SANTA CLARA UNIVERSITY

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Date: August 20, 2020

George Shappell
Kunal Pandey
Ryan Lund

COEN 174 SOFTWARE ENGINEERING

Iron Engineer Final Report

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS OF THE COEN 174 COURSE FOR THE DEGREE OF

BACHELOR OF SCIENCE IN COMPUTER ENGINEERING

_____

Thesis Advisor

_____

Thesis Advisor

_____

Department Chair

_____

Department Chair

Iron Engineer Final Report

by

George Shappell
Kunal Pandey
Ryan Lund

Submitted in partial fulfillment of the requirements
for the degree of
Bachelor of Science in Computer Engineering
School of Engineering
Santa Clara University

Santa Clara, California
August 20, 2020

Iron Engineer Final Report

George Shappell
Kunal Pandey
Ryan Lund

Department of Computer Science and Engineering
Santa Clara University
August 20, 2020

ABSTRACT

The Iron Bronco event is a cooperative triathlon where teams of students are tasked to run 26.2 miles, bike 112 miles, and swim 2.4 miles over a span of two weeks. The current system used to store participant team information and progress is done by hand. Our system, Iron Engineer, aims to move Iron Bronco onto a web application in order to reduce administrative effort. The website will provide a platform for participants to find other team members and tally together team progress. Management of the triathlon on a web application will allow for easier storage and disbursement of information related to Iron Bronco.

Table of Contents

List of Figures

Chapter 1

Introduction

The new Dean of Engineering at Santa Clara University wants to encourage faculty, staff, and students to participate in the Iron Engineer triathlon, an event that spans two weeks in April. Participants are tasked to swim 2.4 miles, bike 112 miles, and run 26.2 miles over the course of the competition. Participants are allowed to create a group of up to three people; the goals of the triathlon can be spread between each team member. Individuals are given the option to participate alone or join a team of fellow Broncos. In order to work together to fulfill the requirements for each activity, participants will need a way to record their progress where each team member's individual progress is tallied together with the team's progress. This system should be easily accessible for all students, faculty and staff. Triathlons and other events that encourage friendly competition among peers helps foster the community of Santa Clara and its campus.

In the current solution, participants fill out a log on paper and turn it in once they have completed the race. Race managers then manually review completion status and date for all participants before storing this information in one document. This not only creates a large amount of manual work for campus recreation representatives and race participants, but it also prevents users from seeing their progress in real time during the time frame of the event. As a result, the current solution is not sufficient because its manual nature is cumbersome to maintain, and the lack of access to information hinders the competition experience. The next iteration of this system must allow for easier access of the competition's information and lessen the burden of information input and upkeep on competition management.

The solution we are proposing is a web application that will allow Iron Engineer participants to register for the race and to track their progress online. Users looking to build a team will be able to search for teammates on the application. Once their team is formed, they will also be able to see the team's progress on each event. The proposed system will lessen the burden on race administrators by giving them the ability to see who has registered, their team and team members, and race completion status. By making the registration and team building process easier for prospective participants, the solution is expected to increase participation in the Iron Engineer competition among engineering students and faculty within Santa Clara University.

Chapter 2

Requirements

Functional Requirements		Key	
Register participants for event, solo or with team			Critical
Allow for searching for team members to create team			Recommended
Log activity completion for all 3 events in triatholon			Suggested
Store activity log progress			
Display activty log/event progress to all participants			
Non-Functional Requirements		**Key**	
Security of participant information			Critical
Mobile app support			Recommended
Leaderboard System			Suggested
Authentication (SCU login)			
Maintain/print out activity log to turn in for Iron Bronco			
Design Constraints		**Key**	
Must run on Chrome and Firefox web browsers			Critical
Must run on DC machines (Linux)			Recommended
			Suggested

Figure 2.1: Requirements Table

Chapter 3

Use Cases

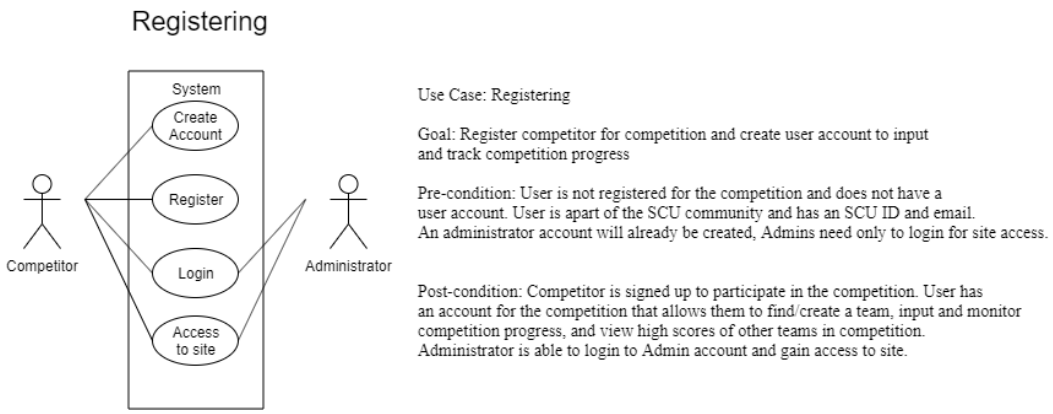

Figure 3.1: Account Registration and Login

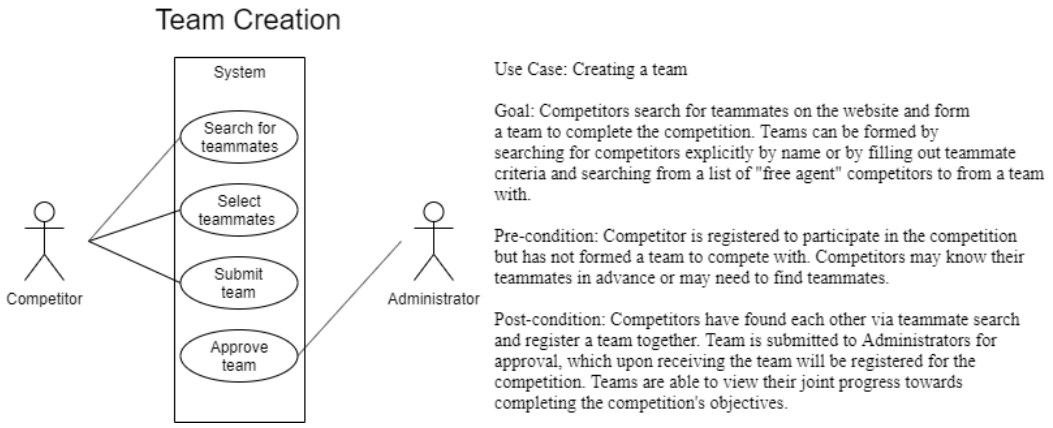

Figure 3.2: Team Creation and Approval

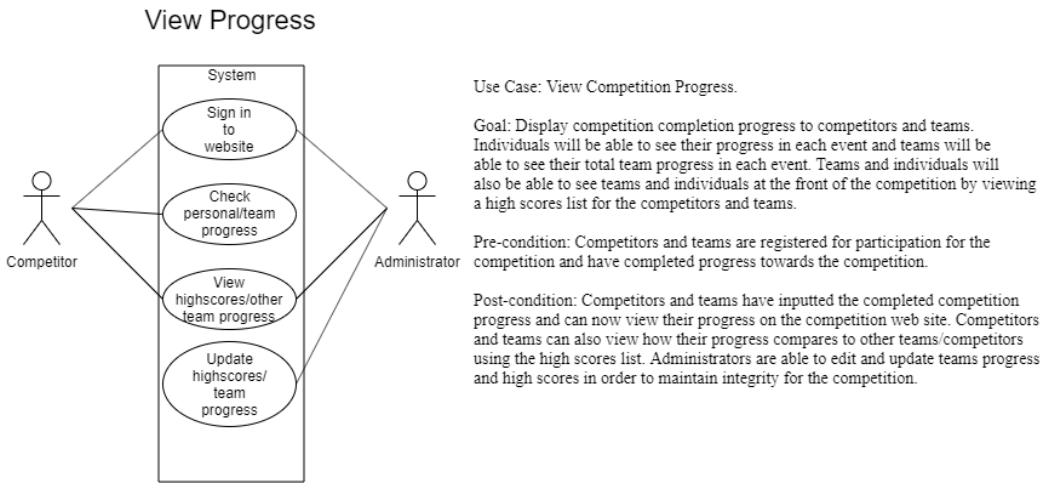

Figure 3.3: View and Update Team Progress

Chapter 4

Activity Diagrams

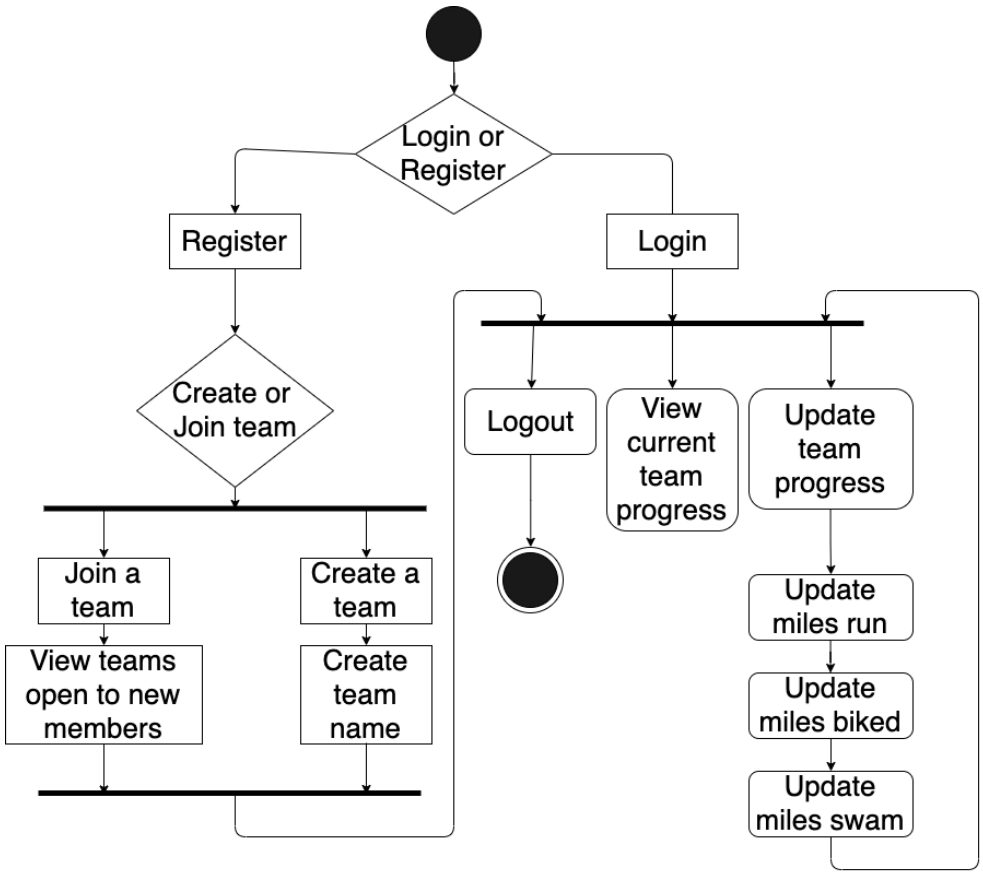

Figure 4.1: User Flowchart

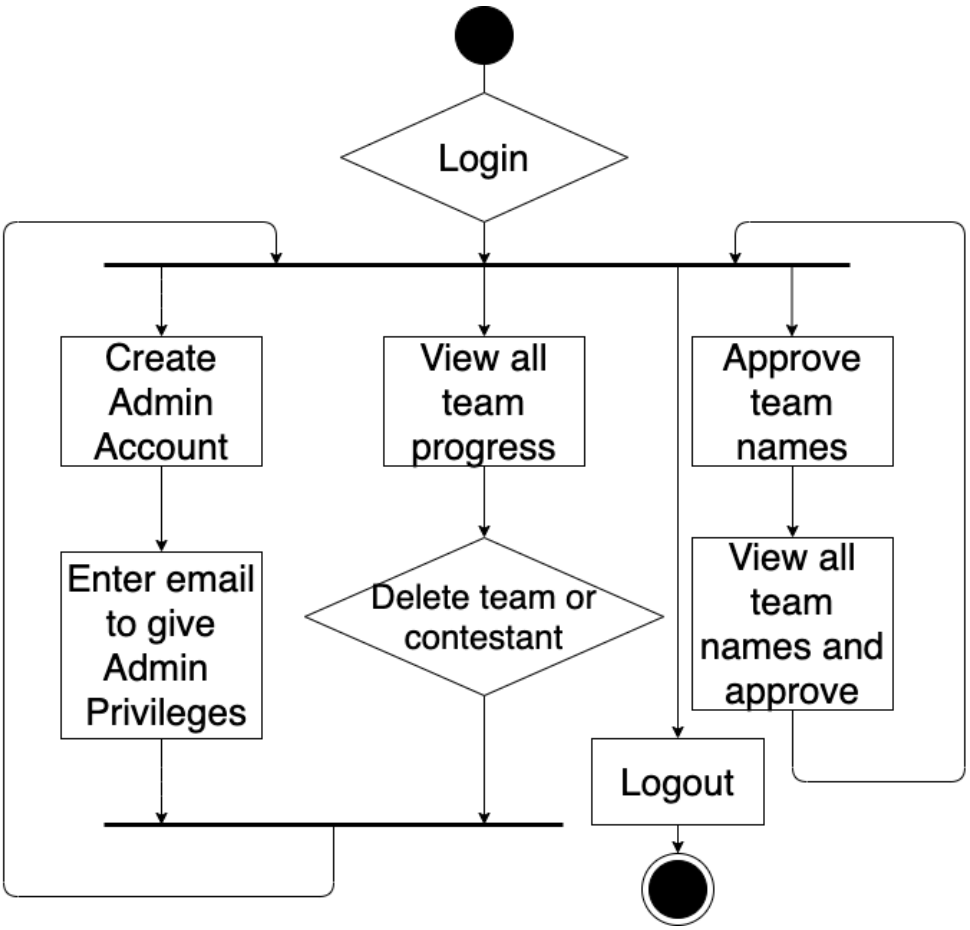

Figure 4.2: Admin flowchart

Chapter 5

Technologies Used and Architectural Diagram

5.1 Back-end Software

- AWS EC2
  - Amazon Web Services' (AWS) EC2 (Elastic Compute Cloud) is a cloud computing platform that allows for rental of virtual computers for services.
- AWS RDS
  - RDS (Relational Database Service) is a relational database hosting service provided by AWS.
- Java and Spring Framework
  - Java is a high level object oriented programming language used for application development. Spring is a Java framework that makes application development easier and more manageable for enterprise.

5.2 Front-end Software

- JSP and HTML
  - Java Server Pages (JSP) is a collection of technologies used in creating dynamic web pages using information supplied by the back-end. For this project, HyperText Markup Language (HTML) is used in conjunction with JSP to provide web pages.
- CSS
  - Cascading Style Sheets (CSS) is a styling language used to describe how the elements of a HTML page are to be presented on a web page. CSS is used to style web pages and present information in an attractive manner.

- Google Login API
  - Google's Login API allows for users to sign into a service by using their google accounts. This allows for user verification without the need for storing passwords in the local system.

5.3 Architectural Diagram

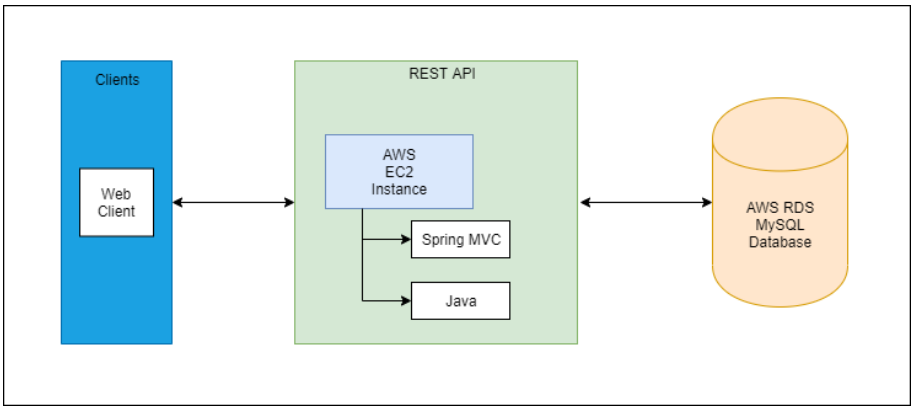

Figure 5.1: Technical Architecture Diagram

Chapter 6

Design Rationale

The Iron Engineer competition can be broken down into a simple model. In this model, there are teams who are trying to complete the race, and registered participants who are trying to find teams. To take advantage of this simple model, we have chosen to use the Model View Controller (MVC) framework. Given the need for database interaction, the support for the MVC framework via Spring, and our team's experience with the language, Java seemed a natural choice. Since the Design Center's Linux machines do not have support for the spring framework, we needed somewhere where we could host our software that has compatibility with these technologies, preferably for free. As a result, we chose to use an AWS EC2 micro instance and an AWS RDS database.

Chapter 7

Description of System Implemented

The system implemented for the Iron Engineer competition is a web-based application that can be accessed via web browsers such as Google Chrome or Mozilla Firefox. This accomplishes both design constraint presented with the project, that it be accessible in the Design Center (DC) and web-based.

Our system meets all functional requirements laid out in the initial design document; all competitors are able to compete solo or with a team, can log their distances completed, search for teams to join, as well as view progress towards completing the competition.

The implemented system met many of the recommended non-functional requirements described in the design document. Security and authentication of competitor information is handled with Google's login API. However, our system in its current state does not provide a leader-board system for competitors, and there is no functionality for printing out activity logs to turn in to the Iron Bronco competition. These features are highly suggested for a future iteration of the Iron Engineer system, as well as the addition of a mobile application for the system.

The CSS styling of the website fits the school colors of Santa Clara University, white and red.

Administrators of the system have full access to the website and are able to monitor competition progress and individual competitors. In order to maintain a fair competition, administrators are able to remove teams and competitors in the case of dishonest distance logging. With the exception of high-scores and leader-boards, the implemented system satisfies all use cases presented in the initial design document.

Chapter 8

Test Results

Step	Test Step	Test Data	Expected Results
1	Create account for participant	Username and password for account	Account created and registered for competiton
2	Create team on website		Team created and registered for competition
3	Join team using team search	Team name to search for	User has joined a team for the competiton
4	Enter competition progress to website	Data on compeition progress	Progress in cometiton is logged to website
5	Track competiton progres on website	Team competition progress	Competition progress accurately displayed

Figure 8.1: Test Cases for Iron Engineer Web App

Unit testing was used early in the development process. By isolating each component and testing functionality ensures that the component works independently. As such, each step was first tested to work individually. These tests were done manually, and aimed to check various inputs such as SQL command lines when creating team name, or negative numbers during the logging of progress. This allowed our team to find issues within each module quickly and resolve the bugs before merging with the working repository.

After ensuring proper functionality, the component was added to the working build. Then integration testing was done. After a step was integrated, all previous steps were given an overview to make sure no other components were affected by the changes made. Integration testing ensures the system works as a whole after changes are made. These two testing strategies allowed our team to resolve bugs individually on each component, and as a group to make sure each other's components were working correctly.

Chapter 9

Difficulties Encountered

Most of the challenges and difficulties our team encountered can be boiled down to three main issues.

The first and most time consuming challenge our team encountered was the difficulty of properly setting up software environments. Our team had varied experience working with the technologies we used in our project. However, we did not have any experience with getting these different technologies set up and working together. Unsure of what to do at this stage, we looked to online tutorials to teach us what we needed to know. However, given that each team member was setting up the technology they were most familiar with, all team members ended up following different build configurations. This meant that we couldn't combine our work as our differing build environments made it so that one person's code would run properly in their environment, but not on their teammates'. This forced us to scrap our work and consolidate to a single build environment.

The second obstacle which inhibited our project was the lack of consistent communication. Given differing schedules and locations, our team opted to work remotely for most of the implementation stage. Not being in the same place at the same time meant that there were breakdowns in communication about what team members were working on, where they were in their progress, and what this left other team members to do. The consequence of this was duplicate work and buggy code which failed to interact with the other portions which were programmed in parallel.

Lastly our team struggled with time estimation for our development timeline. The part of our estimation which we underestimated the most was the process of connecting our various technologies together. While we knew how to use all of the technologies independently, connecting them together was difficult. Specifically, we struggled to properly connect our database to our Spring web application. We spent over 10 hours working on this while we estimated that the process would take 2 hours. The bug was so costly and time consuming that we were unable to fix it for our initial system demo. By properly estimating how long this would take we could have blocked out more time for it and made sure that it was functional in time for our demo.

Chapter 10

Suggested Changes

Our system has all of the hard functional requirements outlined during the development process. However, a few functionalities could be added to further improve the system. Currently, there is no real-time leader-board system for the Iron Bronco event. Administrators manually record the dates of completion for each team and upload a document listing all teams who completed the event - ordered by date. A leader-board page on the Iron Engineer website that sorts all the teams progress and displays the approved team names in descending order would bolster competition within the community. This could further incentive students to participate in the event. Furthermore, this system would be updated in real-time allowing for participants to see the progress of the top few teams before anyone has completed the event, encouraging competition among participants.

Furthermore, better integration with the concurrently running Iron Bronco event would be beneficial. The current system requires each new progress entry to be dated and approved by an administrator. To simplify this process, a printable sheet should be available for every team that shows each log and it's respective date of entry. This would make it easy for participants to turn in to the Iron Bronco administrators and expedite the administrators verification process.

Chapter 11

Lessons Learned

The biggest lessons we learned came from the mistakes we outlined previously. We will discuss each lesson learned by matching it to the mistake it emerged from.

The first lesson learned came from our struggles with setting up our development environment. We learned that getting all developers on the same page using the same technologies is a crucial step that takes time and cannot be implemented on the fly. As is outlined in software development methods like the waterfall method, it can be more important to finish the current step before moving on to the next one, rather than to try to move quickly and develop iteratively. While our team still sees the value in iterative development, we also now recognize that like the requirements, the development environment should be complete and consistent before beginning implementation.

The next lesson was the importance of constant communication in a remote software project. Not having the luxury of working side by side with our teammates, it became our responsibility to maintain communications as if this were still the case. The duplicate work and buggy code that resulted from our lack of communication cemented the realization that documentation and status updates have to be frequent as well as thorough in order to be meaningful in the context of a remote project.

The last lesson we learned was the challenge of time estimation and over promising in a software development context. We were aware of this issue, and doubled all of our estimates in order to be on the safe side. However, we found that our initial estimates were off by a magnitude of 5x. Granted, if apply the previous lessons learned related to environment configuration and frequent communication our initial estimates may have been more accurate. Unfortunately the idea that things will go smoothly next time is trap created by outcome bias that many software engineers fall victim too. As a result, it is our responsibility to estimate conservatively and promise only what the requirements specify. If ideas for additional features come up they should be documented but only discussed with clients after the core functionality has been successfully completed.

Appendix A

Installation Guide

A.1 Accessing The Repository

The Iron Engineer repository can be cloned from the following github:
https://github.com/rbl312/IronEngineerMaven.git

A.2 Database Setup

A.2.1 Database Schema

The database management software implemented in this project is MySQL. In order to create the database and schema needed for the system, a database design tool such as MySQL Workbench can be used to create the necessary database. To create the schema needed for Iron Engineer, execute the SQL script IronEngineerDatabaseScript.sql found with the installation guide. Once the database schema has been created, Amazon Web Service's Relation Database Service (RDS) is used to host the database for the system.

A.2.2 Connecting to AWS Relational Database Service

Within the repository, navigate through the following file directories to find the configuration file for hosting on an RDS instance.

- IronEngineerMaven
- src
- main
- java

- resources

Within the resources directory is the application.properties file. In this file, input the URL of the RDS instance, the username, and password used to access the database.

A.3 Deploying code to the cloud

The simplest way to get a web application up and running on Amazon's cloud services is through Elastic Beanstalk. A free deployment service included with EC2 purchase. Navigate to the Amazon Web Services (AWS) main console and select the Elastic Beanstalk (EB) option.

- Enter your application name.
- Choose the Tomcat Platform for the dropdown menu.
- Upload the code as a WAR file. In order to condense the application as a WAR file, navigate to the repository directory on your local computer and type mvn package into the command line. The output WAR file will be stored under the target folder of the repository on your local machine.
- Press the Create Application Button. EB will create your application and place it on a t2.micro EC2 instance with auto generated configurations.
- To further scale the application, refer to the AWS pricing guidelines and modify your instance in the configuration window of your EB portal.
- Whenever updates need to be made to the code running on the cloud. Simply make the change in the code, recompile the code as a WAR file, and select the upload and deploy option on the EB dashboard and upload the new WAR file.

Appendix B

User Manual

In our system we have two users - the participants and the administrators. As such, this section will be divided into two sections respectively.

B.1 Participant User Manual

Initially, a new participant is directed to the register page that allows one to either create or join a team.

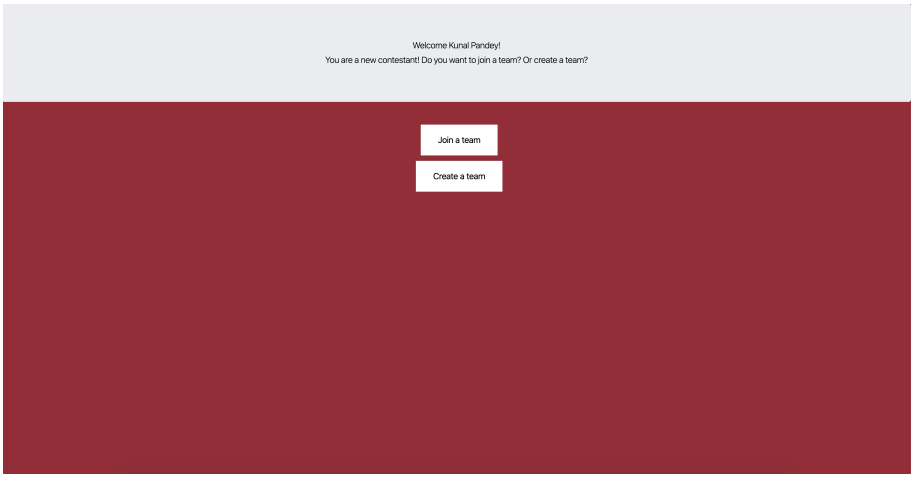

Figure B.1: Initial Registration Page

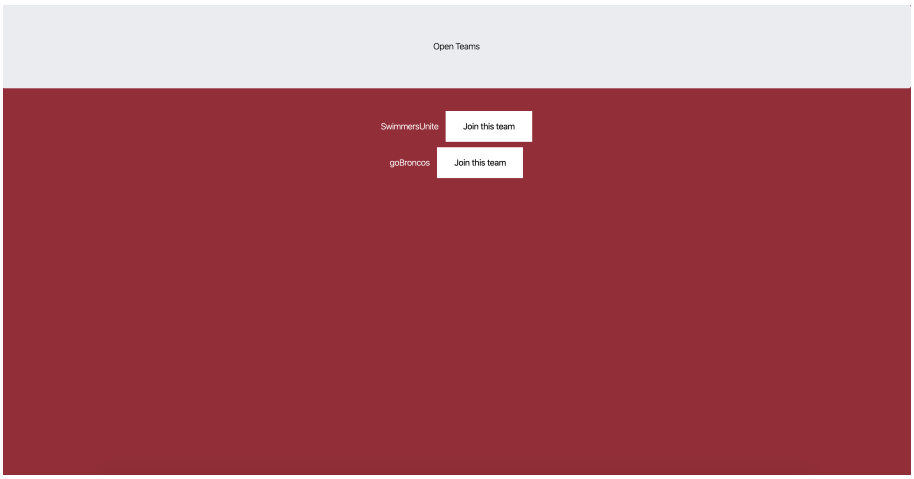

Figure B.2: Join Team Page

Figure B.3: Create Team Page

After joining/creating a team, the participant will view the home page where one can view team progress and update team progress with a new log entry.

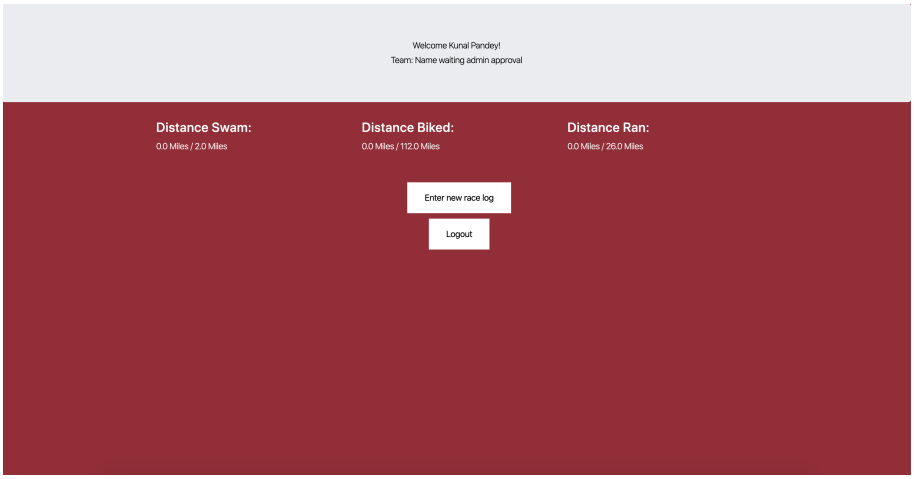

Figure B.4: Home Page

Figure B.5: Log Entry Page

B.2 Administrator User Manual

The administrator account will be enabled before hand-off to Iron Bronco administrators. The administrators are able to approve team names, view all team progress, remove a participant from the competition, and enable a different email with administrative privileges.

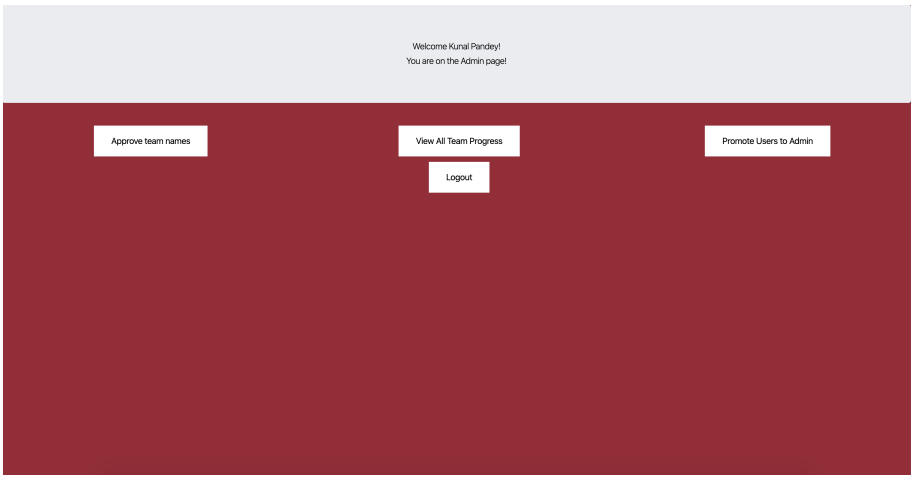

Figure B.6: Administrator Home Page

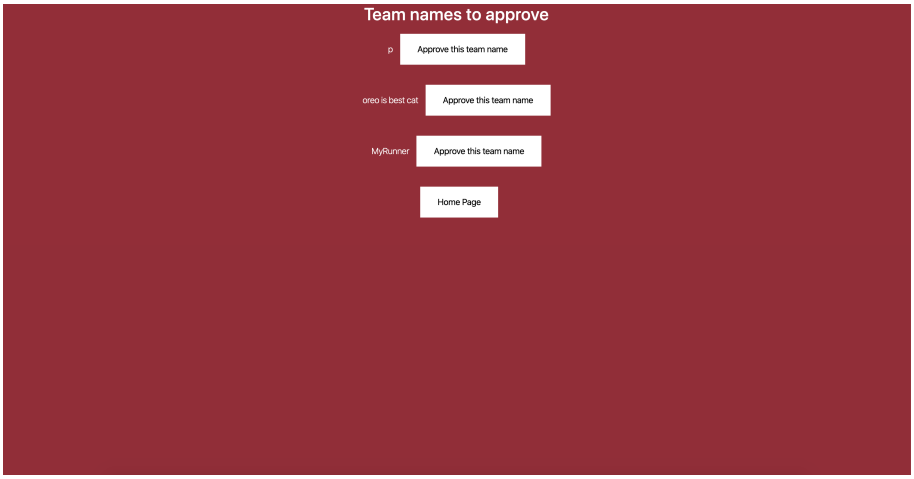

Figure B.7: Approve Teams Page

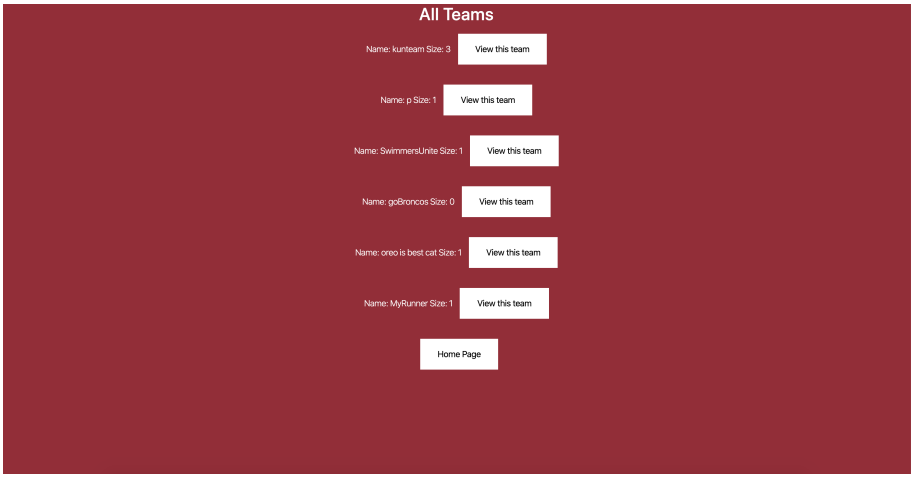

Figure B.8: View All Teams Page

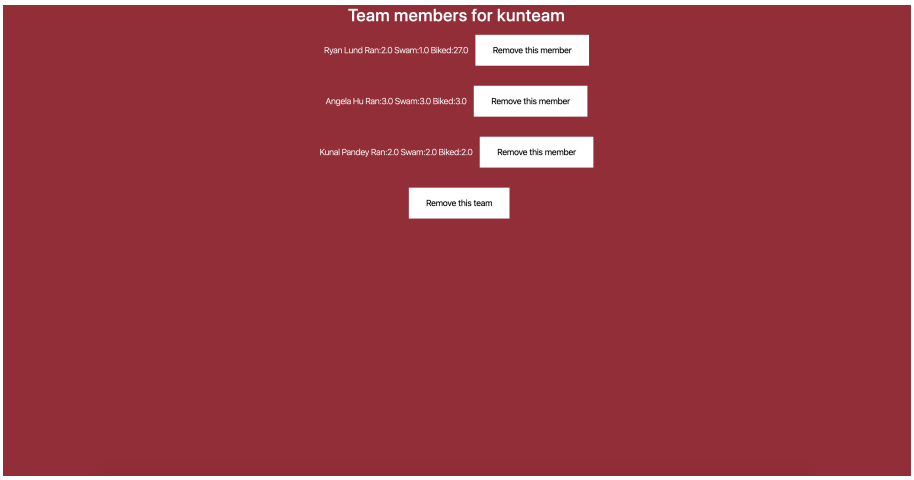

Figure B.9: View a Single Team Page

Figure B.10: Administrator Account Creation Page