CS476 Tech Feasibility
Transit Trackers

24 October 2024

Drake Stanton, Lauren Bushman,

Alonso Jimenez Alamilla, Benjamin Griepp

NAU Shuttle Services

Paul Deasy

Table of Contents

Introduction

Transportation is a vital aspect of any efficient modern society. Being able to reliably get to where one needs to be is a daily necessity for nearly all members of an organized community. At Northern Arizona University, thousands of students, faculty, and staff rely on the campus shuttles to get to class and work on time. In 2023, there were 1.3 million individual rides provided around campus by NAU Shuttle Services. This department is responsible for the care of the university's physical assets, which mainly consists of around twenty buses. NAU Shuttle Services also manages several dozen employee bus drivers, many of which are part-time student drivers who often don't have a consistent schedule week-to-week. Everyday, these drivers and buses must be arranged into a smoothly-flowing schedule in order to keep all of the routes operating as they should.

The current system used to create the daily driver schedule is ineffective and time-consuming. The management team, consisting of Michael Seitz, Rafael Riviera, and James Volturo, must be very involved and do many tasks manually that could be automated instead. The system consists of several parts: a third-party scheduling system called Shiftboard, a spreadsheet, and another third-party software called BusGenius that tracks the physical location of the buses and when the drivers begin and end their shifts. A typical exchange goes like this:

1. A manager sets the available shifts in Shiftboard.
2. A student driver sees that a shift is available and picks it up in Shiftboard.
3. Shiftboard sends a notification via email to the management team.
4. A manager manually enters the shift and driver information into the spreadsheet, and assigns an available bus – one that is not undergoing maintenance.
5. The driver clocks in using the BusGenius tablet in the bus.

6. BusGenuis gathers all of the clock-in/clock-out information into a graph.
7. A manager interprets the graph to ensure employees are working at the correct time.

On a typical day, the first part of this exchange will occur multiple times, requiring that the spreadsheet be manually updated often. This current scheduling process requires many man hours per week that could be saved by the development of a more integrated, automated system. In this document, our group will propose such a system and argue its technical feasibility.

The new scheduling system will be an interactive web application that will replace Shiftboard and the spreadsheet, while integrating BusGenius. The system will support logins for both driver and manager/administrator users, and will have the following functionalities:

- Provide a clean Gantt-chart style display of the bus schedule, showing available shifts.
- Automatically carry over the set driving schedule for full-time employees every week.
- Allow student and part-time drivers to pick up shifts.
- Automatically assign an available bus to a recently picked up shift.
- Allow administrators to edit the available shifts and manually assign drivers and available buses when necessary.
- Allow administrators to change the status of a bus or other asset as necessary.
- Take the clock-in/clock-out data from BusGenius and provide management with valuable statistics about employee work habits.
- Allow administrators to send email and text messages to employees through the system.

We believe that the development of this new scheduling system will have a significant positive effect on the daily operations of NAU Shuttle Services. It will save many hours of work every week, and will allow drivers to be more confident in the shifts they will be covering on a

given day. Overall, this system will allow NAU Shuttle Services to continue to serve NAU students and faculty and provide smoothly running and reliable transit.

In the next section we will outline a more detailed analysis of the technologies we will use in development. We will present some of the potential technological challenges that are inherent in this system and the services we believe are best suited to addressing these challenges.

Technological Challenges

The development of this software will include many of the following technological challenges that we will need to address. Our system will require the following:

- A secure login and user authentication on our web app.
- A reactive and intuitive frontend to ensure both users and administrators are seeing up to date data and understand the information displayed easily.
- A reliable backend to ensure the web app is consistently available and is up to date with the database.
- A reliable SMS and Email API to ensure the system and administrators are able to communicate with users outside of the web app interface.
- Access to and understanding of the BusGenius API to implement clock-in time reliant features.

Technology Analysis

Frontend framework - React

The challenge is to build a platform that is both user-friendly and interactive for our scheduling web app. The interface needs to allow users such as admins and employees to be able to create, view, edit, and manage schedules efficiently. The platform should support data visualization to display schedules and also support real-time communication to support shift updates. Key functionalities should include a dashboard for admins to manage employee shifts, an employee-facing schedule view, and shift management features. The framework should support dynamic interfaces, promote a great user experience, and also integrate well with the database we choose and other back-end APIs.

Desired Characteristics - The front-end framework we are looking for has a lot of desired characteristics. The framework should enable fast rendering and be responsive to handle real-time updates without lag. This is important because scheduling apps need to display frequent data changes efficiently to ensure a smooth interaction for the user. The framework should allow the code structure to be easy to manage, and update, making it easy to scale the application over time. This is important because the client is looking for an application that can be maintained and updated after we have all graduated. The framework should also have large community support to allow us to have access to a plethora of third-party libraries and make finding solutions a lot easier. In addition to this, it should integrate well with a lot of back-end APIs, real-time messaging tools, and data visualization tools to make sure that we can build the project without encountering a lot of issues and can deliver exactly what the client wants.

Alternatives - Some possible approaches we can select for this project are using React, Angular, or Svelte which are all common front-end frameworks that are used widely today. React has been adopted by companies such as Facebook, Airbnb, and Netflix. It was originally developed by Meta and is also still maintained by them today. It has been around since 2013 and is one of the most popular JavaScript libraries for building user interfaces. It offers a component-based architecture and a virtual DOM for efficient UI updates. Angular was developed by Google and was released in 2010. It is a comprehensive front-end framework that has been used in applications such as Gmail. It is most suitable for large-scale applications due to its structure. Another interesting choice would be to go with a framework such as Svelte. Svelte was developed by Rich Harris in 2016. It gained popularity for its simplicity and speed with companies such as The New York Times and IKEA using it for some of their projects.

Analysis - After analyzing those three choices, in terms of performance, React renders efficiently with virtual DOM and supports lazy loading for better performance. Reach performs well when it comes to real-time updates compared to other options. Angular has a lot of mechanisms in place that slow down performance but is effective for data-heavy applications. Svelte offers better performance than React since it does not have as much overhead making it better for smaller applications that could benefit from high performance. In terms of ease of maintenance, React's modular structure can make it easier to maintain as the app evolves. Angular makes it a lot harder to maintain down the line because the structure is very complex. Svelte's built-in functions make development a lot faster but it could make it harder to maintain as the app scales. Angular and Svelte also lack the depth of community resources that React can offer. In terms of community support, Reach offers the largest by having regular updates, and extensive third-party library support. Angular and Svelte have a growing community but not as

extensive as React's. For integration capabilities, React has easy integration with APIs and data visualization libraries such as Chart.js. Angular also offers a lot of support for integration with REST APIs and third-party services. Svelte integrates well with APIs and other back-end services but not as much when compared to React.

Based on our analysis, React is the most suitable choice for the project. When comparing it against all other choices, it is the best choice in terms of performance, ease of maintenance, community support, and integration capabilities.

Framework	Performance	Ease of Maintenance	Community Support	Integration Capabilities	Average
React	5	5	5	5	**5**
Angular	4	3	4	4	3.75
Svelte	5	4	3	4	4

React is the most promising solution due to its modular architecture, fast rendering with virtual DOM, and extensive community making it the ideal choice for developing the scheduling web application.

To validate that React is the best and right choice, we will outline the following demos for the "Technology Demo" assignment. We will create a mock-up for a basic scheduling interface, and show what the Gannt chart visualization will look like. We will also demonstrate what the real time messaging will look like.

Backend - Firebase

Choosing an appropriate backend service is crucial to the entire development process of the application as the backend will manage serving the frontend application, user login, database management, and API requests.

Desired Characteristics - The key characteristics of a backend service for our project are reliable service of the application, reliable integration with our APIs, security, pricing, ease of development, and ease of future maintenance. Reliable service is integral to our application as all users and administrators must be up to date with current scheduling. Reliable API integration is especially important for sending SMS/Email notifications to users to ensure urgent messages reach users timely, as well as ensuring BusGenius data is correct. We must have appropriate security rules to ensure the application is not compromised by an outside user and to protect private user data. We must keep backend pricing within our sponsor's budget and especially important that we do not incur unnecessary costs due to unnecessary/unoptimized computation. Ease of development will help ensure the application is developed within the deadline as well as done with minimal bugs to maintain the longevity of the project. Ease of future maintenance is crucial to our project as the project will ideally be utilized long after development, ensuring the backend is accessible and understandable by future maintainers will help ensure the application's longevity.

Alternatives - The three backend services we have considered are Firebase, AWS, and NAU's ITS. Firebase is Google's backend as a service and has operated since 2012. It is often praised for its prioritization of developer experience through an intuitive interface, well-structured documentation, and easy-to-use software development kits. Firebase provides built-in systems for real-time databasing, authentication, serverless functions, and computation

optimization. AWS (Amazon Web Services) is Amazon's backend as a service and has operated since 2006. AWS is praised for its flexibility of use allowing control of virtually all backend elements and integration with a wide array of technologies and services. AWS shines in supporting highly specialized architectures but comes with a steep learning curve and understanding of concepts such as virtual machines and networking. NAU ITS provides general-purpose web and database servers to support departmental applications and web pages via Unix servers and MySQL support. This option has the benefit of cost effectiveness but provides little interfacing, documentation and software development kits leading to a steeper learning curve and increased complexity of maintenance.

Analysis - To analyze these options we thoroughly researched the three and documented the overall pros and cons of each approach. Firebase provides the most streamlined developer experience by providing built in software development kits to handle databasing, serverless functions, authentication and computation optimization. Several of our team members have past experience with Firebase and looking through past projects of ours, our team agrees the interface is intuitive and easy to navigate, pricing is competitive and the code implementation for frontend and backend code is straightforward and well documented. The automatic virtual machine instance handling of serverless functions makes for a straightforward approach to handling backend requests without having to worry about manually managing multiple instances during times of high request loads. The serverless functions use an identical syntax to ExpressJS leading to a straightforward learning curve and low boilerplate. The NoSQL approach to databasing leads to a more intuitive approach to storing and indexing data while automatically optimizing database requests, reducing time writing SQL queries that may be unoptimized. The low configuration nature of Firebase leads to straightforward security rules that reduce the likelihood

of vulnerabilities which is essential to our application. AWS provides a highly customizable environment which is beneficial for unique systems utilizing custom or niche technologies making it a popular choice for large software projects. Some of our team members have past experience with AWS and found this high allowance of customization to be overwhelming, forcing the user to manage virtual machine instances and their network interactions manually as well as the database structure and fine-grain IAM security rules. AWS Lambda functions allow for greater control of function triggers allowing for triggers other than HTTP requests such as changes to other AWS deployments but therefore require a large amount of boilerplate in a custom YAML syntax to configure. Function bodies are more complicated as well and require a large amount of boilerplate as well as manual encoding/decoding of JSON data. Pricing on a large scale is similar to Firebase but is more complex and tiered, often charging higher prices for low usage than Firebase which offers a generous free tier that could entirely encompass our application's amount of computation. NAU ITS offers little developer tools requiring the entire system to be essentially built from scratch. This option would have no serverless function support requiring all server request threads to be manually handled with a traditional server. All security rules would need to be manually implemented via a custom login/authentication system leading to the highest likelihood of the three to security vulnerabilities. Additionally there is little documentation for the service and no user interface making maintenance by an outside party extremely difficult. The sole benefit of this option would be the free pricing, however, an option such as Firebase could very well not incur any cost as well given the small user base of the application.

Backend Service	Ease of development	Ease of maintenance	Customization	Pricing	Security	Average
AWS	3	3	5	2	4	3.4
Firebase	5	5	3	4	5	**4.4**
NAU ITS	1	1	4	5	1	2.4

Overall the table shows AWS as a solid option for a highly customized backend infrastructure but incurs a higher learning curve for development and maintenance and therefore a longer development time and likely will be the most expensive option. This high amount of configuration may also lend itself to a higher likelihood of security vulnerabilities. NAU ITS has the benefit of free pricing but will by far incur the highest development time and make for difficult future maintenance and requires complete manual security configuration making it the most vulnerable of the options. Firebase provides the most streamlined development and maintenance experience of the three with the sacrifice of customization of the architecture. However this customization is really not necessary to our application as it will have a fairly standard web application infrastructure. The low configuration nature will lead to low likelihood of security vulnerabilities and the generous free pricing tier could lead to the application incurring no or very low cost. This all leads us to the conclusion that Firebase is the best option for our application and really does not incur any downsides over the other options besides possibly the price when compared to NAU ITS.

To highlight the technologies usage in our tech demo we plan to show Firebase working with a user login/authentication as well as the frontend making changes to the database and reflecting these changes real time in our frontend.

Secure login/ user authentication - Firebase Authentication

For our project, we will require a system for secure login and user authentication. This is a critical aspect of the project. The application needs to ensure that only authorized users such as admins and employees can access sensitive information and perform actions like scheduling and messaging. Key functionalities include secure user registration, sign-in, and password reset.

Desired Characteristics - There are several key characteristics that we must consider to choose the right approach for authentication. The first one is security, the solution must provide a lot of security features such as encryption and secure session management. It is important to protect sensitive scheduling data and user credentials to minimize unauthorized access. The service should also be relatively easy to integrate with the front-end framework we decide on and the back-end. This is important to ensure a seamless login experience and reduce development time. The service should also allow for custom user roles to personalize access controls. This is so we can enable specific permissions and role-based access controls within the scheduling application.

Alternatives - Some approaches we are considering for this aspect of the project include Amazon Cognito, Firebase Authentication, and NAU CAS. Cognito is designed by Amazon Web Services and can handle sign-up, sign-in, and access controls. It offers a lot of scalability and customization but is more complex to set up than other options. Firebase Authentication was developed by Google as part of the Firebase suite. It offers a lot of user authentication capabilities, including support for various sign-in methods, for example, email/password, phone, Google, and Facebook. It is also designed for easy integration with other Firebase services making it highly compatible with our project backend. Another approach we could go with would be to use NAU CAS. This is a single sign-on solution that is used by NAU for

authenticating users across all of its services. It is designed for integrating with all of NAU's internal systems which would make the authentication process familiar for university staff and students. However, this could require some custom integration.

Analysis - When it comes to analyzing these three methods in terms of security, Amazon Cognito offers a lot of strong security features, including MFA, password policies, and encryption. NAU CAS provides a secure SSO functionality which is tailored for NAU's systems and infrastructure already. However, it may require a lot of additional setup to gain access to that system and to implement it into ours. Firebase authentication, like Cognito, provides a lot of built-in security features and various sign in methods which synergize with a lot of Google's systems. In terms of the ease of integration, Cognito integrates well with React but it can be complex to set up given AWS's vast ecosystem. Firebase authentication will integrate seamlessly with other Firebase services such as their database. Since we chose to use firebase for our database as well, it could offer a more consistent development experience and make it easier on us since it would integrate seamlessly. In terms of scalability, Cognito is built for thousands of users to be registered simultaneously and will scale automatically which would make it a great choice for large scale projects. Firebase scales well for small to medium sized projects which would be suitable for the scale of our project. NAU CAS's scalability really depends on NAU's infrastructure and policies. This could prove difficult during the development process. In terms of customization, Amazon Cognito is highly customizable and would allow us to designate custom user roles and access controls. Firebase also provides the same level of customization. NAU CAS could pose some potential constraints since the roles are predefined.

Authentication Service	Security	Ease of Integration	Scalability	Customization	Average
Amazon Cognito	5	3	5	5	4.5
Firebase Authentication	5	5	5	4	**4.75**
NAU CAS	4	1	4	2	2.75

With Firebase already chosen for our backend, Firebase Authentication becomes the most logical choice for our project. It would be the easiest to integrate while also providing great security features that are necessary for our project.

SMS/Email API - Twilio

A major challenge identified for this project is the development of an alert system that allows bus managers/administrators to send bus drivers currently working messages/information. The system should be able to support SMS and email messages, ensuring that they are delivered in real-time, and in a reliable manner. To add to this, scalability is a concern with this system; the system should be able to handle messages to individual drivers, or larger scale groups of drivers. The challenge with developing this system is that we must find a solution that meets these requirements, is cost effective, and easily integrable into broader systems.

Desired Characteristics - In order to implement the alert system, we must concoct a solution that should exhibit (ideally) all of the following characteristics: real–time messaging, cost of efficiency, scalability, ease of use, reliability, and integration with the backend. The alert system is to be implemented with an emphasis on real-time messaging. The significance behind

this is that certain information (emergencies, route openings/closings) needs to be delivered to the bus drivers instantaneously. Real time messaging eliminates the possibility of delayed alerts, resulting in bus drivers receiving critical notifications that are impervious to their job. Cost efficiency is also important, as the service will be used frequently, so it is important to utilize a solution that will avoid becoming a financial burden to the client. The chosen solution should offer a pricing model that minimizes the costs/fees of SMS and emails sent to the bus drivers, especially if the project were to be scaled to larger groups, i.e. scalability.. This segues into another characteristic of the alert system, scalability. The alert system needs scalability because it needs to send small scale notifications to specific people, or larger scale notifications to larger groups of people. Sending an alert to a single driver to broadcasting messages to entire groups ensures that the system remains responsive and scalable. In addition, the alert system needs to be easily usable for users in order to get messages out quickly, especially if they are time sensitive messages. Simplicity in design ensures ease of usage, and therefore reduces human error when sending alerts. Reliability is also an important characteristic of our alert system. We need to ensure that drivers are able to receive necessary notifications regardless of any external factors such as network conditions or other technological issues. It is crucial that the alerts are sent out reliably to our drivers. Lastly, backend integration is important as it would allow for a seamless operation of our alert system. This would ensure that data handling and secure access to driver information for message handling would be efficient, and greatly increase the effectiveness of our alert system.

Alternatives - After identifying the characteristics of our alert system, we researched viable options and came up with three possible technologies that take all of these characteristics into consideration. They are Amazon SNS, Amazon Pinpoint, and Twilio. Each of these three

services provides different capabilities that make them suitable for our alert system. We delved deeper into our research of these three to identify how they compare in terms of strengths and weaknesses. First, Amazon SNS (Simple Notification Service). Amazon SNS is a cloud based messaging service within AWS, which is designed for push notifications and simple notifications via SMS and email. It contains a pay-per-message pricing model, which helps keep costs relatively low, especially for a project like this. Its strengths include real-time messaging, scalability, and low cost integration to AWS. It is regularly used for system alerts and mass notifications within larger organizations. Amazon Pinpoint is another service that expands upon SNS by allowing advanced messaging capabilities, including audience segmentation and delivers detailed analytics of these messages. It is mostly ideal for systems that need more personalized messaging. As it is more advanced, it also features a higher pay model rate. However, for a basic alert system we are attempting to implement, it may add unnecessary complexity. It is regularly used for marketing campaigns and personalized messaging. Lastly, Twilio is another service that provides APIs for SMS and email messages. Its strengths include real-time messaging and notifications, extensive scalability, and huge API support across a plethora of communication channels. It has a higher cost than SNS, which could impact the cost effectiveness characteristics of our alert system.

Analysis

Each API provides its own strengths and weaknesses with Amazon SNS giving the most straightforward approach but possibly lacking features necessary for individual and two way communication if we choose to implement it. Twilio and Amazon Pinpoint offer similar features at a similar price point however Amazon SNS and Pinpoint are exclusive to AWS leaving Twilio as the most viable option for a non AWS backend. Since we have decided to utilize Firebase as

our backend and given that there is extensive documentation for Twilio's implementation for a good price we have decided to choose Twilio for our SMS/Email communication API.

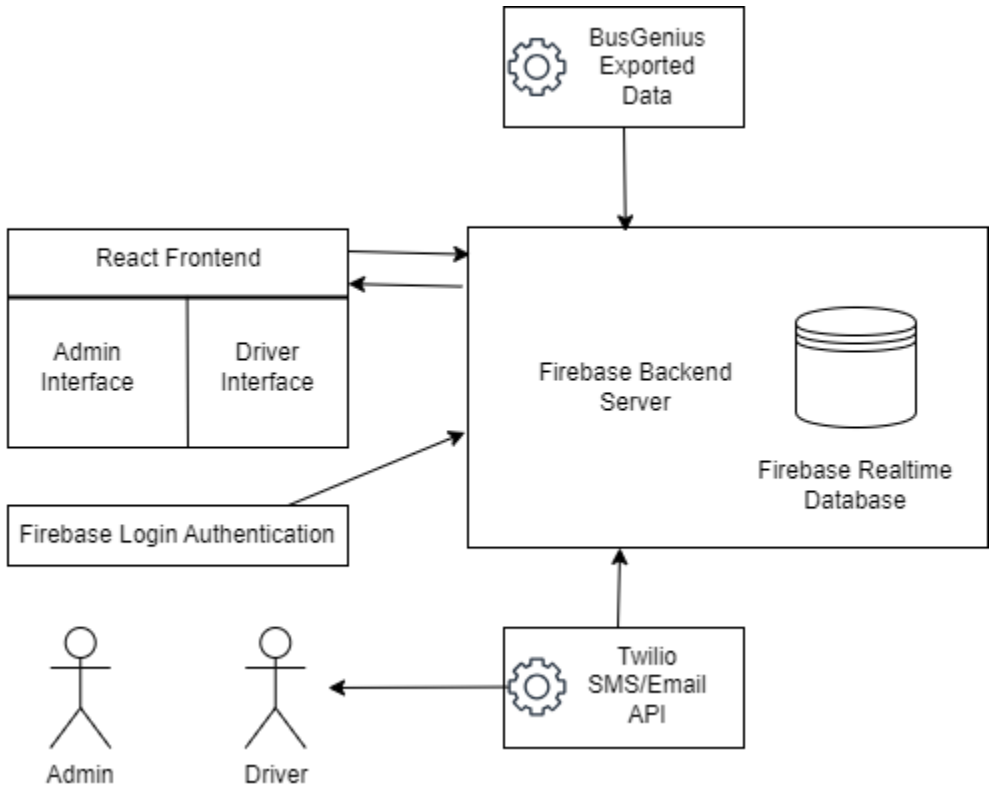
Characteristic	Amazon SNS	Amazon Pinpoint	Twilio
Real time messaging	5	5	5
Cost efficiency	4	3	3
Scalability	5	5	5
Ease of use	4	2	5
Reliability	4	3	5
Average	18.8	15.6	**19**

After evaluating each solution against the desired characteristics, we came to the conclusion that Twilio would be the best choice for our alert system. It provides a straightforward and scalable solution for our basic alert system, and leaves room for expansion by future owners of the project. It also seamlessly integrates with our chosen backend, Firebase. While cost may be an issue here, the other characteristics greatly outweigh the cost, and would allow for the one-way alert system we are aiming to achieve with this alert system.

Ultimately, with Twilio being chosen, we will be able to easily achieve the simple alert system for bus drivers we want to develop. It will also greatly speed up the development process of our project, as if the chosen technologies are all in sync with each other out of the box, we will have a much simpler time joining our technologies.

Technology Integration

All of the necessary parts detailed above will be integrated into a web application with the architecture shown in the diagram below:



The two main parts of our system will be the frontend interface that users will interact with, and the backend system that will hold all the system code. There will be two types of users with different privileges: Admin type users and Driver type users. Drivers will be the main users of the system and will be able to view the schedule and pick up shifts. Ideally, they will also be able to see their past clock-in and clock-out data from BusGenius. Admin users will have extra privileges, being able to edit shifts, assign drivers and buses manually, update the status of assets, and send messages to other users via SMS and email. Both of these user types will have a secure

login which will be authenticated through Firebase's login authentication feature. Once logged in, they can view and interact with the frontend interface built with React.

The frontend will interact with the Firebase backend server, being able to read and write data to and from the Firebase Realtime Database, a no-SQL database built into Firebase. This database in the backend will also gain access to data from BusGenius, likely through exported data, and perform mathematical functions on it to provide admin users with statistics about drivers' real clock-in and clock-out times. The Twilio API will also be integrated into the backend of the system and will allow administrators to send SMS and email messages, individually or en masse to drivers.

We believe all of these features and technologies, once implemented and integrated into one another, will create a cohesive system that will allow users to accomplish all the tasks they need to in order to keep NAU Shuttle Service's schedule well organized.

Conclusion

The technologies we've chosen and outlined in this document will allow all of the parts of our system to be well-integrated and work together effortlessly. This will allow the system to accomplish its primary goal effectively, which is to provide NAU Shuttle Services's drivers and management team with an easy and efficient way to create, modify, and update all of the information needed to keep campus transit running smoothly every single day. Having an easy-to-use schedule and asset management system like this will save time and provide more peace of mind to everyone involved in NAU Shuttle Services. With drivers able to quickly see what shift, route, and bus they will be taking over, shift changes can happen with less confusion

and complication. Our system will help NAU continue to have a smoothly operating and reliable transit system that will benefit everyone across campus – not just those working for NAU Shuttle Services. All members of the Northern Arizona Community who rely on transit services to get to their destination will benefit from our scheduling application.