Electronic Golf League Scheduler

Design Document

Team Number: sddec21-03

Client: Tina Prouty

Advisor: Mai Zheng

Team Email Address: sddec21-03@iastate.edu

Team Website: https://sddec21-03.sd.ece.iastate.edu

Executive Summary

Development Standards & Practices Used

- Agile development
- Responsive Web Development
- Continuous integration
- Code reuse
- Data privacy
- Software unit testing
- User interface testing

Summary of Requirements

- Dynamic scheduling system of golf hole assignments for both individuals and teams
- Account for absences and change in partners within schedule
- Notify users of hole assignments and tee times via push notification
- Store data related to what team played one another, starting hole assignment, and score
- Fast load times and no delays in API requests
- Minimal cost or no cost to client
- Low technical maintenance
- User friendly interface

Applicable Courses from Iowa State University Curriculum

List all Iowa State University courses whose contents were applicable to your project.

- Software Engineering 309: Software Development Practices
- Software Engineering 319: Construction of User Interfaces
- Software Engineering 329: Software Project Management
- Software Engineering 339: Software Architecture and Design
- Computer Science 363: Introduction to Database Management Systems

New Skills/Knowledge acquired that was not taught in courses

- ReactJS
- Node.js
- TailwindCSS
- DatoCMS
- Netifly
- Amazon Web Services
- Docker

Table of Contents

1 Introduction

1.1 Acknowledgement

We want to acknowledge Tina Prouty for the opportunity to work on this project. Thank you for your time and willingness to work with us. This project allowed us to work with industry-leading technology. Also, this project taught us to interact with a client and take an idea from conception to production.

Mai Zheng, we would like to thank you for your time and guidance during this project. We appreciate your technical expertise and feedback.

1.2 Problem and Project Statement

Problem Statement:

The women's golf league at Honey Creek course has gained more members than the current system can handle, an excel spreadsheet. Our client Tina Prouty would like to have an application that can enter individuals/team names and create hole assignments for them. If there is an absence by a team member, the absent golfer must be replaced, and the system must record the new golfer. If a team is missing altogether, then the application must reschedule the starting hole assignments. Each team member who golfed, the score they finished the course with, and their starting hole assignment from week to week must be recorded until the end of the golf league season. Currently, there are no free or low-cost options out there for a similar application. The available options do not have all the necessary features desired by the client.

Solution Approach:

The solution to this problem will be developing a responsive web application. The application will be used on mobile or desktop devices for ease of convenience for the client. Amazon Web Services will manage the application's backend since it provides a free tier and will be low cost to the client. The application will be deployed on Netfily, which will be low-cost to the user. The web application will have its registered domain specified by the client. This solution will provide a low setup cost and minimal monthly fees.

1.3 Operational Environment

Netfily will host the web application, and Amazon Web Services will manage the database. Upon deployment of the application, the user will have little to no interaction with these services, making the application maintenance-free. If the client needs to interact with either of these services, there will be instructional manuals for each service. Otherwise, she can consult the developer team.

1.4 Requirements

Functional Requirements

- Enter individuals and team members names
- Create an initial schedule for the hole assignments
- Account for absences by individuals/teams and able to enter replacements as needed
- Schedule will dynamically switch based on individual/team absences
- Keep track of golf scores, what team played, and the starting hole assignments throughout the duration of the league

Economic Requirements

- Low Cost
- Minimal monthly cost or no monthly cost

User Interface Requirements

- Simple to use
- Features are clearly defined
- Design is modern and professional

Version Control Requirements

- Each commit must contain a message with a detailed description for what is being pushed
- New features will be implemented on separate branches
- Feature branches being merged to master must go through a code review done by another team member

Software Testing Requirements

- User interface testing will be done with Selenium to ensure UI is working properly
- Mocking frameworks will be used to test API requests
- Continuous integration will be used to ensure code being push adheres to system specifications

Software Security Requirements

- Application will use modern security practices to prevent any known security flaws
- Developer team and client are the only ones to have access to root accounts for third party services

1.5 Intended Users and Uses

The end-user will be the person who is administering the golf league. The administrator will have full access to the application, which includes changing hole assignments and teams. The other users involved will be able to view this schedule and be notified of any schedule changes. These users will not have any of the privileges of the administrator.

1.6 Assumptions and Limitations

Assumptions:

- Information from previous technology will be used (excel document)
- Website will be accessed from mobile and desktop computers
- There will be only one user using the application, essentially an administrator
- The application will only be used for the women's golf league
- Application will only be used for one golf course (Honey Creek)
- Two players on each team, able to be changed based on member absences
- Automatic rescheduling

Limitations:

- Must follow standard design principles
- Users must be able to access the application from the golf course main website
- The application must be low maintenance to the client
- Budget constraints, the cost should be little to no expense to the user

1.7 Expected End Product and Deliverables

The expected end product will be a web application able to be accessed via a specified URL domain by the client. Amazon Web Services will host the database. The benefit of this is that it will allow the server to be up at all times, and the application will not need a dedicated server. The client can expect this application to be in production by the end of December 2021. Before then, there will be multiple prototypes developed to gain feedback from the client. Along with the application, the user can expect a manual on how to use the application and manage any third-party services. The user can expect to have a detailed document describing all login information used for third-party services.

2 Project Plan

2.1 Task Decomposition

The tasks below are broken down into sections consisting of planning, designing and implementation, testing, and delivery

Planning:

- Meet with client to discuss software requirements
- Decide on technology stack

Design & Implementation:

- Create wireframe and confirm design with client
- Setup development environments
- Develop the user interface of the web application
- Design the database
- Migrate the front-end and back-end

Testing:

- Perform unit testing and user interface testing
- Security testing and code refactor

Delivery:

- Migrate application to a production environment

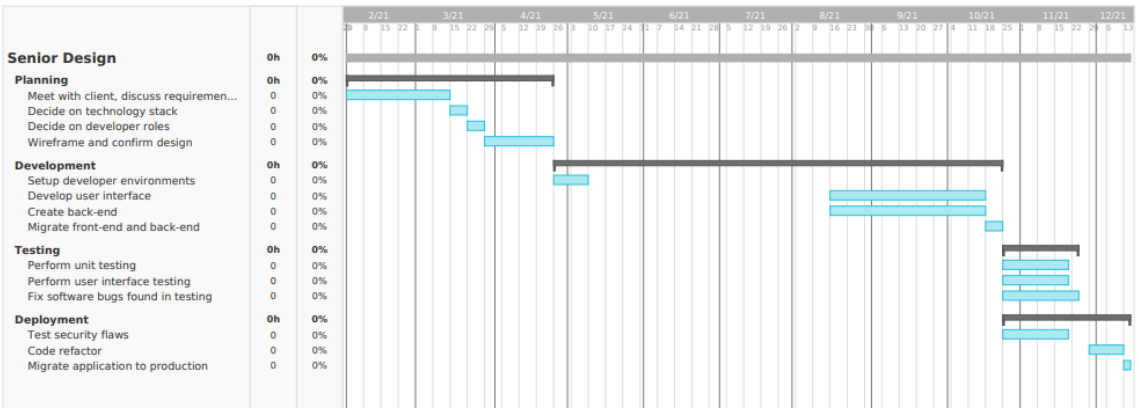
2.2 Risks And Risk Management/Mitigation

Task	Risk Description	Risk Level (1 is Lowest & 10 is the Highest)
Meet with client to discuss software requirements	Risk of not understanding the requirements as the user has described	2
Decide on technology stack	The selected tools do not work together or do not work correctly together	3
Create wireframe and confirm design with client	The client does not like the wireframe designs and takes more iterations than anticipated	3
Setup development environments	Docker containers take more of a learning curve and not able to get the tooling to work correctly	5
Develop the user interface of the web application	The CSS code is tough to create, resulting in the actually interface not looking like the	4
Migrate the front-end and back-end	The code libraries do not migrate well and makes it tougher for the developers to debug	1
Perform unit testing and user interface testing	The tests that are created fail and reveal more bugs in the code than originally anticipated	3
Security testing and code refactor	There are more security flaws and takes longer to refactor the code	2
Migrate application to a production environment	The way in which take the application to production proves to not work and must find a different way	2

2.3 Project Proposed Milestones, Metrics, and Evaluation Criteria

We will be using a Kanban software development process. This type of development process is defined as being incremental not iterative. To measure the progress of the project we will be utilizing a Kanban board.

2.4 Project Timeline/Schedule



2.5 Project Tracking Procedures

Our team is implementing a Trello board as our primary method of keeping track of the status of various tasks. Our hope in documenting our schedule in this way is to more effectively portray the status of the tasks we will be completing. Additionally, we are using a Slack group as our primary method of written communication as it allows for a faster workflow than email due to the ability for users to talk at the same time in various channels. We also plan to utilize GitHub to easily share code between various members, especially since some of us are not on campus. We plan to use git issues to easily show what needs to be done on the project, the ability to assign issues to different members will come in handy here as well.

2.6 Personnel Effort Requirements

Task	Task Description	Hours to Complete
Meet with client to discuss software requirements	Meeting with the client will allow us as a team to better understand the end users and the functionality of the application. Once these are solidified we will then create a requirements list to make sure that each requirement is being met.	3 Hours
Decide on technology stack	Upon finalizing the requirements we will begin to research what technology will best suit the clients needs. We will be using a comparison chart to do this.	5 Hours
Create wireframes	Based on the requirements and functionalities listed by the client we will begin to design wireframes to provide the client with an overview of the user interface. This process will have multiple iterations that will include designing and then confirming the design to clients expectations.	15 Hours
Setup development environments	To reduce the likelihood of compatibility issues between operating systems and likelihood we will be creating docker containers. Within this task the team will be setting up a CI/CD and a git environment to be used. The Git environment will follow a certain file structure that all developers must follow. Git will be using feature branches to merge to master and code review.	5 Hours
Design user interface	Taking the wireframes we will begin to put these designs into code. While creating these we will enforce strict styling guidelines for the colors and text used. React will allow for the reuse of components which reduces the amount of code to maintain.	30 Hours
Design the database	The team will design an ER diagram to ensure all developers understand the design. The database will be created using a variety of technologies including DynamoDB and node.js.	30 Hours

Task	Task Description	Hours to Complete
Migrate front-end and back-end	Upon completion of the front-end and back-end the team will begin to migrate the code. Ensuring that the application as a whole works together and is performing as expected.	10 Hours
Unit Testing & User Interface Testing	Each API request will be tested using Unit Testing. This will ensure that the backend performs as expected under certain circumstances. The user interface testing will be conducted to ensure that each part of the interface is usable and does not crash the system. Upon completion of unit and user interface testing the developers will fix the bugs found and then confirm that all tests are working as expected.	15 Hours
Security Testing and Code Refactor	The application will ensure that the user authentication is working correctly. There will be a refactor process to ensure all code is ready for production and all security measures are ensured	5 Hours
Migrate application to production and deliver application to client	The application will then be moved from the development environment to a production environment. The application will then be hosted via a third party service and all users will be able to access.	5 Hours

2.7 Other Resource Requirements

This project will be created using online resources available to the public. The only requirement is that we have access to the internet in order to use these resources.

2.8 Financial Requirements

There will be no financial requirements involved in the creation of this project. There may be a minimal monthly payment required for the use of AWS.

3 Design

3.1 Previous Work And Literature

There are a few competing applications in the same space as our product, with PlayPass and GolfSoftware being the most prevalent options. Both options are free or cheap and offer the ability to schedule a tournament or round robin league for a group of golfers. There are a few points that separate this project from these applications.

This project is built with a specific user set in mind, which allows the team to fine tune certain choices to deliver a more comprehensive product. The first point is the inclusion of automatic pairing shuffling and reassignment. Neither of the leading products offer this option, any changes in personnel would require a manual regeneration of the tournament.

This project also differentiates itself by tracking the weekly scores, winners, and standings of each round in the league. GolfSoftware tracks which golfer in a pairing wins, but not scores or any side competitions such as closest to the pin.

Finally, this project lets groups compete as a team. Neither of the competing products offer this capability.

3.2 Design Thinking

There were a large number of ways to accomplish this task, being that there was no prior application to build off of or integrate with. While helpful in a sense, it posed a different challenge of narrowing down our options to a select few that were simpler to analyse and compare. To start, we selected a few requirements that would play a large part in the decision making process, such as:

- Cost needs to be kept low.
- Project needs to be easily extensible, so that the team can add new features if opportunities for improvement arise.
- The application must be user friendly for a non-technical audience.

With those requirements laid out, we were able to throw out some options such as platforms that have a monthly cost for hosting and platforms without a free tier. It was helpful that the amount of traffic expected to flow through the application is low, meaning that it is likely that it will stay within the free tier of some platforms.

Possible development technologies for the project were compared and selected based on the familiarity of the tool/language/platform of the team, simplicity, and cost. Cost was the dominant feature in the decision making process.

3.3 Proposed Design

IEEE defines multiple software engineering standards. Of those standards there are ones that are more applicable to our project than others. The types of standards we defined as being applicable to our project relating to customer interaction., product development, and testing. IEEE has a standard called *'Standard for Application and Management of the Systems Engineering Process'*. In this standard, it is described how to meet the customers requirements, needs and constraints.

The next standard that is applicable to our project is the *'Software Life Cycle Processes'*, this engineering standard describes the framework for software lifecycle management. This standard goes into detail describing how to develop, maintain, and operate the software of our project. Unit testing is going to be largely incorporated into our project. The standard that relates to unit testing is *'Standard for Software Unit Testing'*, in this standard it describes best practices and approaches to implement unit testing for our project.

There are a few software development standards that played a large role in the development and evaluation of the various application designs presented below.

API - Microservices/Serverless - <u>Selected</u>

Each microservice will run on an AWS Lambda instance to provide low-latency operation, even during cold start situations. AWS DynamoDB will be used as the main database for the application. To facilitate this, the team will be utilizing the Serverless Framework with NodeJS to manage infrastructure as code and reduce the complexity of deploying to AWS and track all changes to infrastructure inside of git for version control.

Microservices will use Gitlab CI/CD for Continuous Integration and Deployment to AWS.

Each service will contain a single domain area. By doing so, each service is very loosely coupled to other services, allowing for rapid iteration between services without causing breaking changes elsewhere. Listed below are the services that the team will be implementing.

Services:

- Auth: Handles authentication and authorization for users.
    - Register
    - Login
    - Logout
    - Reset Password
- Users: Tracks individual details and performance for a golfer throughout the season.
    - Personal details such as email, name, etc.
    - Average Score
    - Season standing
    - Contest wins
- Teams: Groups of users that make up individual teams. Handles team shuffling and result tracking.
    - Season standing
    - Current members
    - Contest wins
- Outings: Handles scheduling and team pairings for each golf outing.
    - Schedule
        - Team
        - Starting Hole

- Leagues: Overall league standings and organization.
    - Team Leaderboard
    - Individual Score Leaderboard
    - Contest Winners

Pros:

- Utilizing a serverless architecture with microservices aids in keeping costs low by only running the application when absolutely necessary.
- Microservice applications are extensible by design. As the project progresses, the team will have the ability to add features if requested without the need to alter other aspects of the project.
- HTTP Request validation handled automatically by AWS.
- Simpler testing through mocking other services.
- Auto-provisioning of resources from AWS.

Cons:

- Local development requires a lot of setup and can be problematic if not configured properly.
- Steeper learning curve for communication between services.
- Continuous Integration and Deployment is complicated.

API - Monolithic - <u>Not Selected</u>

The team will implement a layered architecture. By doing so, each layer will be loosely coupled with the other layers of the application, allowing for easier testing of each layer by using mocks when interacting with outside layers.

The team will "code to interfaces" to allow the logic of each layer to be changed without other layers requiring knowledge of the changes. The interface that outside layers use will remain the same, while the logic underneath is changed.

The application will be made up of four layers:

- External: Any execution details for external services such as email platforms are held in this layer.
- Persistence: All database interactions will be handled here.
    - Repositories
    - Database configuration
        - Migrations
- Core: Domain logic for the application is stored here.
    - Services
- API: This layer exposes a REST API for the frontend to communicate with.
    - HTTP Endpoints

Layers are only dependent on the layer directly beneath themselves. The API interacts with the Core layer, but has no connection to the Persistent or External layers. The Core layer interacts with both the Persistence and External layers. The Persistence and External layers do not interact with any other code, they only expose interfaces in which to be interacted with by the Core layer.

Hosting of the Monolithic application will use the Heroku free tier. This provider provides git-connected Continuous Integration and Deployment and a PostgreSQL database.

Pros:

- Smaller learning curve for development.
- Continuous Integration and Deployment is simple and triggered by pushing to the main branch on a git repository.

Cons:

- Heroku cold-start time is very slow and could impact user satisfaction.
- Much larger code-base due to using an interface for every service and repository.
- HTTP Request Validation is done manually by the developer.

Application UI - Web Application - Selected

A web application provides a cross platform and device-agnostic user interface for the project. The team will be using ReactJS as the application framework to aid in development, with MaterialUI as the design foundation to ensure that the application is visually appealing. React Hooks will be preferred over class based components during development.

The project will be broken up into individual features, with a main component to handle routing and application context and state management. Each feature will be structured similarly to the overall application, with a main component and sub-directories for internal components. Each directory will hold these files:

- ComponentName.js - Markup and state management for the component.
- Requests.js - All API requests used in the component.

The web application will use Netlify for free hosting and git-driven Continuous Integration and Deployment. The domain will be managed by Route53 on AWS.

Pros:

- No need for installation or updating once the application is deployed.
- Works with all devices.
- Free hosting.

Cons:

- An internet connection is required to use the application.

3.4 Technology Considerations

AWS was chosen over Azure and Google Cloud Platform due to the very generous free tier for their services. AWS costs for the application are expected to be less than one dollar a month.

DynamoDB was chosen over other SQL and NoSQL database options due to the AWS free tier. Its low latency for reads and writes also makes it ideal for serverless applications where runtime is what monthly costs are based on.

NodeJS was chosen over other Lambda runtimes due to its speed during cold-starts, which we expect to be the majority of interactions with the API. Jest will be used for unit testing.

Netlify hosting was chosen over AWS Cloudfront for the web application due to the simplicity of deployment and domain configuration.

3.5 Design Analysis

The design plan outlined in this document is meant to be a starting point. We believe that this design will change iteratively throughout the development of the project as new needs are discovered or shortcomings appear.

When one of the previously specified events occurs, the team will first evaluate which area of the current design needs to be addressed. Once identified, themteam will then decide whether the current implementation plan can be altered to fit the new requirements, or if a new plan needs to be created.

In the event that the current plan is insufficient and needs to be replaced, the following process will be followed to develop and refine any new solutions.

1. Brainstorm with a wide field of view to identify and select any possible options.
2. Narrow down the options found in the previous step by digging deeper into each one. The goal will be to narrow the option count to 3 or less.
3. Create an implementation plan for the remaining options.
4. Evaluate the new implementation plans and select the best one, keeping the following qualities in mind:
    - Cost.
    - Complexity.
    - Addition of new technology or platforms.
    - User friendliness (if applicable).

3.6 Development Process

The team will be utilizing a Kanban development process for organizing and selecting tasks for this project. A Kanban process was selected for a couple of reasons. Firstly, it allows the team to add more items and features to the backlog as the project progresses. This will help with prioritization and let the team extend the feature set of the end product without overreaching. Secondly, due to Kanban being a looser form of Agile development than the strict sprint-based method of Scrum, the teams varying schedules and workloads during the course of each semester can be accounted for more efficiently without seemingly bringing progress to a halt during exam weeks.

3.7 Design Plan

Our plan is to separate the backend and frontend into a web application and REST API.

Web Application

The user interface for the project will be a Single Page Application written in React. Each page of the application will be contained within its own directory, with all components pertaining to the page held within that directory. In doing so, adding new pages to the application is trivial and will not affect the operation of other pages.

Interface components will be split by functionality on a case by case basis. When a component seems to gain more functionality than originally intended, the developer will break it up into smaller components for readability.

Hosting and deployment will be set up in a free Netlify account to automatically deploy when a commit is pushed to the main branch of the git repository.

REST API

The backend for the project will be written in NodeJS, with each endpoint (GET, POST, UPDATE, DELETE) being hosted on it's own AWS Lambda instance. Prior to reaching its destination lambda, a request will pass through an AWS API Gateway.

The API Gateway will do the following:

- Validate the request body and respond with an error message if it is not properly formatted.
- Attach the requestor's authorization credentials so the endpoint can return user-specific data.
- Pass the request to the correct lambda endpoint.

Each service, shown in Figure 1 as a box of nodes, will have its own table in the DynamoDB database schema. DynamoDB is distributed by design, so there is no need to have multiple instances running.

AWS API Gateway, Lambda, and DynamoDB will run within their respective free tiers. The only exception is that API Gateway will have a negligible cost (< $0.50) after the first year of use.

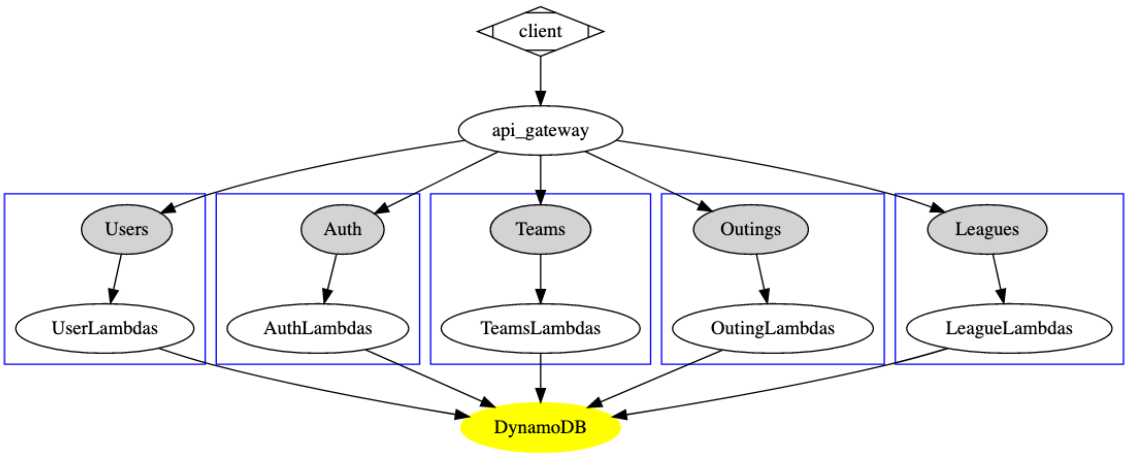


Fig. 1 - High level backend architecture

4 Testing

4.1 Unit Testing

Unit testing will be executed within each microservice in isolation. The team will be following Test Driven Development (TDD) to help drive the development cycle. TDD will help by allowing for simpler development with the Serverless Framework, due to the relative complexity of working with services that must be mocked to run locally.

Each service will require tests for all business logic and database interactions. All tests will be executed within the continuous integration pipeline when a merge request is created. If any tests fail, that pull request will need to be remediated and resubmitted when the branch is in working order.

4.2 Interface Testing

Testing multiple services will be done in the form of HTTP requests to the application that mimic how it will be used in production. An endpoint will commonly need to communicate with multiple services to accomplish the required task, by treating interface testing and integration testing as the same step, a greater level of test coverage can be achieved with less time spent on the development of the test suite.

Interface/integration testing will also be executed in the continuous integration pipeline after the completion of unit testing. To execute these tests, a script will be written that sends a curl request reflecting each test and its expected outcome. If any tests fail, the situation will be handled in a similar manner to unit test failures.

4.3 Acceptance Testing

To ensure that the team is delivering features and functionality that is in line with the client's goals and objectives, each feature branch will be pushed to a staging environment before being deployed to a production environment. This staging environment will mimic the production environment and will be available to the client.

When a new feature is ready for review, the team will notify the client and request that they take some time and test out the new feature and overall "flow" of the application. Upon the completion of the client's review, the team will address any issues or changes brought to attention. If none arise, or it is deemed that the feature is at a stage that it can be promoted, the staging environment will be pushed to production.

4.4 Results

Our initial testing results are centered around the Acceptance Testing area of the testing plan. Multiple client meetings have allowed the team to develop and present to the client a set of User Interface Mockups that will be the starting point for the development of the Web Application. The client has provided feedback pertaining to these mockups, allowing the team to iterate on our design and start the development phase of the project with a firm understanding of what the client is expecting of the application.

5 Implementation

Our implementation plan for next semester is to separate the frontend and backend development and work to build them out at the same time. We will designate team members to either frontend or backend, however each team member will still have an understanding of each project layer and will be able to contribute anywhere they are needed. After several meetings with our adviser, we should have a really good understanding of what our client wants and what is most important, and this will help give us a good structure to build off of. We will also be working hard this semester to finalize our design plan, which will give us confidence in beginning the actual buildout. We may start setting up our environments towards the end of this semester to make sure each team member feels comfortable writing code at each layer and there are no hiccups getting started next semester.

6 Closing Material

6.1 Conclusion

We have met with the client to discuss the goals for the project. Since the user experience plays a large role in the success of the project, the team has been designing wireframes to display how the application will appear. The stack of languages and technologies has been decided upon. Also the architecture of the project has been agreed upon and the design strategy. The driving force behind these decisions was free to the client and the ease of development. Over the semester we will be meeting with the client to discuss the wireframes and being sure all requirements are being met.

6.2 References

Important Engineering Software/Hardware Design Standards. Software/Hardware Design Standards. (n.d.). http://users.encs.concordia.ca/~ecewebdv/EDS/Software/std_list.htm.

6.3 Appendices

Manuals will be created for the client on how to use AWS if any problems ever occur.