Technical Document (PSPMMFDW)

Jack Anderson

Cam Ball

Daniel Miller

Yongfeng Ye

Oliver Norris

Capstone Project Spring, 2024

The University of West Florida

2/24/2024

CIS 4592 Capstone

E. L. Fridge

Abstract

The Pipeline Sports Performance Member Management and Facility Data Warehouse web application is focused on creating software that improves the workflow of coaches and provides useful information to members. This document details the technical side of the application, including security, testing, use cases, our development process, requirements, development, and deployment. It is important to mention that our team's approach is slightly more long sighted due to the expected future of the application being a out-in-the-world piece of software that will need to be maintained.

Some portions of the document reference other previous documents that we have submitted. Also, there are links to some important resources that we will be using at the end of the document.

Table of Contents

1 Development Process

We have chosen to use the scrum model for our development process. This entails the use of scrum roles, sprints, and continuous delivery of features. Requirements and functionalities are decided before the start of the development, and implementation of these tasks are separated into different sprints. Each sprint is relatively three weeks long, and the retrospection will be held at the last week of that sprint. The team is expected to finish the tasks that are assigned by the deadline of the sprint.

Compared to other development processes, scrum has the advantage of quick and efficient project delivery. Large projects are divided into easily manageable sprints, and the programs are tested during the sprints. Short sprints period adopt changes very well, make it flexible enough when there are changes happen to the project plan. Compared to the traditional scrum development, we chose not to have the daily scrum because it requires a shared schedule for every team member. We also didn't adopt the use of sprint review, or rather, we combined it with the Sprint retrospective.

2 Requirements Elicitation Strategy

The strategy used to elicit requirements for this project was and is fairly simple. We were able to draw on experience from within the group to create requirements, as well as going to the owner of Pipeline Sports Performance and getting his wish list for the application. Through multiple meetings with the customer and internal team discussions we were able to come up with our requirements.

The creation of epics was a straightforward process once we had our requirements. The epics were generated on cards on our Trello board as we talked through the requirements for the project as well as the timeline. From there, User Stories were created by breaking apart the epics into more manageable and concise entities. We did this by examining the high level

requirements from our unique “lead perspective” and thinking about what the user's interaction with a specific feature would entail. From there, as we selected User Stories or features for the current sprint, Use Cases in the form of more detailed and technical task lists were the natural evolution of our Trello cards. The cards in our Trello board will serve as the main means of documenting the process of feature generation and shall be referenced throughout the evolution of the project for evaluation purposes.

Our project development timeline has lined up nicely with the requirements generation thus far. Our initial phase of generation took place before the beginning of the first sprint which allowed us to spend more of our sprint time writing code. By generating technical task lists at the beginning of each sprint we maintain efficiency while not getting stuck on methodologies or structures that are no longer feasible due to unexpected ad D aptations arising from technical challenges. Further, we are able to better adapt to changes in our development cycle as we always have the ability to add more tasks to a particular sprint.

3 Use Cases

For each of the different use cases there will be a concise description of what the user story that it is generated from entails, followed by a control flow with more technical details associated with use cases. After the control flow, the priority level of the use case will be discussed.

3.1 Use Case 1

A user has arrived at the application and is going to login to their account.

1. The application will check the login status of the user
2. If the user is not currently logged in, they will be redirected to /google
3. That will render the OAuth consent screen where the user enters their information
4. OAuth will then talk to our Passport strategy to get the user profile information

5. The application then checks the returned profile information against the database and sets the user as logged into the application
6. The user will then be redirected to the url that they were originally looking to access
7. This use case is ranked as a must have with a high priority level on our Trello Board. If this use case is not implemented our application will not function correctly.

3.2 Use Case 2

A coach wants to assign a player a set of workouts and drills, that then is shown on the players profile for them to follow along.

1. A coach who is logged into the application will open the dropdown list of players that they are connected to
2. From the list of players the coach will make a selection and be redirected to a page where they have many options for actions associated with that player
3. The coach will select the assign workout option
4. The application will then go grab a list of workout options from the database
5. When the object with workouts comes back, it will be rendered as a list that has checkboxes next to each option, this will all be done using responsive html in handlebars
6. Once the coach has selected all of the options they want to assign, a submit button at the bottom of the page will take in all of the selected workouts and create a new workout object that is connected to the player's id
7. When the player next logs into their account, the active workouts that are attached to their id will be rendered on their page using responsive html in handlebars

This use case is ranked as should have, with a medium priority on our Trello board. This feature is not integral to the operation of the app at deployment time, but would be nice to have as a feature at that juncture.

3.3 Use Case 3

A player who is logged into their account on the application wants to view the information and statistics associated with their account.

1. The logged in player will be directed to the "/" route which will serve as their home or jumping off page
2. On that home page, the player will be able to view the data associated with their account
3. The application will fetch the data from the database that has a playerID matching that of the logged in player account and return that in the res object
4. A middleware function will then parse the returned object and pull out the data needed to render the page
5. The data will be passed into the handlebars template where it is then rendered via responsive html

This use case is ranked as must have, with a high priority on our Trello board. This feature is a key part of the application that is near the top of the requirements list. The application would not fulfill its purpose without this feature.

3.4 Use Case 4

A parent who has a player, wants to get notification of events that their player is a participant in.

1. Given that a parent has linked their account to their player's account, they will be able to receive an email notification of an event that their player is associated with
2. When a new calendar event is created by a coach or team administrator, the players who will be participating in that event will be added to the event
3. The application will then automatically check the player objects whose playerID matches the playerIDs on the calendar event
4. When the playerID is confirmed to match, the application will then check if that player has a parentID associated with it

5. If there is a parentID, the application will use the Gmail API to automatically send an email with the details of the event to the email address associated with the parentID
6. If no parentID is found to be associated with the player object, an email will be instead sent to the email address associated with the player object

This use case is ranked as could have, and a medium priority on our Trello board. At the time of this document creation, we have not yet started on this feature so the control flow is more generic than some of the other use cases. We would like to implement this feature as it would elevate our application and add a high level of functionality to the application.

4 Development Platform

We will be implementing the use of the Typescript superset of Javascript as our primary language. We chose this language because of its readability, security, and its ability to prevent bugs prior to the code being shipped. As we have members in our team with no experience in the language, let alone the superset, it will allow for much easier understanding for the developers to grasp what is being written and what the code is doing. It also allows developers to use modern language features while shipping code for multiple different versions of Node.js. We chose to use Virtual Studio Code as our IDE because of how well it integrates with typescript as well. We will be using the following APIs' during the development of our web application:

- Passportjs
- Gmail API

We chose the Passportjs API because of our Team Lead and Coding Lead's previous experience working with the API and have found implementation of the Google OAuth2.0 authentication to be very simple with this API. It follows Node.js' convention of callbacks and closures, and is compatible with other Node.js versions, making it very easy to integrate into our system as we will be using Node.js as our runtime environment.

We chose to work with the Gmail API because of the many features we will need to implement into our web application. The Gmail API allows for the following applications:

- Read-only mail extraction, indexing, and backup
- Automated or programmatic message sending
- Email account migration
- Email organization including filtering and sorting of messages
- Standardization of email signatures across an organization

We will need automated message sending when notifying parents and players about upcoming events. We will need email account migration when creating new accounts for all users. We will need standardized signatures when sending these notifications to users to maintain a level of professionalism that we see is fit for the facility when sending the automated messages. [1]

As for code management we will be using a spreadsheet as well as github to document and clarify exactly what is being done, what needs to be, and what issues arise from our work. The spreadsheet requires the github commit sha, the completion date, the branch of use, a description of the task, issues/concerns, the main developer, and any contributors. This spreadsheet will be updated weekly by the coding lead, if not done after completion of the task by the main developer.

5 Deployment Platform

We will be using express js to route our deployment. The server we will be using is an EC2 instance that uses express js as its routing framework. When the application is deployed for real world use, any requests that hit our url will be routed with the help of express js through our application to our database and then back to the client.

Our choice of an EC2 instance hosted by AWS was driven by its unique intersection of ease of use and performance capacity. AWS will allow us to connect our GitHub repository for ease of integration of features and code. Once we have our code connected to AWS, we will

configure the database connection to be fully integrated with the deployed application. There are many other possible platforms that we could deploy with, such as Heroku, but given our requirements and future aspirations, AWS makes the most sense.

We will be storing all of our user data in firestore. Firestore is an enterprise-level NoSQL database compatible with complex data models that need more scalability. We chose firestore because of its scalability compared to firebase as well as its offline synchronization for web applications. Our intentions with this web app are not just for the course but to propose for a real company as previously mentioned in other forms of documentations, so we do not want to limit our customer in the long term, so the scalability was a massive plus for us.

6 Functional and Security Testing

6.1 Unit Testing:

Using the jest framework unit tests will be made for critical functions such as those involving authentication, user input, data retrieval, database connection, and API integration. These listed functions are all critical for the main functionality of the application; if they are not tested, we run the risk of inadvertently creating bugs.

6.2 Automated/Regression Testing:

We will use the testing framework jest that runs automatically by Github everytime there is a commit to main. This will test the routes in our application and the database retrieval responses to ensure that nothing other than the current commit is affected.

6.3 Performance/Usability:

We will conduct real-world end user runs with the users interacting with the application. And providing a follow up survey after these runs we can look at user feedback from the survey and analyze the actions performed by the user during these test cases. It will also be possible to

use the developer console to track how long webpages take to load, this way we can see if any of our pages take an inordinate amount of time to return a response.

6.4 User Acceptance Testing:

To determine if this application meets the standard to be implemented for our intended users (Pipeline Performance) we will determine the user's minimum expectations for things like usability and functional business goals through an interview. This will help us to compare user requirements to our finished product.

6.5 Security Threat Identification and Testing Strategy

This is our security requirements discovery process and is accomplished through a combination of threat modeling and risk assessments. Our threat modeling will be conducted using the Microsoft Threat Modeling Tool [3] to identify threats specific to the project's technology stack. The risk assessment process is used to analyze the identified security threats early in the development cycle and provide mitigation strategies. Node JS has various modules and libraries used for security testing middleware [4]. Security planning and testing will be documented within the PSPMMDFW Security Requirements & Testing Document.

6.6 PSPMMDFW Security Requirements & Testing Document

This will be our primary security documentation tool that will cover the following topics

- Security Requirements: Conditions to ensure system and data protection against threats, including authentication and authorization protocols.
- Misuse Cases: Scenarios outlining potential system abuses or attacks to guide the development of preventative measures.
- Data Protection Methods/Encryption Standards: Techniques like encryption and hashing to secure data against unauthorized access and breaches. Use of strong

encryption algorithms for data at rest and in transit, along with secure key management practices.

- Secure Coding Practices: Guidelines for writing code that is resilient to attacks, emphasizing input validation, secret management, and secure API use. [2]
- Access Control Measures: Strategies for managing resource access through authentication, RBAC, and the principle of least privilege.
- Security Testing Strategy: Threat modeling and risk assessment methodology. This will also include the list of node JS modules and library middleware used for security testing.

The main objective of security requirements and testing is to force adherence to the basic principles of secure software development. We will use the different listed measures in conjunction with the other types of testing to ensure that we do not violate any principles of security and create issues for our user base.

References

[1] Gmail API Overview | Google for Developers

[2] Threat identification

[3] Guide to Security in Node.js

[4] Microsoft Security Development Lifecycle