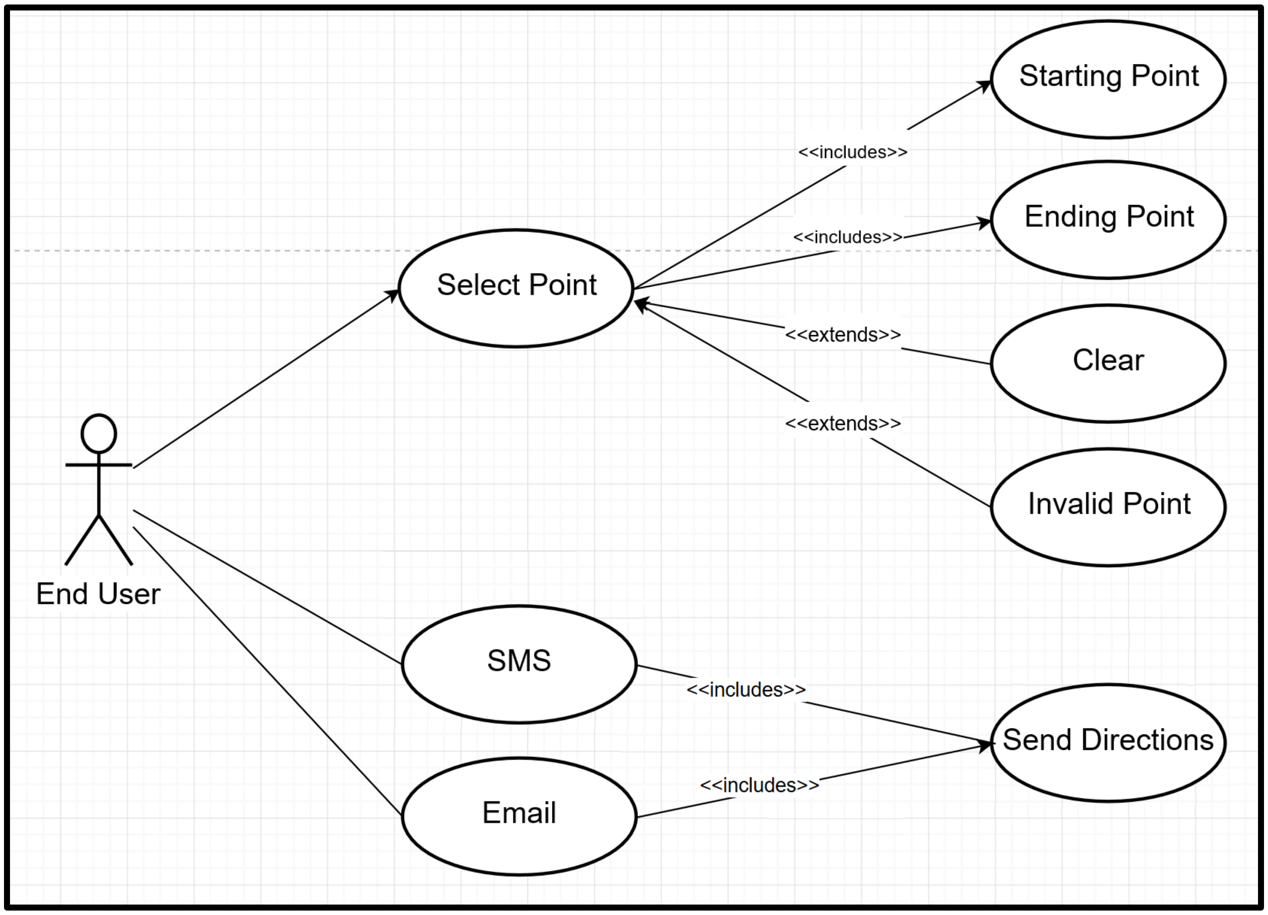


*Figure 12. A use case diagram depicting different use cases of Atlas that an end user might experience.*

## 8.4.2 Use Case Diagram for System Administrators

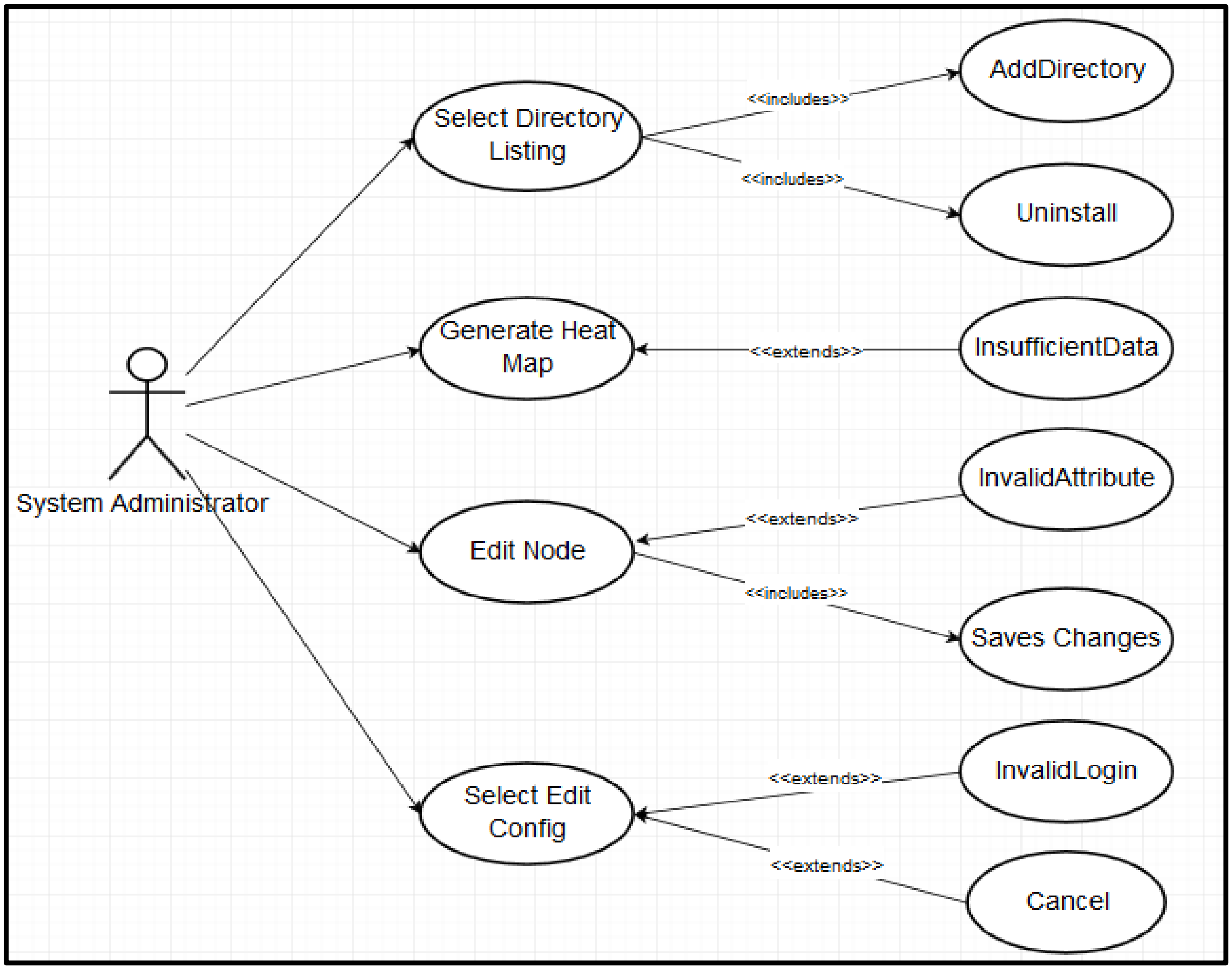


*Figure 13. A use case diagram depicting different use cases of Atlas that a system administrator might experience.*

### 8.4.3 Use Case Diagram for Business Owners

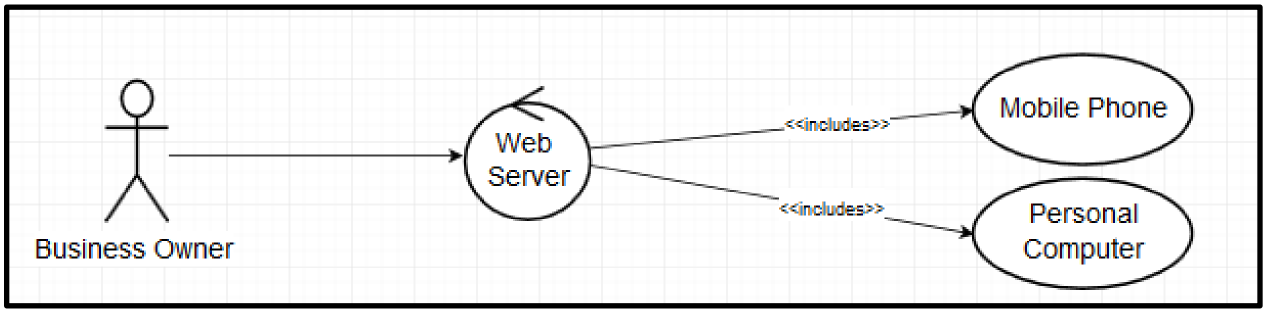


*Figure 14. A use case diagram depicting different use cases of Atlas that a business owner might experience.*

## 5.5 Non-Functional Requirements

**Security**

Security is an important aspect of this project because there needs to be some kind of authentication system to differentiate admin accounts from general users. It would not be wise to give general users full control of the application, and definitely not access to all of the database contents. Users should always have the minimum amount of access based on their relationship to the project. Passwords will be encrypted and have specific criteria to make them harder to crack, such as minimum required length or having at least one symbol.

**Consistency**

Consistency is very important because this application can be run on many devices. This means that any changes made in the background by administration should propagate to all devices at once. This also means that the user experience across devices should be the same, no matter the screen size. A user who transitions from a computer to a mobile phone should easily be able to figure out how to use the application as the core experience should be the same.

**Scalability**

The entire design of this project is with scalability in mind. The system architecture is carefully planned to be extremely modular. This means that it is very easy to add new features to the toolkit, and also work on specific features without changing the core pathfinding part of the application. Future developers on this project will be able to continue working where this iteration leaves off without having to learn the entire code base.

**Usability**

The end goal of someone using this application is to create their own navigation experience. Usability by the administrators building the navigation should be fairly straightforward and not too cumbersome with all of the features. In addition to this, usability by the end user (ie a visitor to an airport) should be extremely simple and somewhat familiar to existing GPS applications. This experience should be defined naturally by the map editing tool process.

**Performance**
Performance is always important to consider, especially when working with graphs. Depending on the size of the campus, the navigation application could see a high volume of users attempting to do costly pathfinding calculations. This process needs to be dealt with efficiently to avoid the system being bogged down by too many users. Caching common routes may be a helpful solution to reduce pathfinding calculations, leaving the only work to be done as the time estimate.

# 6. Design

This section focuses on modeling the program. It is very important to note that this design section is purely for the minimal application and not for the end product. The minimal application should be able to find the shortest path within a single map. This requires map editor portion to have basic functionality, such as adding points and edges to a map image. In addition, the package manager system should be in place and may contain a few preliminary features.

## 6.1 Database Schema

For the minimal application, the database is fairly simple. There only needs to be documents that record nodes, maps, and edges. Package state information will be added at a later iteration because the minimal application will only setup the framework for adding packages and not fully implement them. Having single documents for nodes and edges will be part of the initial design, as Mongo allows for up to 16MB of space per document. This threshold is difficult to reach even with thousands of nodes, and reading back node data using filters is still sufficiently fast.

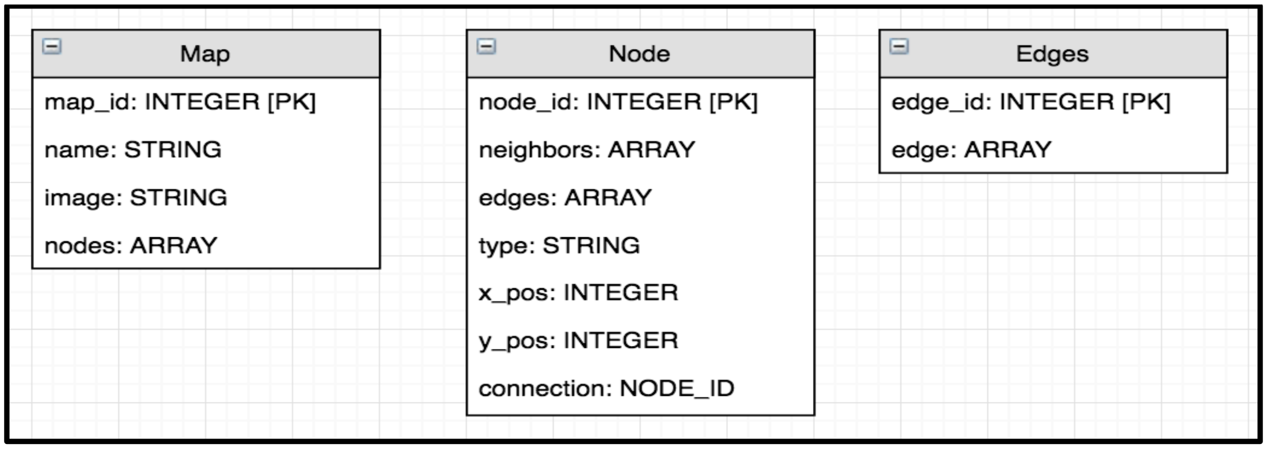


*Figure 15. Simplistic database schema featuring documents for nodes, maps, and edges. The minimal application requires a very basic setup and the fields listed are not an exhaustive list. In its current state, there is no advantage using neighbors over edges but this will be addressed in a later iteration through the Observer software design pattern.*

## 6.2 User Interface Mockups

The mockups focus on the map editor tool and the package manager. The minimal application will start into the map editor by default.

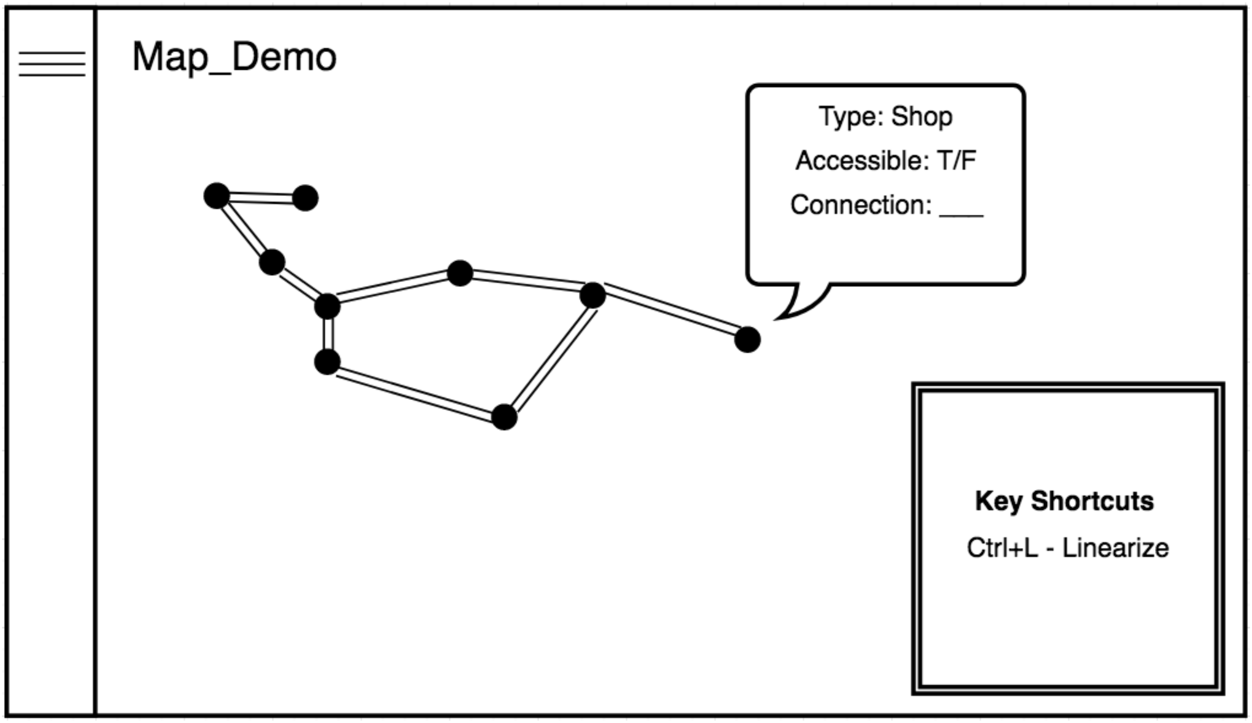


*Figure 16. Main interface for the map editor tool. The image shows an example graph of nodes and their edges. The tooltip represents the user having clicked on the far-right most node to edit its attributes. In the bottom left is a transparent window showing keyboard shortcuts; this will only appear during inactivity or if toggled via a shortcut. Nodes can be placed by clicking on the background (which would have a map image loaded) and edges are created by dragging between two nodes. In the top left, the name of the map (specified by the user) is displayed.*

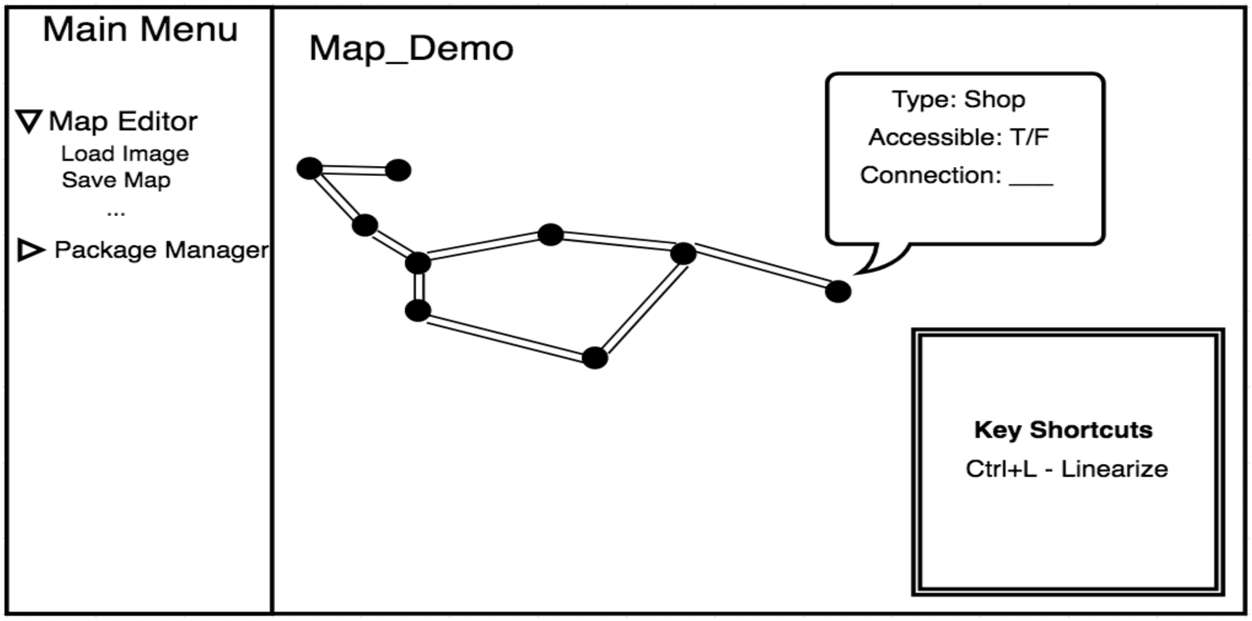


*Figure 17. The same map editor screen as Figure 16, however now the main menu button has been pressed and the menu is open. The map editor portion has been resized so nothing is hidden behind the menu. In the menu, there is an accordion style list of options available. Upon opening the "Map Editor" section, all of the major commands are shown such as loading a new image to create a new map, or saving the current map.*

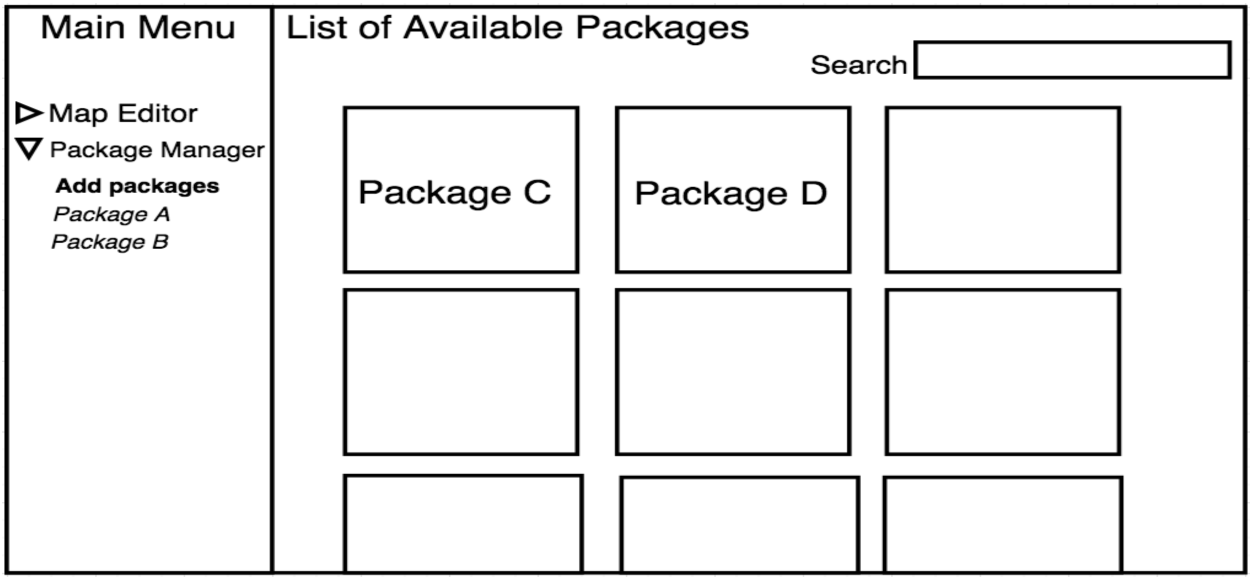


*Figure 18. An example of the package manager with the "add packages" option selected. The main window displays a list of all packages available to add. There is a search bar where the user can enter keywords to find relevant packages. In the main menu side bar, packages A and B represent those who have already been installed. Selecting one would change the main window to display information specific to that package.*

## 6.3 System Architecture Model

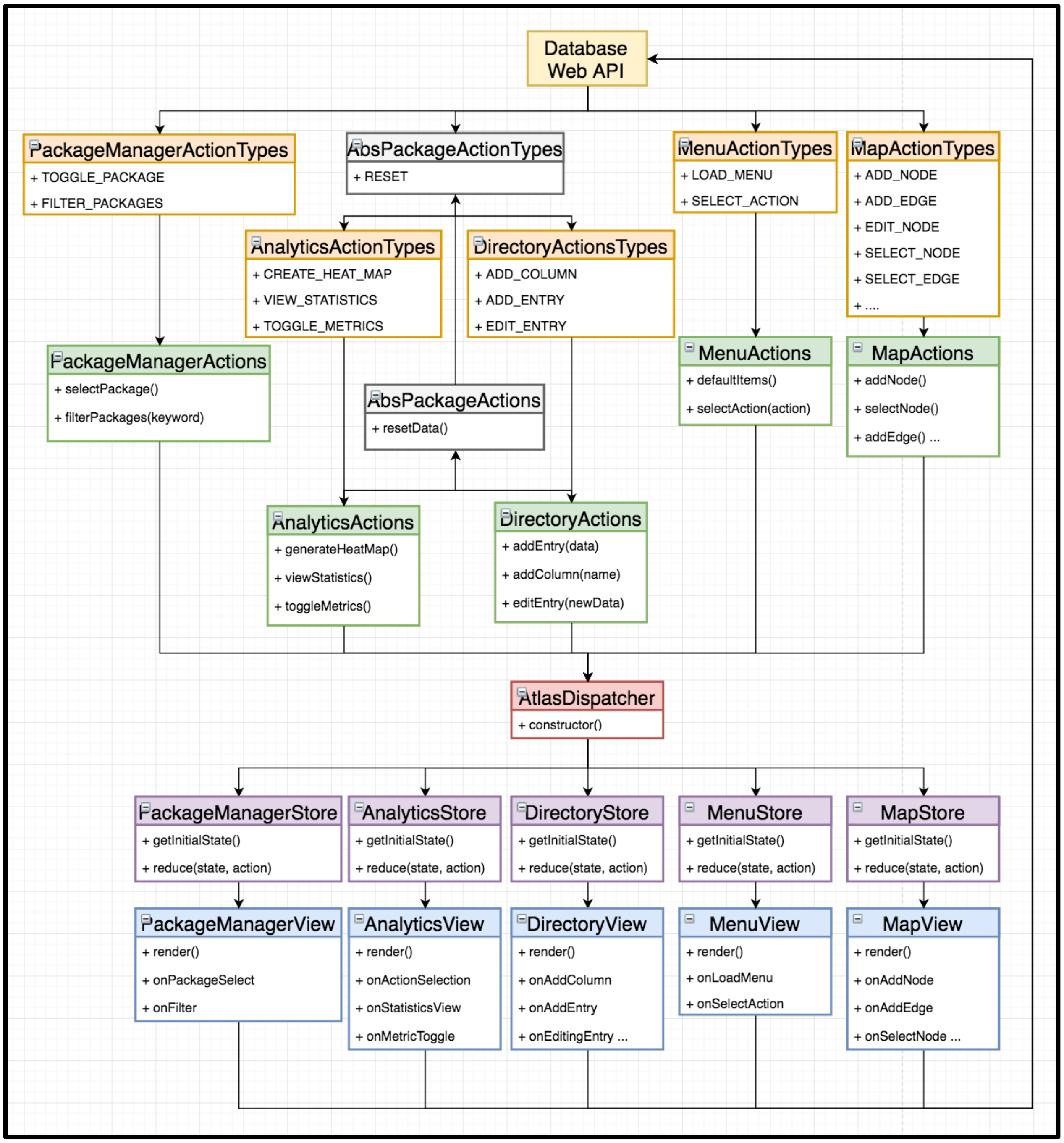


*Figure 19. An overview of the system architecture for the minimal application. This model also serves as a data flow diagram as it follows the flux architecture created by Facebook. The flow reads from top to bottom and is unidirectional. This model should serve as a reliable basis for adding future features because it isolates each piece of the application into its own module. The functions and actions are not an exhaustive list, but they serve as a general overview of what the component is used for. The number of stores may also change during the implementation.*

Actions capture the way anything might interact with the application, whereas action types are essentially events that could occur during the execution of the program that are tied to specific actions handled by the system. The dispatcher receives actions and dispatches them to the relevant stores that are registered with the dispatcher, and there should only ever be a single dispatcher. Each store will receive all actions, but a store is only responsible for a defined subset of actions in the system. This is conveyed in Figure 19 by the AnalyticsStore only handling actions related to the Analytics actions, through the dispatcher, to the AnalyticsStore. A store simply holds the data of an application. Data is only changed as a response to an action being dispatched, and upon doing so, a change event will occur from the store (Flux).

## 6.4 System Overview

The system overview will be used in place of a traditional class diagram. A class diagram would mean displaying every single React component and listing all of its prop types. This diagram is very cumbersome and difficult to read, considering the sheer number of unique React components. Instead, this overview diagram shows the layout of the application in its *current state.* Refer to section 6.3 for an architecture of the proposed implementation which more closely resembles an object diagram.

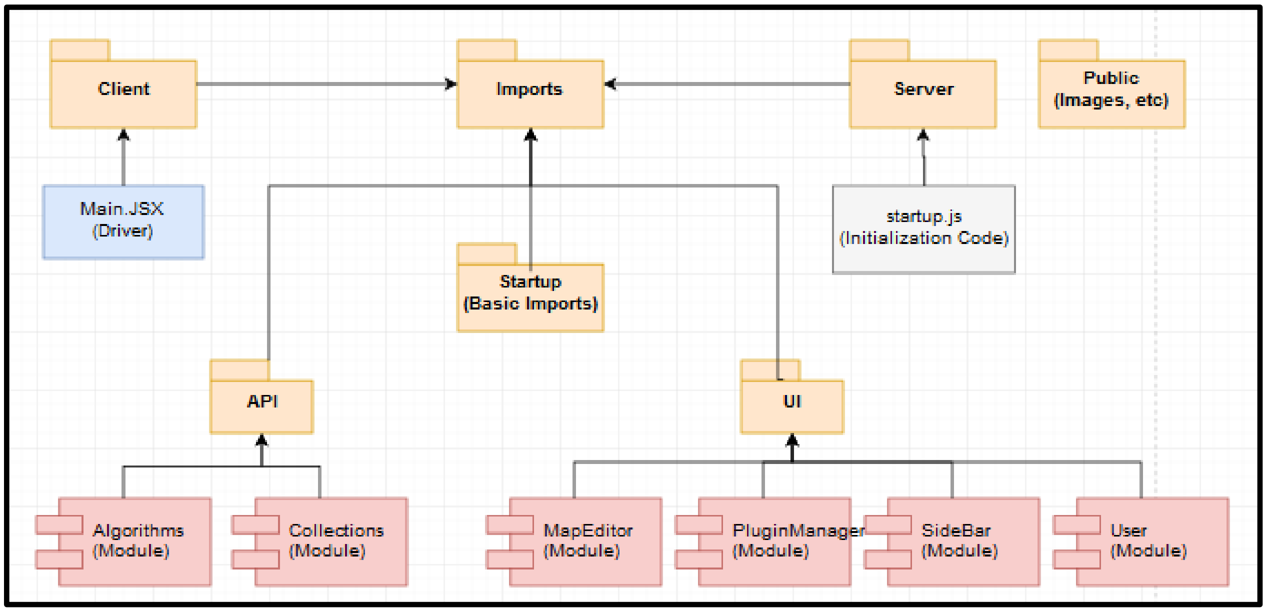


*Figure 20. An overview of them system in its current state. The diagram shows the various packages that make up the application, and the individual modules that are running. Within each module is a set of javascript files and React components.*

The client, imports, and server folders are traditional MeteorJS components that contain makeup the project. The client folder is generally the relevant HTML and CSS applied from the client side of the application. The server folder contains any server code that is  desired on startup of the application, as well as connections to web services. The impots folder contains the bulk of the project. This folder is broken down into the API, some small startup code, and the ReactJS components inside of the UI folder. Imports as a “middleware” between the client and server portions of the code.

The API folder contains two modules: Algorithms and Collections. The Algorithms module contains the pathfinding algorithm and other helper functions to accomplish repeated tasks throughout the program. Each module has some interface for accessing these functions easily. The collections module handles all of the database actions with MongoDB. Each collection has its own script for executing Mongo commands through the MeteorJS API. These are abstracted into simple functions such as addNode() that can be called anywhere in the application.

The UI folder contains four modules: MapEditor, PluginManager, SideBar, and User. The MapEditor module contains all of the React components for rendering the map editor interface and manages sending data between components that are part of the map editor. The PluginManager module behaves similarly to the MapEditor module but for the package manager system of the application. The SideBar module handles rendering relevant information onto the sidebar. This includes loading map names, applicable actions depending on who is logged into the application, and other tasks. The User module handles managing user status through the Meteor API and instructs other parts of the application what to render based on who is logged in.

# 7. Implementation

This section outlines each iteration completed throughout the course of this project. Each iteration lasted roughly one to two weeks, however, some are longer and are noted in the respective section. The time variation was due to complications in schedule from the author traveling and attending school. Code testing throughout each iteration was primarily done manually. Most of the code directly affects the user interface, so for each addition to the code, the team would go through each menu and enter dummy information to ensure the application was still functioning as intended. Manual testing would include adding, editing, removing, and loading maps/nodes, and eventually the same for plugins. If an issue was found, it would be documented as a bug and would become the next priority thing to fix and a commit would not be made until these issues were addressed. Testing is one area needing improvement in this project and this discussion is elaborated in section 8.3. Unit testing in JavaScript can be done through third-party frameworks such as Mocha and Chai similar to using JUnit in Java, but testing purely visual changes is not straightward in web development.

## 7.1 Iteration One

The goal of the initial iteration was to get a minimal application map editor in place. The minimal application should be able to add, remove, and move nodes. This also requires a database to be setup and be able to do the basic CRUD operations with the map data. However, it quickly became apparent that this was too much to do all at once.

The developer for Atlas was initially inexperienced with web development, and certainly a novice with using Redux. Redux plagued the first iteration, as inexperience caused serious issues with getting even the most basic features to work. For example, deleting nodes was one of the strangest behaving features of the first iteration. Sometimes only the most recently added node would disappear (visually) regardless of which one was selected for deletion, but upon refreshing the browser window, the *actual* node selected would be gone! Even stranger, before the refresh, if a user attempted to click on the node that was attempted to be deleted, the application would act as if it was not there. This issue had something to do with the state of the map being held in the reducers even though the database was updating correctly.

Using React and Redux together was very confusing because they both hold a state which is technically independent of the other, so part of the challenge is getting these states to be the same. This was the fundamental reason why Atlas' interface seemed to be out of sync with what was actually happening in the background.

Another issue with the first iteration was that too many libraries were added initially. Perhaps this is an oversight in design and the introduction of new libraries should have been more incremental. This was an issue because the author was unable to spend time learning how to use a single library and therefore many complications arose due to poor implementation. Many of the code demos provided by these libraries were too simplistic for Atlas' purposes, and if by some rare chance there were MeteorJS examples, they were almost always for a very early build of Meteor when the coding style was completely different.

The first iteration was not a total failure though. The map editor window was completed using the react-leaflet library. With it, the ability to add, delete, and reposition nodes was put into place and saved to the database properly. The Redux data flow was working to an extent and connected the map editor state to the redux state, updating the database in between. A toolbar was added to the map editor that allowed users to swap between different actions such as adding nodes and creating edges (a placeholder for now). The author also managed to do bounds checking for placing nodes only over the map image, however it was later found out in iteration 3 that this was actually incorrect! Fuzzy search for loading different maps worked by map name.

Although this iteration did not completely meet the goals it set, it was a good wake up call as to how much work would be needed to accomplish this project, and solid progress was made on different aspects of the application.

## 7.2 Iteration Two

With hope, the second iteration began and its goals were to get a pathfinding algorithm functioning and the following features to the map editor: edges between nodes, editing node properties, connecting nodes across multiple maps.

The A Star pathfinding algorithm was implemented in this iteration and appeared to be working as intended on a single floor. The author experimented with a few edge designs, and ultimately chose to keep a running list of edges within a map that denoted which nodes were part of the edge. Not only would this solution work now, but it would also stand in the future when doing cross-map edges because the pathfinding algorithm can simply look at a global list of edge objects. This system was chosen over using a node neighbors property of nodes because this would result in an infinite loop in MongoDB.

The majority of the development time was spent debugging Redux related issues in the application. The reducer code had to constantly be updated from complex to simple to complex, and it felt as though every time something unrelated would change, the reducer code would break again. The previous iteration had out of sync problems with deleting nodes, but this time it was with loading nodes upon switching maps. Sometimes when switching from map A, which had nodes, to map B, which had different nodes, some of the nodes from map A would overlay map B but in different positions!

There was also a problem with infinite loops occurring from the redux dispatcher. In some cases, updating the database would trigger redux to think that the subscription to the database kept changing even though the update call occurred once. It seemed to happen if actions occurred too quickly, meaning some kind of race condition was happening with the database and redux, and the map would eventually disappear until the page was refreshed.

By the end of the iteration, it was decided to restart the project from the beginning again as a giant refactoring of the current project. While not a total overhaul since many of the React components were standalone, the project was made anew without the middleware libraries such as Redux and within an afternoon the application was up and running again at almost the exact same point in a fraction of the original time. The codebase was much simpler and the data flow was very clear. The author was also able to create an API for interfacing with the database and updating component states. This decision saved many hours of debugging and improved the sanity of the author.

## 7.3 Iteration Three

Many new additions came with iteration 3 after scrapping Redux in the previous iteration. The focus of this iteration was adding many small features to improve the usefulness of the map editor.

Dynamic image overlay fitting was added to ensure that no matter what the size of the map image, the leaflet container will fit the image. This also sets the default zoom levels and locks the minimum and maximum zoom.

The ability to find locations was improved with a fuzzy search system and is now populated based on only nodes that are marked as locations. Searchable nodes now have a “Locations” type and is one of a few types with the new attributes system to differentiate them (ie: hallway, location, etc).

This iteration added the ability to upload custom map images and save them into the local project directory. Any image can be uploaded given it is within a certain size limitation and of a specified file type. A unique link is created for this image and it can hosted directly from the web app which remains persistent between bringing the application up and down.

Lastly, a sidebar navigation menu was added using the react-sidenav library. The intent of the sidebar was to give both users and administrators more options for interacting with the application without the clutter of having too many buttons. With the sidebar came many new additions to the map editor, in the form of popups. These popups were made using the react-popups library and provide functionality to add, edit, remove, and load maps.

## 7.4 Iteration Four

Iteration four's focus was mainly bug fixing and making quality of life changes. Quality of life changes are features that enhance the user experience by making certain tasks easier for users. There are not any major updates, however, the pathfinding algorithm was finally corrected to accurately select the best path. The previous error was due to not selecting the lowest fScore (the neighbor node with the lowest cost to get to) and was rectified by sorting the list of potential neighbors to travel to. On top of this, the algorithm was upgraded to run on a global set of nodes instead of only working within a single map. To do this, the ability to connect nodes across maps was added by simply searching for the node to connect and updating the edge objects to also contain a list of maps they are part of. This small change to edges was necessary for being able to display the route preview with all maps involved.

A few minor changes were made to setup code for the next iteration. One of these changes is that the list of maps traversed is added to the edges in order of traversal. This simplifies the process of showing map image previews once a route has been established. Another change is that edges denote maps by Mongo assigned ID instead of map name which it had done previously. This is to ensure that the maps are unique in the event of duplicate map names (which is allowed).

## 7.5 Iteration Five

The focus of iteration 5 was to begin to differentiate user and administration interfaces via an account system. In addition, a major new feature to building many different applications was born: the plugin manager.

The new account system is based upon Meteor's built-in account system. It provides easy to use methods to create accounts, secure passwords, and login/logout. This system was integrated with the sidebar to allow easy account management and a visual to indicate if a user is logged in or not. The actual login/account creation is done via another popup that is invoked when clicking on the account section of the sidebar. Currently, doing the *actual* differentiation was not completed in this iteration, but the general idea is to show the different user types information relative to them. For example, we would not want a general user to have access to the map editor!

The route preview container was added and appears after making a route. It shows, in order, the different maps traversed across throughout the route (if there is more than one). Clicking on any of the images will load the corresponding map into view. This feature was coveted by members of the Brigham and Women’s Hospital during previous Software Engineering classes so it seemed natural to include it in this project.

The final addition to this iteration was a preliminary version of the plugin manager. The plugin manager is a system to create a more universal application. Many pathfinding systems are built for specific locations so the entire codebase is focused on a single scenario. In order to be applicable for multiple places, there needs to be a way to select features relevant to the use case. For example, a hospital using this application might want a directory of the people who work there. That way users could not only search for rooms, but also doctors and their offices. Of course, an airport would have no use for something like this, but perhaps they want a feature to access their flight time data so a user could find the Gate associated  with a certain flight and be routed to it. The plugin manager is a way to allow administrators to pick and choose extra features for their application and configure them.

The manager itself is a shop-like system similar to the Chrome Add-ons store where a gallery of available plugins are shown and administrators can choose to install/uninstall the ones they want to use. This is a powerful tool because it opens the door to a few major benefits. First, it can provide many analytical options for the application. For example, a plugin for keeping track of frequented locations and generating heat maps of an area; or maybe logging general information such as time of day. These features would help businesses plan the layout of their facilities to better accommodate their customers. Second, the plugin manager allows outside developers to contribute their own features. The plugin manager provides a set of base classes that can be used to create any kind of plugin and submit it for approval to the system.

Each plugin requires multiple components to function correctly. A plugin needs to be able to run on the administration side as a configuration view, a sideview for displaying general information, and a user side view (if applicable). The user side view would be able to be loaded by the user to interact with that feature. In the case of the hospital directory, a user could select the "Directory" plugin and be able to search that database for information. Sometimes the plugin would tie into the main application's search as well, so this system is robust enough to handle these actions. The rendering is done by using the factory design pattern and going through the list of registered plugins. While this iteration did not complete the full system, the interfaces were laid out and it was accessible via the sidebar. There were not any function plugins or means to install them as of yet.

## 7.6 Iteration Six

While the previous iterations were generally one week to two weeks long, iteration 6 encompasses the entirety of A term. As this was an MQP, the author was only registered for work during the summer and B term, and with an overloaded schedule, development was sprinkled in when possible so this iteration is a summary of the work done during the off-term.

The biggest addition this iteration was the long awaited differentiation between admin and user interfaces. A custom map component appears on the user side that allows them to find routes and click on location notes for further information about the node. Only location nodes will be visible on the user side. The sidebar functions were placed with maps in the system, so users could easily swap between maps for viewing. Plugin names were added to the sidebar, however, their user/admin implementations were not. The plugin names were formatted to be in title case instead of their constant-like names previously. Plugins could now be installed via the main plugin manager screen and only installed plugins appear in the sidebar.

A few changes were made to the login features. A default admin account is loaded into the system on startup. This is more of a convenience feature and would not remain in a real production build. The application will automatically logout before the application closes. Logging in can now be done via email or username, unlike solely by email in the past.

Lastly, there was a lot of CSS work done during this time to generally improve the look and feel of the application. The majority of this work was in the plugin manager view to better position the plugins and give them a more uniform look. The bootstrap library was added during this time for nicer looking buttons and text fields to save development time.

## 7.7 Iteration Seven

This iteration spanned the first half of B term and its focus was making the application have a more professional look and feel. It was apparent that time was a major factor and while the application is functional enough to build demos, it is not quite at a point to say it accomplishes all of the goals it out set to do.

On a feature level, the overall system design was overhauled to swap components in an out via a Layout Manager component. This drastically reduced the overhead of swapping between user and admin components, and very elegantly controlled which components would render inside the main body of the application. Previously, there was a convoluted system of passing React props back and forth to decide whether to load the map editor, user map, plugin manager, etc.

Development time was also spent building functional plugins with views on the admin and user side for building demos. During this time, it was decided that plugin building was very time consuming because there was not a uniform styling system in place so making new interfaces was a daunting task. Bootstrap was in use, but it was decided instead to switch over to Material-UI Design. The remainder of this section will describe the journey of upgrading the application to use Material-UI design.

The original front-end relied mostly on custom CSS and a few libraries for formatting. For example, the sidebar was made using the react-sidenav package but the styling was all custom. This system worked, and there was a definitely a lot of educational value in doing it by hand, but it was very time consuming trying to precisely position components and to make a uniform theme across the application. It seemed like much of the development time was spent styling and re-positioning elements than feature development.

Bootstrap was already in use and is one of the most well known styling libraries for web development. It contains many layouts to use as well as custom styling on all of the HTML components. Bootstrap comes with many examples, themes, and documentation to use it. Material-UI is a React version of the Material Design library. It provides many React components styled with Material Design and allows for easy customization of components. Out of personal preference for aesthetics, the author decided to use Material-UI.

Updating the front-end proved to be a fairly time consuming endeavour, however, the end results far outweighed the investment. Most of the components could easily be replaced by Material-UI components. For example, the standard <input /> tag in HTML could be swapped for a TextField component, and these new components offered more flexibility in what they are used for. The biggest change was handling popups throughout the application. Originally, a library called react-popups was in use that required a fair bit of code to render each popup. With Material-UI, there is a Dialog component that replaced this system very nicely, but it required rewriting most of the functionality of these popups from scratch. This was time consuming, but it drastically reduced the complexity of the code so this was a welcome change. A majority of the React components did have to change, but for the most part they were smaller changes that easily swapped out the old code. The biggest advantage in upgrading to Material-UI was the uniformity it brought to the application. Every component has the same look and feel, and also a modern and visually appealing one. While visuals are not true functionality, it is important to have an aesthetically pleasing application to add an extra level of professionalism. Material-UI also came with many small animations when interacting with the application that truly add a polished feel that improves the user's experience. Everything feels much smoother and there is a clear performance boost.

# 8. Assessment

This section looks back on the project as a whole and reflects upon the project's progress from start to finish.

## 8.1 Migrating to Web Technology

This project served as an exploratory means to building a web version of a pathfinding application. Migrating to a web technology proved to be more daunting than initially anticipated by the author. However, a web based implementation is definitely the way to move forward with this type of project. There are nearly infinite ways to set the look and feel of the application, and many libraries for doing so, such as Bootstrap or Material. This alone gives so much flexibility that no two applications will look alike, and gives developers the freedom to easily make a beautiful looking application.

In addition, the web has become so popular that there is a library for almost anything one could imagine. Many libraries are for ReactJS specifically, meaning a developer could save so much time using a component from another library without having to invest the time to develop one himself. There are components that handle auto completing text, editing tables, managing navigation between views, and many more.

Web development comes with the added benefit that the project is compatible in some way with any web-capable device. Instead of a standalone application, a web version is always available and accessible through a web browser. Users do not need to worry about downloading and installing it, and having the right libraries installed on their system. This opens the application to many more users and cuts down on configuration issues.

## 8.2 Lessons Learned

The educational value from developing this project is unparalleled to any other projects completed at school. In some respect, this was an introductory project to web development for the author. Full web development requires knowledge of many different technologies, and in Atlas this included front-end development, back-end development, database management, user experience, documentation, and testing. Each one of these concepts has its own programming languages and libraries that need to be compared and contrasted to get a project functioning and can be overwhelming at first. By developing Atlas, the author has a deeper understanding how each of these components interact and have multiple options when working with each one.

Atlas provides a good basis for what is required to build a full web application. Even having prior experience developing a pathfinding application only got the author so far. Incremental development through the agile methodology worked very well in this setting. Smaller, testable features could be added one at a time and this improved the overall quality of the code since the majority of it was modular. Atlas is in a functional state and could be used to demo very basic pathfinding across multiple regions of a location, and this alone is a huge positive for this project.

## 8.3 Areas Needing Improvement

Testing should be more formalized and thorough. Research into testing technologies for React and Javascript needs to be done to improve the quality of Atlas and any future projects. The majority of the testing was done manually since much of the project was user interface design. This means that while problems with the UI were well known and documented in JIRA, it also means that every aspect of the UI was not consistently tested for each new feature. This did not cause any issues down the line or result in major bugs sneaking into the program since anything that went wrong was documented and debugged.

Working as a single developer on a project of this scale is difficult. As explained in section 8.2 about all of the different components of a web project, there is simply too much for one person to continuously balance. This means that a single developer would have to become an expert in all of these fields to make a truly polished application, and since the developer started with almost no web experience, the reality of this happening is nearly non-existent given the timeframe of this project. Having a team of three or four developers who specialize in different areas would set the stage for a truly well-rounded and polished product in the short lifetime of an MQP project. There were many times when the developer was simply too burnt out from researching and programming all of the time, and this greatly reduced morale and project quality. On top of all this, documenting everything in report format was extremely time consuming and that detracted from time spent developing.

Atlas is definitely a good start on a pathfinding framework, but it did not meet all of its goals. While it does technically support any location, the mechanisms in the application to meet this goal are still very primitive. Mobile compatibility is a big goal that was not met as it requires parallel development of a mobile-friendly version of Atlas. Simply put, creating a very polished mobile version could be a project in itself. Atlas *could* technically be accessed via browser, but the interface is not optimized for this in its current state.

One final point is that more research into web technologies should be conducted if another version of this application is created. There are still so many frameworks and libraries that the author did not get a chance to fully learn and understand, and it is very likely that the frameworks chosen for this project could be improved.

# 9. Future Work

This section will outline features that were planned to be implemented, but due to time constraints were not completed as well as other functionality to improve the application.

## 9.1 Mobile Compatibility

Mobile compatibility is a major feature that would improve the usefulness of the application. Having an application accessible by phone, for example, increases the amount of potential users. This is especially true for a pathfinding application since many users of the application are on the go. Mobile was not a high priority because the main focus of Atlas was building the top-level administrative framework.

To build the mobile application, a separate version could be created using React Native. The majority of the codebase would not need to change for running Atlas with React Native. Another great benefit to React Native is the application would be deployable on both Android and iOS, instead of having to maintain separate versions. For the mobile version, it may be worth switching over to Firebase as the main database (and therefore in the web application as well) , since Firebase is built for mobile and performs well for web applications too.

## 9.2 Bridging Outdoor to Indoor Transit

The ability to go from any location outside to a place defined within the pathfinding application would improve be beneficial. Doing this requires a mechanism for taking an incoming route, from Google Maps for example, and determining which point becomes the starting location for the indoor section and ultimately going to the true destination. An example route might be from a user’s house to Doctor Smith’s office in room 212. This requires support for multiple different GPS tools, such as Google Maps, Apple Maps, etc., and is country dependent. Having this seamless transition would allow users to easily get to their ultimate destination, but to do this in one step may require integration with the other GPS tools on their side to allow Atlas to takeover at a certain point.

### 9.3 Directions, Distance, and Time Estimation

Atlas is currently missing proper step by step directions, distance calculation, and time estimation. These are all features we are accustomed to having in our everyday pathfinding applications and should be included here as well. There are fields in map entries that account for map scale (if being taken from engineering floor plans, for example) already in the codebase but due to time constraints, these features were not implemented. While not particularly difficult to implement in a primitive stage, getting accurate calculations for distance and especially time would be a time consuming task and could be implemented by a separate developer.

### 9.4 Default Plugins

Atlas as a completed application would contain more default plugins than it currently has. Some plugins are specific to certain locations, but many of the “out-of-the-box” plugins would provide proper analytics and debugging capabilities to administrators. In addition, the plugin manager system should be more automatic and allow for custom plugins to be added to the main application through a well documented API. This system should be synchronous with the mobile version of the application.

### 9.5 Error Checking and Handling

Many fields entered in the application do not go through proper error handling if they are invalid. Every variable sent into the database is properly type checked to ensure at the very least the data is in the correct format, but in the event this is not true, the application does not inform the user. Having better visual feedback for things going wrong in the application is important as a general user is not going to look at the developer’s console when something goes wrong, and an administrator should not have to. Solutions could be as simple as alert messages, but ideally text fields and other input boxes would constantly error check values as they are typed and provide visual feedback indicating if the data is valid or not. Form submission should also be disabled until all of the data has been checked and entered.

## 10. Conclusion

The goal of this project was to create a web-based pathfinding framework for building indoor GPS-like applications. While many of its final goals were very ambitious, such as full compatibility with any web compatible device, this project was still successful as a proof of concept. The ability to turn simple map images into routable graphs exists, albeit in a fairly primitive state, and it could be used to route between multiple graphs. Atlas has a mechanism for visualizing the entire route and swapping between different maps, but the user experience is definitely lacking and requires a lot of manual input. Atlas has the capability to allow for any location specific feature to be added fairly automatically with the plugin manager system, however due to time constraints, proper showcasing of this feature is not present and more work needs to be done to polish it into a proper API.

Overall, Atlas is a decent start in building a web-based pathfinding framework but it is nowhere near finished. The author was was able to complete many of the goals originally made, but there are still features that did not get addressed. Atlas was an extremely invaluable experience on web technologies (and its complications!) for the developers and serves as a great basis for the level of planning needed from start to finish in creating a large-scale web application.

## 11. Bibliography

Agile Manifesto. (2017). “Manifesto for Agile Software Development.” Accessed July 17, 2017. http://agilemanifesto.org/.

AppFusions. (2010). “CVCS vs. DVCS In a Nutshell - Product - Source Code Importer for GitHub.” Accessed July 17, 2017.

Atlassian. (2017). “JIRA Software - Issue & Project Tracking for Software Teams.” *Atlassian*, https://www.atlassian.com/software/jira.

Atom. (2017). “A Hackable Text Editor for the 21st Century.” *Atom*, https://atom.io/.

BitBucket. (2017). “Bitbucket | The Git Solution for Professional Teams.” *Bitbucket*, https://bitbucket.org.

Brackets. (2017). “A Modern, Open Source Code Editor That Understands Web Design.” *Brackets*, http://brackets.io.

Bruegge, Bernd, and Allen H. Dutoit. “Object-Oriented Software Engineering, Using UML, Patterns, and Java.” Prentice Hall, 2010.

Class Project Report. (2017). "Default Report", Software Engineering CS3733, Worcester Polytechnic Institute, Worcester, MA

CPrime. (2013). “What Is Agile? What Is Scrum?” *CPrime*.

Crunchbase. (2016). “WiFiSLAM | Crunchbase,” https://www.crunchbase.com/organization/wifislam.

Firebase. (2017). “Firebase.” *Firebase*, https://firebase.google.com/.

*Flux: Application Architecture for Building User Interfaces. JavaScript. 2014. Reprint, Facebook, 2018. https://github.com/facebook/flux.*

GitHub. (2017). “Build Software Better, Together.” *GitHub*, https://github.com.

Google. (2017). “About - Google Maps,” https://www.google.com/maps/about/partners/indoormaps/.

Infsoft GmbH. (2017). “Indoor Navigation, Indoor Positioning and Indoor Tracking by Infsoft.” Accessed July 17, 2017. https://www.infsoft.com/.

Jackalopes, Jazzberry. (2017). "Hospital Way Finding" Class Project Report, Software Engineering CS3733, Worcester Polytechnic Institute, Worcester, MA

JetBrains. (2017). “WebStorm: The Smartest JavaScript IDE by JetBrains.” *JetBrains*, https://www.jetbrains.com/webstorm/.

Mack. (2014). “Five Advantages & Disadvantages Of MySQL.” *DataRealm*.
MongoDB. (2017). “NoSQL Databases Pros And Cons.” *MongoDB*, https://www.mongodb.com/dynamic/node.

Patently Apple. (2016). “Apple Indoor Mapping Invention Automatically Determines the Mall Floor a User Is on and Updates the Floor Plan.” *Patently Apple*

Reactide. (2017). “Reactide.” Accessed July 17, 2017. http://reactide.io/.

ScrumDo. (2017). “Online Scrum Tool / Agile Scrum Software.” *Online Scrum Tool / Agile Scrum Software*, http://www.scrumdo.com.

Software Testing Help. (2017). “What Is SDLC Waterfall Model? — Software Testing Help.”

Srivastava, Bikesh. (2017). “What Is Agile Methodology?Disadvantage of Waterfall Model.” *LinkedIn Pulse*.

Statista. (2016). “Unique U.S. Visitors to Mobile Apps 2016 | Statistic.” *Statista.*

Team Gamma. (2017). "Project C" Class Project Report, Software Engineering CS3733, Worcester Polytechnic Institute, Worcester, MA.

Team Jade Jersey Devils. (2017). "Project C" Class Project Report, Software Engineering CS3733, Worcester Polytechnic Institute, Worcester, MA.

Team Purple Pirates. (2017). "Project C" Class Project Report, Software Engineering CS3733, Worcester Polytechnic Institute, Worcester, MA.

Trello. (2017). “Trello,” https://trello.com.