

CS CAPSTONE PROJECT HAND OFF

JUNE 7, 2019

GYMNASTICS SCORING SOFTWARE

PREPARED FOR

OSU GYMNASTICS

MICHAEL CHAPLIN

Signature

Date

PREPARED BY

GROUP 18

10.0 SOFTWARE

ZECH DECLEENE

Signature

Date

THOMAS HUYNH

Signature

Date

MARY JACOBSEN

Signature

Date

NICHOLAS GILES

Signature

Date

Abstract

This document is designed to introduce, and describe the development and design of the 10.0 scoring software, as well as provide useful information to future capstone students on how to build upon this project.

CONTENTS

| | | |
|----------|---|-----------|
| 1 | Introduction | 5 |
| 1.1 | Who requested it? | 5 |
| 1.2 | Why was it requested? | 5 |
| 1.3 | What is its importance? | 5 |
| 1.4 | Who was/were your client(s)? | 5 |
| 1.5 | Who are the members of your team? | 5 |
| 1.6 | What were their roles? | 5 |
| 1.7 | What was the role of the client? | 5 |
| 2 | Requirements Document | 5 |
| 2.1 | Introduction | 5 |
| 2.1.1 | System Purpose | 5 |
| 2.1.2 | System Scope | 5 |
| 2.1.3 | System Overview | 6 |
| 2.1.4 | Definitions | 7 |
| 2.2 | Gantt Chart | 8 |
| 2.3 | System Requirements | 8 |
| 2.3.1 | Functional Requirements | 8 |
| 2.3.2 | Usability Requirements | 9 |
| 2.3.3 | Performance Requirements | 9 |
| 2.3.4 | System Interface | 9 |
| 2.3.5 | System Operations | 10 |
| 2.3.6 | System Modes and States | 10 |
| 2.3.7 | Physical Characteristics | 10 |
| 2.3.8 | Environmental Conditions | 11 |
| 2.3.9 | System Security | 11 |
| 2.3.10 | Information Management | 11 |
| 2.3.11 | Policies and Regulations | 11 |
| 2.4 | Verification | 11 |
| 2.5 | Appendices | 11 |
| 2.5.1 | Assumptions and Dependencies | 11 |
| 2.5.2 | Acronyms and Abbreviations | 12 |
| 3 | Design Document | 12 |
| 3.1 | Overview | 12 |
| 3.1.1 | Scope | 12 |
| 3.1.2 | Purpose | 12 |
| 3.1.3 | Intended audience | 13 |
| 3.1.4 | Conformance | 13 |

| | | |
|----------|--|-----------|
| 3.2 | Definitions | 13 |
| 3.3 | Design description information content | 13 |
| 3.3.1 | Introduction | 13 |
| 3.3.2 | SDD identification | 13 |
| 3.3.3 | Design stakeholders and their concerns | 14 |
| 3.3.4 | Design views | 15 |
| 3.3.5 | Design viewpoints | 15 |
| 3.3.6 | Design elements | 16 |
| 3.3.7 | Design overlays | 16 |
| 3.3.8 | Design rationale | 16 |
| 3.3.9 | Design languages | 16 |
| 3.4 | Design viewpoints | 17 |
| 3.4.1 | Introduction | 17 |
| 3.5 | Context viewpoint | 18 |
| 3.6 | Composition viewpoint | 18 |
| 3.7 | Logical viewpoint | 19 |
| 3.7.1 | /judge | 19 |
| 3.7.2 | /lineup | 20 |
| 3.7.3 | /meet | 21 |
| 3.7.4 | /player | 21 |
| 3.7.5 | /score | 22 |
| 3.7.6 | /team | 23 |
| 3.7.7 | /user | 24 |
| 3.7.8 | Dependency viewpoint | 24 |
| 3.7.9 | Information viewpoint | 25 |
| 3.7.10 | Patterns use viewpoint | 25 |
| 3.7.11 | Interface viewpoint | 26 |
| 3.7.12 | Structure viewpoint | 27 |
| 3.7.13 | Interaction viewpoint | 28 |
| 3.7.14 | State dynamics viewpoint | 28 |
| 3.7.15 | Algorithm viewpoint | 28 |
| 3.7.16 | Resource viewpoint | 28 |
| 4 | Tech Reviews | 29 |
| 4.1 | Thomas Tech Review | 29 |
| 4.2 | Zech Tech Review | 31 |
| | References | 33 |
| 4.3 | Mary Tech Review | 33 |

| | | |
|--|---|----|
| | | 3 |
| References | | 37 |
| 4.4 | Nick Tech Review | 37 |
| References | | 41 |
| 5 Weekly Blog Post | | 41 |
| 5.1 | Thomas' Weekly Blog Posts | 41 |
| 5.1.1 | Fall Term | 41 |
| 5.1.2 | Winter Term | 42 |
| 5.1.3 | Spring Term | 43 |
| 5.2 | Zech's Weekly Blog Posts | 44 |
| 5.2.1 | Fall Term | 44 |
| 5.2.2 | Winter Term | 44 |
| 5.2.3 | Spring Term | 45 |
| 5.3 | Nick's Weekly Blog Posts | 45 |
| 5.3.1 | Winter Term | 45 |
| 5.3.2 | Spring Term | 46 |
| 5.4 | Mary's Weekly Progress | 46 |
| 5.4.1 | Fall Term | 46 |
| 5.4.2 | Winter Term | 47 |
| 5.4.3 | Spring Term | 49 |
| 6 Final Poster | | 50 |
| 7 Project Documentation | | 50 |
| 7.1 | 10.0-Software-Group-18 | 50 |
| 7.2 | How to run code | 50 |
| 7.3 | Using Postman to Test | 51 |
| 7.4 | Endpoints | 51 |
| 8 Recommended Technical Resources for Learning More | | 51 |
| 8.1 | Postman | 51 |
| 9 Conclusions and Reflections | | 51 |
| 9.1 | Mary | 51 |
| 9.1.1 | What technical information did you learn? | 51 |
| 9.1.2 | What non-technical information did you learn? | 51 |
| 9.1.3 | What have you learned about project work? | 52 |
| 9.1.4 | What have you learned about project management? | 52 |
| 9.1.5 | What have you learned about working in teams? | 52 |
| 9.1.6 | If you could do it all over, what would you do differently? | 52 |
| 9.2 | Zech | 52 |

| | | |
|-------|---|----|
| 9.2.1 | What technical information did you learn? | 52 |
| 9.2.2 | What non-technical information did you learn? | 52 |
| 9.2.3 | What have you learned about project work? | 52 |
| 9.2.4 | What have you learned about project management? | 52 |
| 9.2.5 | What have you learned about working in teams? | 52 |
| 9.2.6 | If you could do it all over, what would you do differently? | 52 |
| 9.3 | Nicholas | 53 |
| 9.3.1 | What technical information did you learn? | 53 |
| 9.3.2 | What non-technical information did you learn? | 53 |
| 9.3.3 | What have you learned about project work? | 53 |
| 9.3.4 | What have you learned about project management? | 53 |
| 9.3.5 | What have you learned about working in teams? | 53 |
| 9.3.6 | If you could do it all over, what would you do differently? | 53 |
| 9.4 | Thomas | 53 |
| 9.4.1 | What technical information did you learn? | 53 |
| 9.4.2 | What non-technical information did you learn? | 53 |
| 9.4.3 | What have you learned about project work? | 54 |
| 9.4.4 | What have you learned about project management? | 54 |
| 9.4.5 | What have you learned about working in teams? | 54 |
| 9.4.6 | If you could do it all over, what would you do differently? | 54 |

Appendix

1 INTRODUCTION

1.1 Who requested it?

New gymnastics scoring software was requested by Oregon State gymnastics administrators.

1.2 Why was it requested?

The gymnastics scoring software was requested because the current software used in Gill Coliseum is not reliable and difficult for users to operate.

1.3 What is its importance?

It is important for the scoring software to work well in Gill Coliseum because gymnastics meets are televised and followed by many fans so scores need to be accurate and easy for fans to follow.

1.4 Who was/were your client(s)?

Michael Chaplin Associate Head Coach of Oregon State Women's Gymnastics

1.5 Who are the members of your team?

Mary Jacobsen, Nicholas Giles, Zech DeCleene Thomas Huynh

1.6 What were their roles?

Nicholas is the team captain. Mary and Zech are in charge of back end development. Nicholas and Thomas are in charge of front end development.

1.7 What was the role of the client?

To give us feedback and get us in contact with the people necessary in order to properly finish the project. The client was also used as a tester for the client side design.

2 REQUIREMENTS DOCUMENT

2.1 Introduction

2.1.1 System Purpose

This system is for the scoring of gymnastics meets held in Gill Coliseum. The system will take in, calculate, organize, and display scores as well as format the scoring data.

2.1.2 System Scope

The purpose of this system is to provide an improved viewing experience for OSU Gymnastics spectators as well as making the scoring process easier for the staff. The system will comprise of a web application running on the Gill server that is accessible by the meet staff to input or adjust scores. The scores will then be formatted to be displayed on the scoreboards inside Gill. The application will be able to format all current data into a score card that the user can print out at any time. The system will also comprise of a back-end API with a SQL database running on the Gill servers to store all scoring information.

2.1.3 System Overview

The system is comprised of three main components.

- **Web Application:**

The Web application's main function will be the user interface which allows the meet staff to add, edit and print scores. The web application will be responsible for formatting and calculating scores then pushing the data to the database and scoreboard.

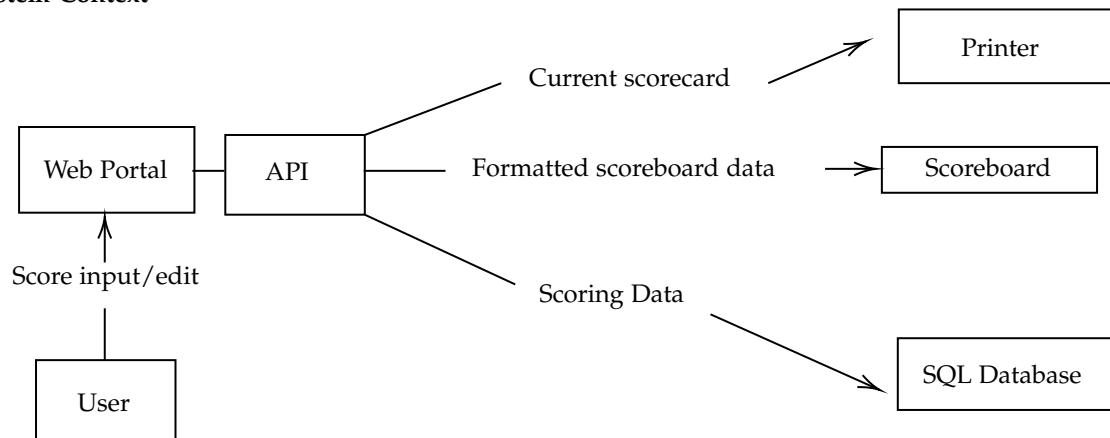
- **Scoreboard Utilization:**

Data collected from the web application will be pushed to display up-to-date information on current events of the meet.

- **Database:**

The database will maintain information on current and past information. The information will include the teams, athletes, and the scores associated with each athlete.

System Context



System Functions

The system's major capabilities will be managing, displaying, and exporting scores. A team score is calculated by adding the top five individual scores from that team for all four events. An individual score is given in 0.05 increments between 0 and 10 points with 10 being a perfect score. The software system should be able to drop the lowest and highest scores given for an individual if there are at least four judges. After dropping those scores, the rest should be averaged to make a final individual score. All individual scores should be sorted from highest to lowest by event regardless of team. In Gill Coliseum, there are six screens that should display the individual and team scores to the audience. Among the six screens, there are three different aspect ratios and resolutions to account for. There are five scoring tables equipped with computers including one master scoring table. Finally, the software system should be able to export the scores data. Also, the scores should be able to be printed in a readable format for the coaches, staff, and judges immediately at any time during a meet.

User Characteristics

One very important user group of the system will be the people inputting scores sitting next to the judges assigning scores to the gymnasts. The number of judges will vary between two and six. After the judges decide on the score, they

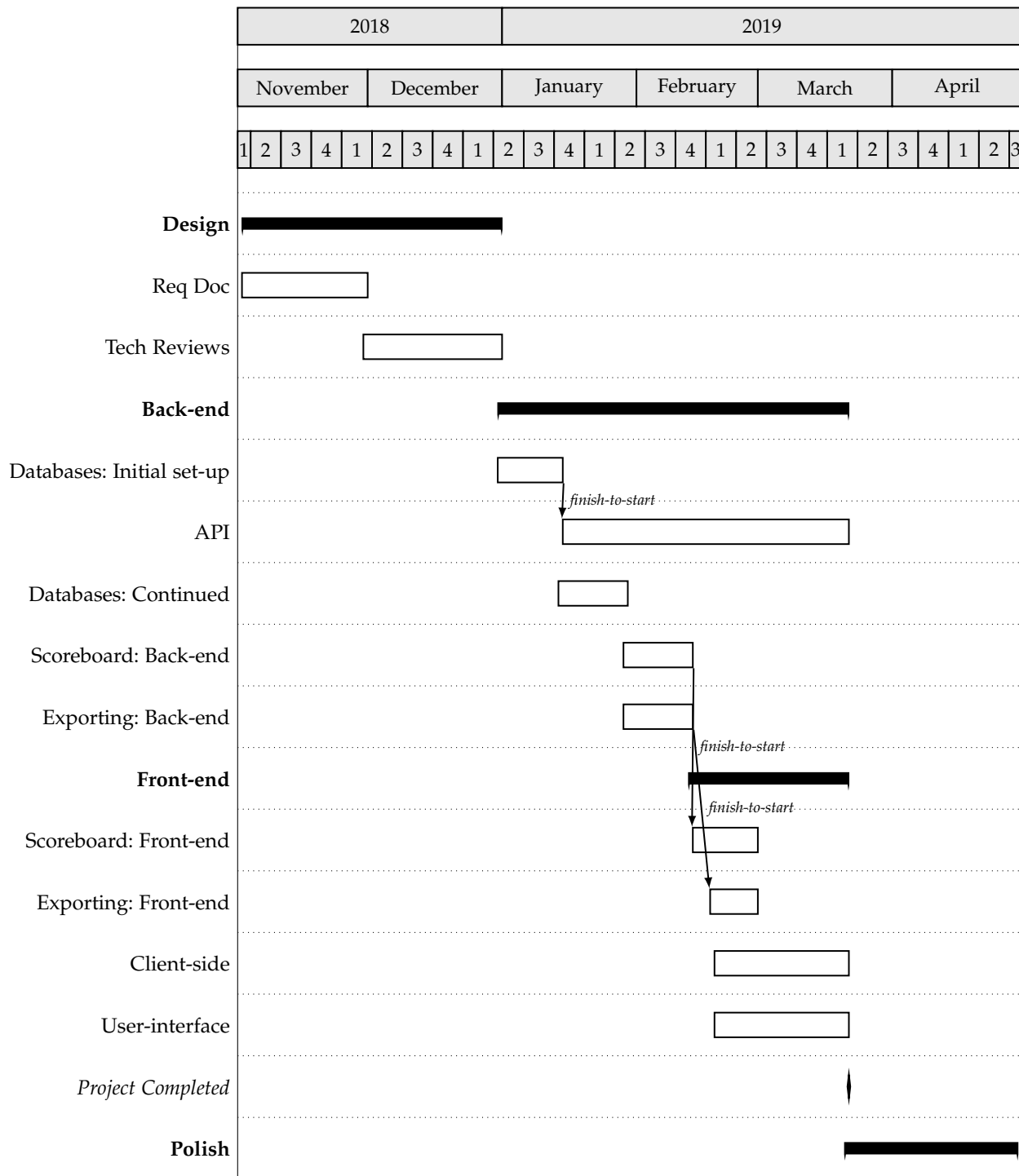
will give the scores to the computer operators who will input the scores into the computers. There will be a person to input scores for each judge. In addition, there will be an information technology (IT) staff of about four people to maintain the system.

2.1.4 Definitions

All Around is all four events which is vault, bars, beam, and floor for women's gymnastics. Not all gymnasts do all four events in college gymnastics so only some gymnasts have all around scores.

Road to Nationals is a website that keeps all the scores for NCAA gymnastics.

2.2 Gantt Chart



2.3 System Requirements

2.3.1 Functional Requirements

The system must be able to take in the number of judges from the user and have an option to drop the high and the low score. The scores given by the judges must be averaged to calculate the score for a gymnast's routine and the score for that routine will be added to the team score. There also must be an option to add exhibition routines on events that do not count towards the team score and are not displayed.

The system must be able to send the scores given by each judge and the final averaged score for each gymnast on each event into a database on the Gill server using SQL. The system must also be able to retrieve the scores from the database using SQL queries. The system must be able to send the data in a text file to the display boards. The display boards take care of displaying.

At the end of the meet, the scores must be able to be exported in the format required by Road to Nationals. These exported scores will include the same information as the printable version.

Making an application with different levels of authentication is a stretch goal. The fans could follow the meets with live scoring without being able to modify data or see everything that they don't need to. The staff with higher authentication could use the application to run the scoring system.

2.3.2 Usability Requirements

The user must be able to set up scoring for a meet by inputting the number of judges, choosing if scores are dropped or not, and entering the line up for each event for each team and any exhibition routines. The line ups and exhibitions can be changed throughout the meet before scores are inputted. The gymnasts doing all four events do not have to be inputted specially for all around. The system should automatically put any gymnast inputted on all four events in the all around and add up the four scores to compute the all around score. After a gymnast goes, her scores are inputted by the user and the system automatically averages the scores (dropping the high and low if that setting is on) and sends the data to a database that gets queried when needed. The displaying of scores is not automatic so that the user can decide when to display scores. The user can click an option to print out the scores at any time. At which point, the system will put the data into the printable format for the user. The user can also click to export the scores and the scores will be exported to the required format.

2.3.3 Performance Requirements

The performance requirements are that the system must be able to handle the maximum number of judges without errors or significant latency. The system must also be able to print out score cards during the meets without interfering with the system or significantly reducing performance.

2.3.4 System Interface

Our system will be implemented with three system interfaces, or five system interfaces if stretch goals are reached. The three main interfaces of the system are the database, the API, and the web portal. The database will be an interface used by the API to safely and securely access the data that will be used and collected by our system. Our database will be accessed using queries to multiple tables. An example query would be `SELECT COUNT(*) players FROM oregonstate` which would hypothetically grab all of the players from the oregonstate table and return the results for use by our API. Continuing, the API will be an interface for the web portal to send CRUD commands to the API that will then do the difficult work of handling the data. Lastly, the web portal is the interface for users to access the API in an easily usable manner. The web portal will require the user to select the gymnast performing, the event the gymnast

performed, and their score and send that to the API. The last two interfaces would consist of an iOS and Android app that would provide users with live scores.

2.3.5 System Operations

There are multiple operations that our system must be able to perform, and the operations can be broken down three ways: database operations, API operations, and client operations.

To start, the database will have to be able to create, read, update, and delete data. These are known as CRUD operations and are common practice when using databases.

Next, are the operations that need to be performed by the API which are much more involved. The API will need to be able to take in data in the form of JSON, and from there the API will have to parse the JSON for three major things: authentication, data, and command. This is the cornerstone operation of our server. After receiving and parsing the data, the API will have to be able to handle three major operations: printing the current score sheet, updating the display boards, and processing the score. Printing the score sheet will consist of the API grabbing the current data of the meet, formatting it into a pdf, and then sending the pdf to a printer. Updating the display boards will consist of the API grabbing the most recently processed team scores, the current gymnast, and the current event then formatting the data into a text document and sending that text document to the display boards. The last operation will consist of the API performing calculations on the scores sent in, followed by updating the team scores, and then querying the database to add the new scores to the correct player and updating the team scores.

The last set of operations are the client operations. These will be performed by the web portal that users interact with. Only three operations are needed to be done by the web portal; these operations are handling log ins, setting up a meet, and sending scores. Due to the security concern and the want for data to only be handled by trust worthy sources, the web portal will require users to provide a user name and password to prove their identity as a non malicious agent. Cookies and sessions will be implemented to store information about current meets and users. The next operation is setting up a meet, this will involve choosing the number of judges and teams in each meet. After the number of judges and teams have been chosen, the operation will then prompt the user for the names of the teams participating and the players participating for each team. For sending scores, the application will allow for users to choose the team and player, and from there the user will be able to insert the scores from each judge then send the scores to the API.

2.3.6 System Modes and States

The modes for the system are different numbers of judges and dropping the high and the low score or not. If there are three or less judges, dropping the high and the low score is not an option since there wouldn't be anything left to average. The system states include the input gathered from the user for setup, the inputted scores from the user, and the scores calculated and put into the database.

2.3.7 Physical Characteristics

Our program will require the physical requirements of a server that can run a SQL database, display screens, and at least one web enabled device. The server will need to be able to run a containerized API with a SQL server attached.

The display screens will need to be able to take in a text document of information that will be displayed to the screens, and lastly, the web enabled devices will need to be able to access a web page.

2.3.8 Environmental Conditions

Our environment will primarily be Gill Coliseum, which is a weather controlled indoor environment. The hardware that needs to be used is also already housed in Gill Coliseum allowing for easy access for both software and hardware. Overall, the environment will not pose an issue for our project.

2.3.9 System Security

System security will be implemented using CIA security principles. We plan to store our data in a SQL database that can only be accessed by specific administrators. Next, in order to ensure the data is not tampered with we will be implementing different levels of users. The base level will be allowed to read team scores and individual scores of players, whereas administrator level accounts will have the ability to access the create, update, and delete functions of our API. Lastly, we plan to keep our data and API available by keeping it on a private wifi network that will avoid the traffic from users at the gymnastics meets to ensure that we do not lose packets due to throughput. Another plan is the stretch goal of rate limiting, which will help protect the API from Denial of Service attacks.

2.3.10 Information Management

We will be collecting data from the organizers of the gymnastics meets that the software will be used with. The data will be sent by the teams, judge's assistants, and organizers. The teams will send their active roster, the judge's assistants will send their scores for each event, and the organizer will send data about the date and location of the event. The data will be sent over the network to a local server that will be set up by the organizer. Each local server will have a local SQL database that will store all of the information needed, and this database will only be accessible by a root user that is created by the API allowing only the API, and the organizers of the event to be able to directly access the database. At the end of each meet, the scores can be downloaded as a CSV and be sent to Road to Nationals which is an organization that stores and compiles gymnastics scores from across the nation.

2.3.11 Policies and Regulations

This section is intentionally left blank so we can check with the sponsor to make sure there are no Oregon State University, NCAA, or Federal policies or regulations that we have to follow for potential protections of student athletes.

2.4 Verification

The client will verify that the system meets the requirements by running a mock meet that goes through all the steps of a regular meet for the scoring system. After going through the scoring of a mock meet, the client and our team will decide if the software meets each of the requirements listed in this requirements document. We will also decide if any stretch goals were met but those are not necessary for the system to be functional.

2.5 Appendices

2.5.1 Assumptions and Dependencies

Some assumptions that our team is making is that the hardware provided by Gill, such as the scoreboard or server, are adequate in terms of performance and no major work will need to be done upgrading these hardware components.

2.5.2 Acronyms and Abbreviations

API: Applied Programming Interface

CIA: Confidentiality Integrity Availability

CRUD: Create Read Update Delete

HTTP: Hyper Text Transfer Protocol

HTTPS: Hyper Text Transfer Protocol Secure

NCAA: National Collegiate Athletics Association

OSU: Oregon State University

SQL: Structured Query Language

CSV: Comma Separated Value

3 DESIGN DOCUMENT

TABLE 1
Change Log

| Section | Original | New |
|---------|--|--|
| 4.6 | Required input of JSON Files. | Can use JSON files or Manual input. |
| 4.8 | Mentioned Microsoft SQL. | Changed Microsoft SQL to just SQL. |
| 5.4.1 | Teams endpoint object and description | Teams moved to 5.4.6 and 5.4.1 is now judge endpoint object and description. |
| 5.4.2 | Players endpoint object and description | Players moved to 5.4.4 and 5.4.2 is now lineup endpoint object and description. |
| 5.4.3 | Events endpoint object and description | Events endpoint was removed and replaced with meet endpoint object and description. |
| 5.4.4 | Nothing. | Player endpoint object and description. |
| 5.4.5 | Nothing. | Score endpoint object and description. |
| 5.4.6 | Nothing. | Team endpoint object and description. |
| 5.4.7 | Nothing. | User endpoint object and description. |
| 5.6 | UML diagram of database. | Reworked database and changed UML diagram to an Entity Relationship Diagram that more accurately shows the database. |
| 5.8 | No screenshots. | Added Screenshots . |
| 5.11 | Originally required a printing feature and XML export. | Removed printing and XML exporting. |
| 5.12 | Originally didn't have average score algorithm. | Added average score algorithm. |
| 5.13 | Nothing here . | Added NPM modules and descriptions. |

3.1 Overview

3.1.1 Scope

This document details the plans to implement the Gymnastics Scoring Software. To accomplish these goals a system will be implemented that consists of a database, web portal, API, printer interface, and an interface with the scoreboard inside Gill Colosseum.

3.1.2 Purpose

The purpose of this system is to provide an improved viewing experience for OSU Gymnastics spectators as well as making the scoring process easier for the judges and data inputters. This document will provide detailed information on

the implementation of the Gymnastics Scoring Software. The document will detail the implementation of the database, web portal, API, and scoreboard interface.

3.1.3 *Intended audience*

This document is intended for the IT team at Gill Colosseum and anyone responsible for the implementation of this system.

3.1.4 *Conformance*

Firstly, we shall make set-up time for a meet to be under a half hour. Secondly, we shall make sure that the scoring and results are correct to NCAA gymnastics standards. Thirdly, we shall make sure that the program is able to update the scoreboards and display peripherals in a timely manner. Fourthly, we shall ensure that our program can successfully print out the meet score sheet at any point during the gymnastics meet. Fifthly, we shall make sure that our program can export the data from the meet to the Road to Nationals database as required by the NCAA.

3.2 Definitions

| Term | Definition |
|-------------------------------|--|
| V/B/B/F | Vault, Bar, Beams, and Floor. The four events in gymnastics are commonly grouped together. When V/B/B/F is used it means that each will have an individual variable. |
| SDD | Software Design Description. |
| Admin (authentication level) | The highest level of authentication. Moderators have the ability to set-up teams, meets, and delete data. |
| Scorer (authentication level) | The lowest level of authentication, Judge stands for the accounts that will be in charge of inputting singular judges scores. Judge level accounts are able to input scores and get gymnast and team data. |
| UML | Unified Modeling Language. |
| API | Application Programming Interface. |

3.3 Design description information content

3.3.1 *Introduction*

The design description information content section will describe the identification of the software design description, identified design stakeholders, identified design concerns, selected design viewpoints, design views, design overlays, and design rationale. Each of the design viewpoints will have type definitions of its allowed design elements and design languages. languages

3.3.2 *SDD identification*

- Date of issue and status

The design document date of issue is November 27, 2018. The current status of the design document is 2nd draft.

- Scope

The scope of the 10.0 Software design description is the design of the web portal, API, and database, and how each of

those components interact.

- Issuing organization

This project was issued by Oregon State Gymnastics which is a part of Oregon State Athletics.

- Authorship

The members of group 18 will create and own the gymnastics scoring software (10.0 Software). There are currently no plans for any copyrights.

- Context

We are designing 10.0 Software to replace the scoring software that is currently being used for gymnastics meets in Gill Coliseum because it is outdated and has problems such as a long set up time and reliability issues.

- Design languages for design viewpoints

We will use UML for context, composition, logical, dependency, information, patterns, interface, structure, interaction, state dynamics, algorithm, and resources.

- Body

The software design description for this system includes the web portal, API, database, and scoreboard server components and how they interact with each other. For the design of this gymnastics scoring software (10.0 Software), how the components interact will require most of the description. The purpose of each component is to interact with and possibly provide an interface for other components. The API is an interface for the web portal to access the database and the scoreboard server. The web portal will provide an interface for the users to use the whole system.

- Change history

The 10.0 Software project does not have a change history yet since it is still in the design phase.

3.3.3 *Design stakeholders and their concerns*

- Stakeholders

The stakeholder for 10.0 Software is the associate head coach of the Oregon State gymnastics team, Michael Chaplin and Gill Coliseum IT.

- Stakeholder concerns

The design concerns of the stakeholders are that SQL should be used for the database, that the software is designed in a way that the setup time before meets takes no longer than 30 minutes, and that it is designed so that the user can easily run the web portal.

- Addressed concerns

The system is designed with a SQL database. The setup is designed so that the user can upload a JSON file with the gymnasts on a team instead of having to input each gymnast on each team which will save time for the setup. The web

portal is designed in such a way that the user can easily run it by making it as simple as possible while still providing the necessary functionality.

3.3.4 *Design views*

This project has three design views based on the concerns of the stakeholders. The three design views are focusing on setup time, web portal usability, and a SQL database.

3.3.5 *Design viewpoints*

- Viewpoint name

The name of the first viewpoint is setup.

- Design concerns

The setup time of the software for a gymnastics meet must take thirty minutes or less.

- Design elements

The user will use the web portal to set up the meet. Specifically, the meet setup page and the team setup page of the web portal will be used. The setup design viewpoint does not allow for the user to input one gymnast at a time because that may take longer than thirty minutes. Instead, the user will give a JSON file for each team with the gymnasts on that team for team setup.

- Analytical method

The method used to construct the setup time design view is to make design choices based on how much time they add to the setup.

- Viewpoint name

The name of the second viewpoint is usability.

- Design concerns

The web portal must be efficient and easy for the user to run 10.0 Software with.

- Design elements

The web portal is the main element for usability because that is the only part the user interacts with.

- Analytical method

The method used to construct the usability design view is to make design choices with the user in mind.

- Viewpoint name

The name of the third viewpoint is database.

- Design concerns

The software must use a SQL database.

- Design elements

The database is the main design element for the database viewpoint but the API is a secondary element since it interacts with the database.

- Analytical method

The method used to construct the database design view is to make design choices using a relational database.

3.3.6 *Design elements*

- **Web Portal:** The web portal is where the user interacts with the system for entering scores, printing, and displaying information on the scoreboard.
- **Setup:** The setup element requires the user to insert data about both meets and teams.
- **API:** The API is the logical process responsible for reacting to user requests.
- **Database:** The SQL database hosted on Gill Coliseum's server will store all team and gymnast related data.

3.3.7 *Design overlays*

Security and authentication are also important to the stakeholders and are therefore an area of focus in the design of 10.0 Software.

3.3.8 *Design rationale*

We decided to use a Microsoft SQL database and web portal for our gymnastics scoring software. An important stakeholder concern is ease of use. We think that implementing a web application with a SQL database as the backbone will make the system very user friendly as well as fast to set up especially compared to the current system. A desktop application to score the gymnasts was also considered, but we felt the web application fits our purpose better. A design concern is that the stakeholders may want to make changes to the system after testing it and the web application would be simpler to change. In the more distant future, the web application can be continually upgraded with new features easily. We decided to use a SQL based database because that is what the Gill IT staff wanted. The fast set up time design concern was addressed by allowing users to input a JSON file with the team rosters.

3.3.9 *Design languages*

The selected design language for our software design description is UML. The three primary categories of UML diagrams are structure diagrams, behavior diagrams, and interaction diagrams. We will use UML to clarify design viewpoints.

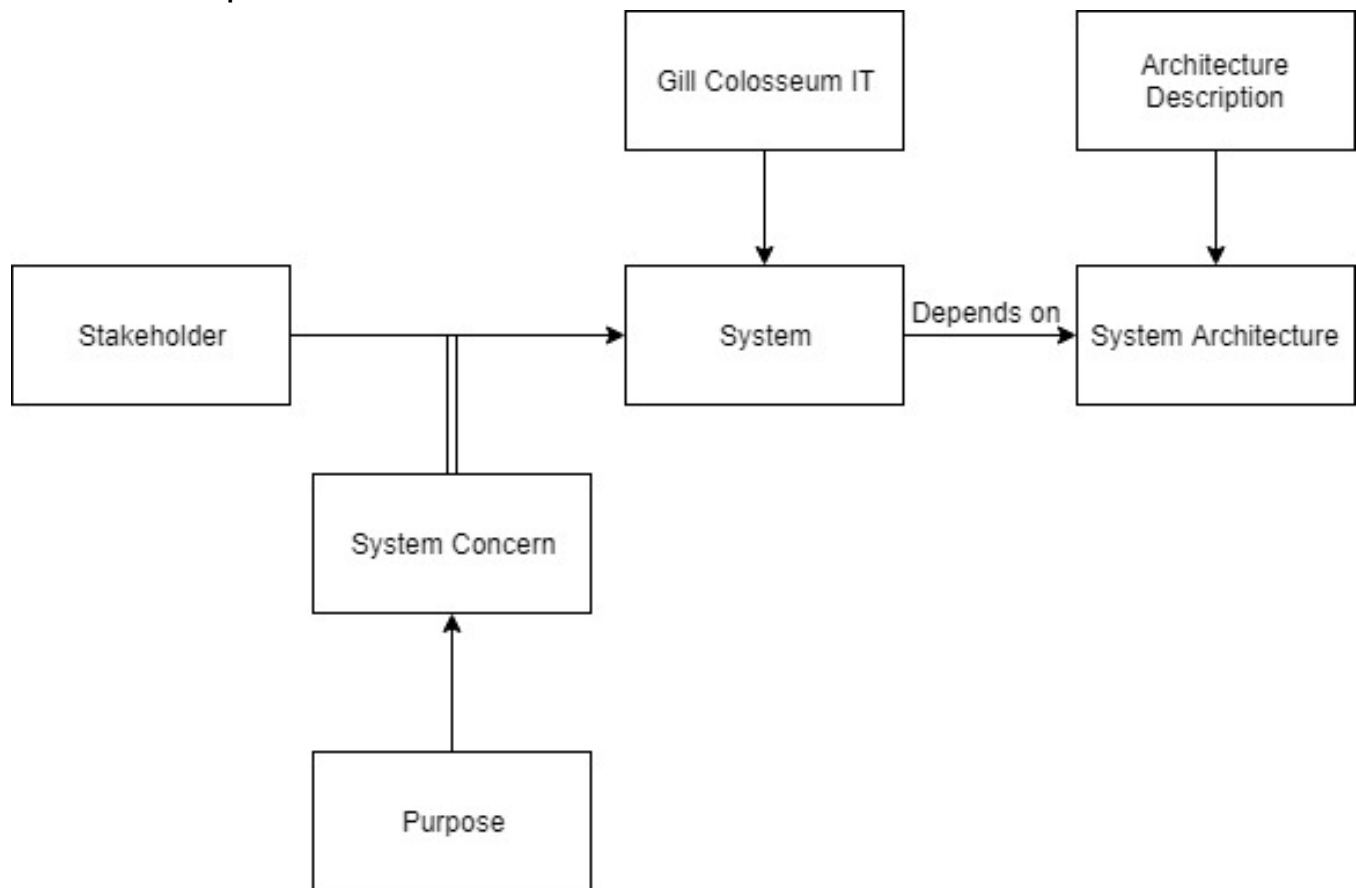
3.4 Design viewpoints

3.4.1 Introduction

Table 1—Summary of design viewpoints

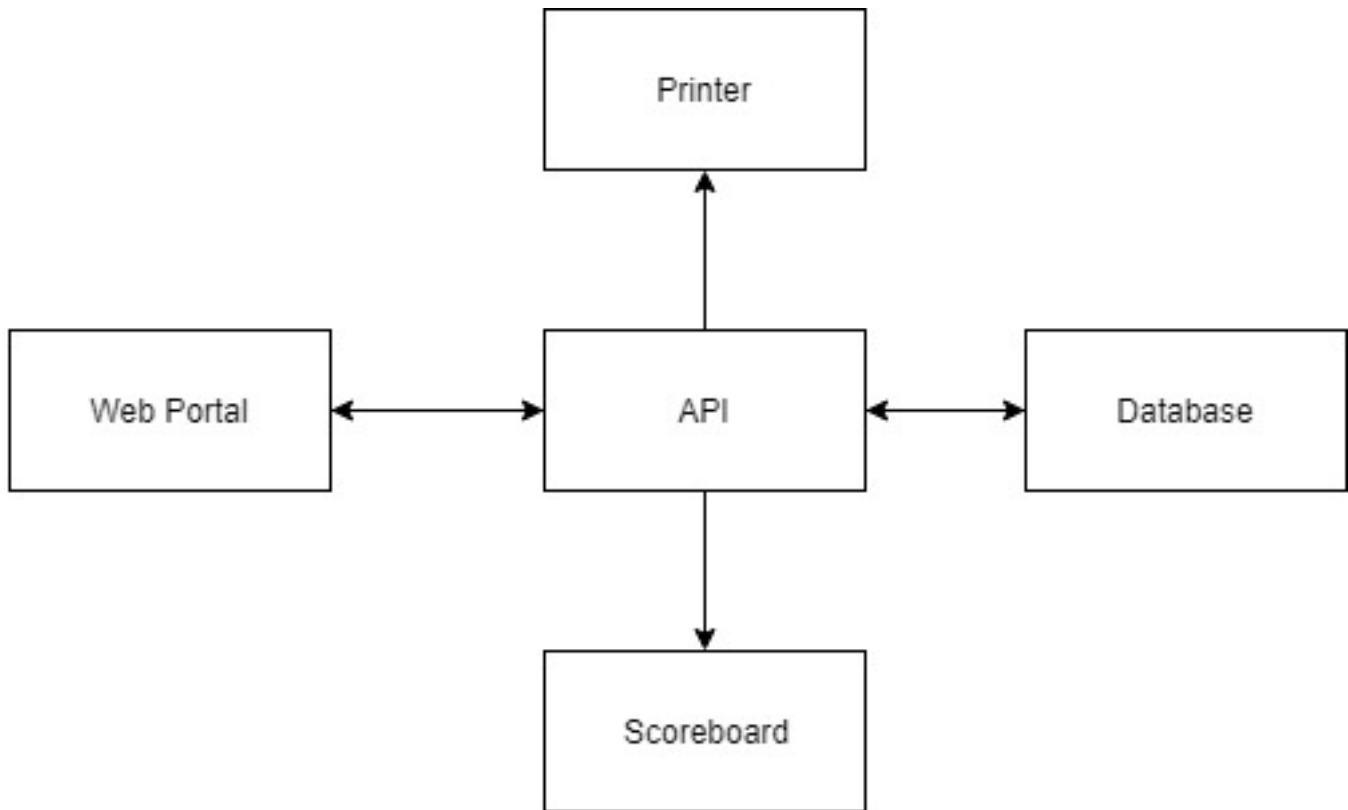
| Design Viewpoint | Design concerns | Example design languages |
|-----------------------|---|---|
| Context (5.2) | Gill coliseum IT people, judge assistants/OSU gymnastic data inputters | UML use case diagrams |
| Composition (5.3) | SQL database, API server, client side web portal, scoreboard server | UML package diagram and UML component diagram |
| Logical (5.4) | endpoints will act as classes due as they will call functions and use data | UML class diagram and UML object diagram |
| Dependency (5.5) | npm modules, endpoints, and web server | UML package diagrams and component diagrams |
| Information (5.6) | Persistent data will need to be constantly reordered and updated as a meet is under progress | UML class diagram |
| Patterns (5.7) | Multiple endpoints that allow for abstraction of data to make simple and effective API calls | UML composite structure diagram |
| Interface (5.8) | Web portal | UML component diagram |
| Structure (5.9) | HTTP server structure with a client-server architecture | UML structure and class diagram |
| Interaction (5.10) | The client will interact with the server, the server will interact with the scoreboard server, printing service, and client | UML sequence and communication diagram |
| State dynamics (5.11) | Setting up meets based on number of teams and judges present | UML state machine diagram |
| Algorithm (5.12) | It needs to be quick enough to display scores before fans get restless | Decision tale |
| Resources (5.13) | npm modules | UML |

3.5 Context viewpoint



3.6 Composition viewpoint

This system is made up of 5 components which are the web portal, API, database, printer interface and scoreboard interface. The API is the core of the system and is responsible for communication between the user at the web portal with the printer, scoreboard, and database. The web portal allows the user to access the database as well as printing and scoreboard features.



3.7 Logical viewpoint

Due to the use of a HTTP server and client-server architecture classes are not primarily present, and are instead replaced with server endpoints. Three major endpoints will be used: teams, events, and gymnasts.

3.7.1 */judge*

| <i>/judge</i> |
|--|
| - int: id - varchar: name - int: meetID |
| + GET <i>/judge</i> + GET <i>/judge/meet/:meetID</i> + POST <i>/judge</i> + PUT <i>/judge/:judgeID</i> + DELETE <i>/judge/:judgeID</i> |

| Name | Return | Authentication level | Definition |
|-------------------------|-----------------|----------------------|--|
| GET /judge | judge[] | None | Returns a list of all judges in the database. |
| GET /judge/meet/:meetID | Judge[] | None | Returns a list of all judges that participated in the meet specified by :meetID. |
| POST /judge | Status code 201 | Admin | Creates a judge object in the database. Data is grabbed from the body of the request sent to the server. |
| PUT /judge/:judgeID | Status code 204 | Admin | Updates the judge object specified by :judgeID. |
| DELETE /judge/:judgeID | Status code 204 | Admin | Deletes the judge object specified by :judgeID. |

3.7.2 /lineup

| /lineup |
|---|
| <ul style="list-style-type: none"> - int: id - int: playerID - int: teamID - int: order - varchar: event - int: meetID |
| <ul style="list-style-type: none"> + GET /lineup/:meetID/:teamID/:event + GET /lineup/score/:meetID/:teamID/:event + POST /lineup + PUT /lineup/:lineupID + DELETE /lineup/:lineupID |

| Name | Return | Authentication level | Definition |
|--|-----------------|----------------------|--|
| GET /lineup/:meetID/:teamID/:event | Lineup[] | None | Returns the lineup for the meet specified by :meetID, the team specified by :teamID, and the event specified by :event. |
| GET /lineup/score/:meetID/:teamID/:event | Float | None | Returns the score of the top five gymnasts for the lineup for the meet specified by :meetID and the team specified by :teamID. |
| POST /lineup | Status code 204 | Admin | Creates a new lineup object. |
| PUT /lineup/:lineupID | Status code 204 | Admin | Updates the lineup object specified by :lineupID. |
| DELETE /lineup/:lineupID | Status code 204 | Admin | Deletes the lineup object specified by :lineupID. |

3.7.3 /meet

| |
|---|
| /meet |
| - int: id - varchar: name |
| + GET /meet + GET /meet/:meetID + POST /meet + PUT /meet/:meetID + DELETE /meet/:meetID |

| Name | Return | Authentication level | Definition |
|----------------------|-----------------|----------------------|--|
| GET /meet | Meet[] | None | Returns an array of all meets in the database. |
| GET /meet/:meetID | Meet | None | Returns a single meet object specified by :meetID. |
| POST /meet | Status code 204 | Admin | Creates a new meet object. |
| PUT /meet/:meetID | Status code 204 | Admin | Updates the meet object specified by :meetID. |
| DELETE /meet/:meetID | Status code 204 | Admin | Deletes the meet object specified by :meetID. |

3.7.4 /player

| |
|---|
| /player |
| - int: id - int: playerNum - varchar: name - int: teamID - float: vaultScore - float: barScore - float: beamScore - float: floorScore - float: AAScore - int: meetID |
| + GET /player/:meetID/:teamID + GET /player/:playerID + GET /player/:playerID/:event + POST /player + PUT /player/:playerName + DELETE /player/:playerID |

| Name | Return | Authentication level | Definition |
|------------------------------|-----------------|----------------------|---|
| GET /player/:meetID/:teamID | Player[] | None | Gets the players for the meet specified by :meetID and the team specified by :teamID. |
| GET /player/:playerID | Player | None | Gets the player object specified by :playerID. |
| GET /player/:playerID/:event | Float | None | Gets the score of the player object specified by :playerID for the event specified by :event. |
| POST /player | Status code 204 | Admin | Creates a player object. |
| PUT /player/:playerID | Status code 204 | Admin | Updates the player object specified by :playerID. |
| DELETE /player/:playerID | Status code 204 | Admin | Deletes the player object specified by :playerID. |

3.7.5 /score

| /score |
|--|
| <ul style="list-style-type: none"> - int: id - int: playerID - int: judgeID - float: score - varchar: event - boolean: exhibition - int: meetID |
| <ul style="list-style-type: none"> + GET /score + GET /score/meet/:meetID + GET /score/average/:meetID/:playerID/:event + POST /score + PUT /score/:scoreID + DELETE /score/:scoreID |

| Name | Return | Authentication level | Definition |
|---|-----------------|----------------------|---|
| GET /score | Score[] | None | Returns all of the scores in the database. |
| GET /score/meet/:meetID | Score[] | None | Returns all of the scores for the meet specified by :meetID. |
| GET /score/average/:meetID/:playerID/:event | Float | None | Returns the average score for the player specified by :playerID from the meet specified by :meetID for the event specified by :event. |
| POST /score | Status code 204 | Admin or Scorer | Creates a score object |
| PUT /score/:scoreID | Status code 204 | Admin or Scorer | Updates the score object specified by :scoreID |
| DELETE /score/:scoreID | Status code 204 | Admin | Deletes the score object specified by :scoreID |

3.7.6 /team

| /team |
|---|
| <ul style="list-style-type: none"> - int: id - float: teamScore - varchar: teamName - float: vaultScore - float: barsScore - float: beamScore - float: floorScore - int: meetID |
| <ul style="list-style-type: none"> + GET /team/:meetID/meet + GET /team/:teamID + POST /team + PUT /team/:teamID + DELETE /team/:teamID |

| Name | Return | Authentication level | Definition |
|------------------------|-----------------|----------------------|--|
| GET /team/:meetID/meet | Team[] | None | Returns an array of team objects for the meet specified by :meetID |
| GET /team/:teamID | Team | None | Returns the team object specified by teamID |
| POST /team | Status code 204 | Admin | Creates a team object |
| PUT /team/:teamID | Status code 204 | Admin | Updates the team object specified by teamID |
| DELETE /team/:teamID | Status code 204 | Admin | Deletes the team object specified by teamID |

3.7.7 /user

| |
|--|
| /user |
| - int: id - float: teamScore - varchar: username - int: auth - varchar: hash |
| + GET /endSession + GET /getSession + POST /login |

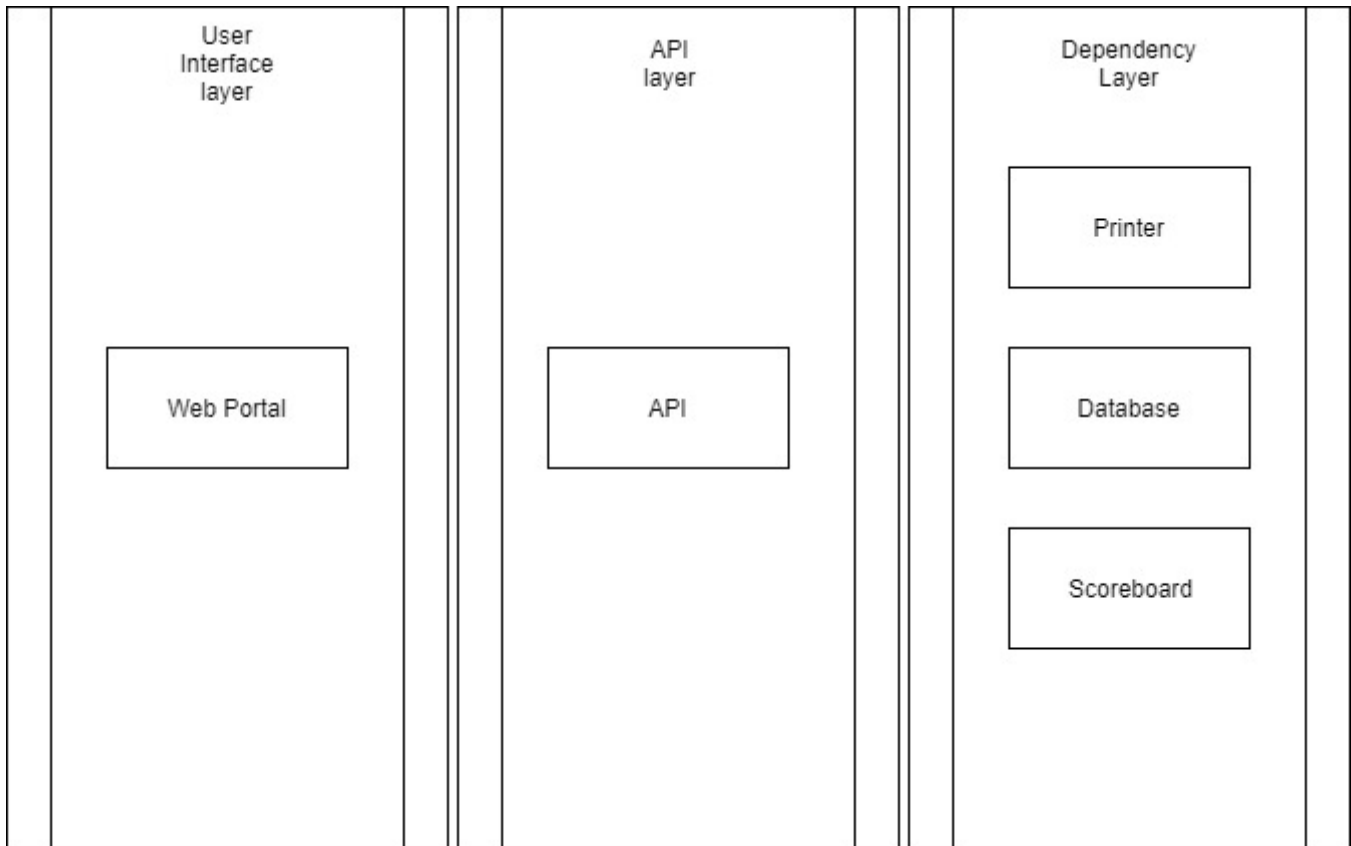
| Name | Return | Authentication level | Definition |
|----------------------|-----------------|----------------------|---|
| GET /user/endSession | Status code 200 | Admin or Scorer | Ends the current session that the user has going. Essentially a log out endpoint. |
| GET /user/getSession | Status code 200 | Admin or Scorer | Connects the user to the current meet that is happening. |
| POST /login | Status code 204 | Admin or Scorer | Allows a user to login. |

3.7.8 Dependency viewpoint

This system implements a three layer architecture in its design. This three layered system allows for modularization of each layer, allowing upgrades and changes to be more easily implemented.

The three layered architecture contains the user interface layer, the API layer, and the dependency layer.

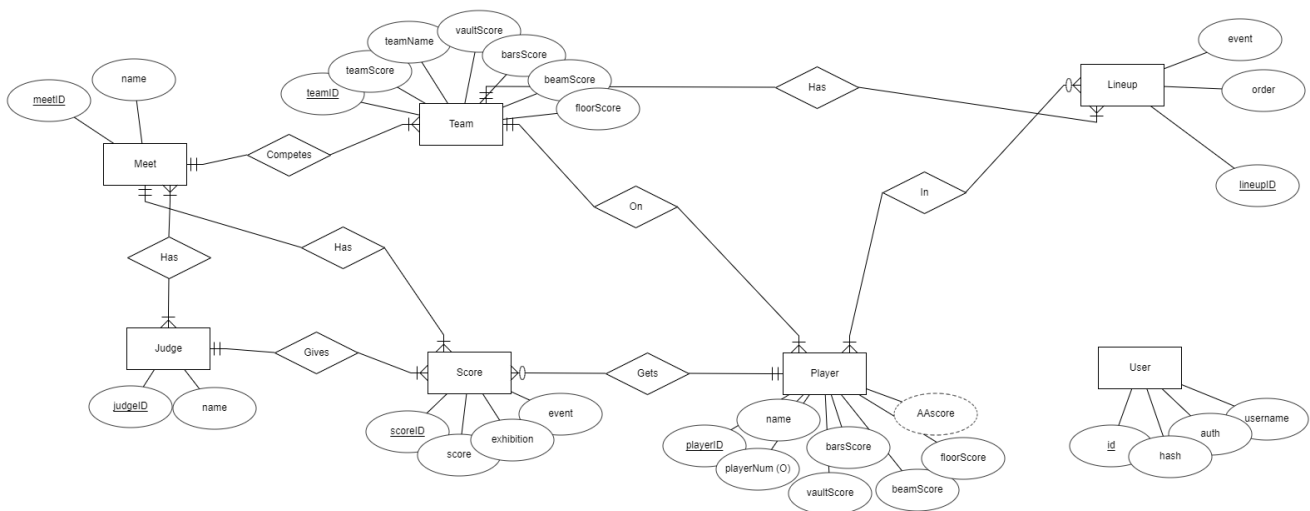
- User Interface Layer: This is the layer that the user interacts with directly. This layer contains functions to interact with the API to get and post scores, printer, and scoreboard.
- API Layer: This is the layer that performs logical operations, accesses the database, and communicates with the printer and scoreboard.
- Dependency Layer: This layer contains the database as well as end systems such as the printer and scoreboard.



3.7.9 Information viewpoint

This sections contains information about our data structure and how the objects are connected.

Entity Relationship Diagram



3.7.10 Patterns use viewpoint

Due to the use of client-server, the common patterns that will be used are endpoints. These endpoints add a layer of abstraction that allows for developers to abstract common processes into easily callable functions.

3.7.11 Interface viewpoint

The interfaces that will be used will all be client side due to the server being an abstraction of processes. There will be three major interfaces, meet set-up, team set-up, and gymnast scoring. The meet set-up interface will allow for moderators to choose the number of teams and judges at the current meet. Next, the team set-up will allow for moderators to enter in data for each of the teams participating in the meets. Lastly the gymnast scoring interface will allow for either moderator or judge level accounts to send judges scores to the API.

Scoring Software 10.0

Meet Lineup Print Score

Number of Teams:

1

Accept

Team Names

OSU

Accept

Number of Gymnasts for OSU

2

Gymnasts for OSU

Mary Jacobsen

Nikola Jokic

Fig. 1. Create Meet Page

Fig. 2. Team set-up

Fig. 3. Scoring interface

3.7.12 Structure viewpoint

The structure of this project can be exemplified by a client-server architecture. In essence, a client-server architecture is a synchronized set of commands that are abstracted away from user to create ease of use for users while creating separation of key components to enhance portability and security. Our system will revolve around a central API that takes commands from a web portal that can be easily accessed.

3.7.13 Interaction viewpoint

Whenever the user wants to access data they must use the web portal to request information from the API. The API then acts as an intermediary to get and set data for the user and returns appropriate information.

3.7.14 State dynamics viewpoint

Our program will have a multitude of modes that will be selected during the initial meet set-up. Once the meet has been set-up it will be impossible to change the number of teams or the number of changes. The state will only be changeable during the beginning set-up phase.

3.7.15 Algorithm viewpoint

This viewpoint details the algorithms used in the design.

- **Team Event Score:** The algorithm for getting team event scores works by iterating over the event lineup of that team. The top 5 scoring gymnasts have their scores added up and the total is returned.
- **All Around Score:** To ensure we don't qualify exhibitionists in the all around scoring, the all around score algorithm checks the exhibition flag before adding up scores for the all around event.
- **Average Player Score:** For each event there can be multiple judges. The scores from the judges are averaged and the result is the gymnast's final score.

3.7.16 Resource viewpoint

This section will contain every npm module we used and the purpose for why it was used. NPM Modules

- Express
a web framework for node with useful routing and middleware features
- Morgan
HTTP request logger middleware for node.js
- Body-Parser
Used to parse incoming request bodies
- Express-handlebars
Used with express to create dynamic page content
- MySQL
Used to make queries to the SQL database
- Client-sessions
Used to keep track of session information such as current meet ID
- Cookie-parser
Used to parse JSON data from client stored cookies
- JQuery
Used for DOM Tree traversal and manipulation
- JS-cookie
Used to handle client side cookies
- Handlebars (client-side)
Client side handlebars module

4 TECH REVIEWS

4.1 Thomas Tech Review

1. Android App Development Stretch Goal

Android Studio

Android Studio is the most popular integrated development (IDE) for Android developers. It is built on the IntelliJ IDE and is the official development tool for Android applications [1]. Google designed Android Studio specifically with Android development in mind and released Android Studio in December 2014 to replace Eclipse as the official Android development tool [1]. Android Studio works best with Java, so the Java Development Kit will also be required (JDK) to compile. Some useful features Android Studio has are a rendering of how the layout of the application would visually appear, widgets (basic application features) that we can drag over to add to the application, and an Android emulator to run our application [1].

Android Native Development Kit (NDK)

Although Java may be the most popular Android development language, it is not the most ideal for performance particularly for games [2]. One engine that allows developers to program Android applications in C or C++ is Android NDK. The source code written in C compiles directly into native machine code (ARM or x86) [2]. This can allow a developer to get more performance out of Android devices. Some features NDK has are CPU profiling (simpleperf), Visual Studio integration, and a variety of application programming interfaces (APIs) [2].

Xamarin and Other Frameworks

Using software frameworks to develop applications would give us a significant head start. Xamarin is a very popular framework that allows us to develop for several mobile platforms including Android, iOS, and Windows [3]. If our group were to develop both an Android and iOS application, Xamarin can be extremely useful. It allows us to use C with the same IDE and API's across different platforms [3]. Instead of writing an Android application in Java and writing an iOS application in Swift, we can code using only C. Using Xamarin allows us to use Visual Studio's powerful features such as debugging on an emulator, debugging, and code completion [3].

2. iOS App Development Stretch Goal

Xcode

Xcode is an IDE available on macOS containing development tools for writing macOS, iOS, watchOS, and tvOS software [4]. It supports many programming languages including Python, C, C++, Objective-C, Objective C++, Java, and Swift [4]. Currently, Swift is the most popular language because it was made specifically for iOS and macOS software development. Xcode contains the bulk of Apple's documentation for software developers and an application to build graphical user interfaces (GUIs) [4]. Since Xcode was developed by Apple from the ground up for iOS development, this is likely the easiest route to take for pure iOS application development. The main disadvantage to Xcode is we will need a Mac unless we use third party solutions.

React Native

Since Xcode will not work on Windows computers (without virtual machines), alternatives such as React Native can be used. With React Native, we program applications using JavaScript [5]. The fundamental user interface (UI) building blocks can be assembled together using JavaScript and React [5]. Many popular applications such as Instagram, Facebook, Skype, and Pinterest were developed using React Native. A handy feature of React Native is instead of

recompiling, we can immediately test changed code. Snippets of code written in Swift, Java, and Objective-C can be used to optimize parts of the application.

Apache Cordova

Cordova is a cross platform framework available on Windows that allows to us develop for both iOS and Android. One disadvantage of these cross platform tools, however, is that they can be much more complex to use compared to native solutions such as Xcode and Android Studio. Unlike Xamarin, Cordova uses HTML5, CSS3, and JavaScript for application development [6]. The advantages of Cordova are cross platform development (potentially cutting development time in half) and it is easy for web developers to get into. The application code resides in a web application making it easy to review changes [6]. Plugins provide an interface for Cordova and allows Cordova to connect native components [6]. Other plugins can provide our application with access to certain device hardware components and software components such as battery, camera, location, etc [6].

3. Display Output Interfaces

High-Definition Multimedia Interface (HDMI)

HDMI is an interface that transmits video and audio data [7]. It was developed by 7 electronics companies (Sony, Hitachi, Thomson (RCA), Philips, Panasonic, Toshiba, and Silicon Image) [7]. The latest specification for HDMI is 2.1 which can support 4k resolution at up to 120Hz refresh rate [7]. The more commonly found and supported 2.0 specification still supports up to 4K resolution at 60Hz. HDMI cables are for very affordable and generally range between a price of 6 and 15 dollars depending on the length of the cable. Generally, HDMI cables can support a video signal reliable at up to 50 feet [7]. In addition, since HDMI is so widely adopted, both the display boards and computers should have HDMI ports. HDMI cables are also backwards compatible with older specifications [7].

DisplayPort (DP)

DP is a digital video interface developed that can also carry audio and USB data [7]. It was developed by the Video Electronics Standards Association (VESA), a consortium of manufacturers, and debuted in 2006 [7]. A passive copper DP cable can support a 4k video resolution over a length of 6.5 feet [7]. If a longer cable length is required, the standard can support a maximum of 2560x1600 video resolution at a length up to 50 feet [7]. This can come in handy if a long cable length is needed to connect the computers to the display boards. Active copper DP cables can carry a 2560x1600 resolution over 65 feet and fiber DP cables can be hundreds of feet long [7]. Another handy feature for DP cables is that a single DP interface can support up to 4 displays at up to 1080p [7]. This means potentially we can use only two DP cables for the six display boards in Gil Coliseum.

Digital Visual Interface (DVI)

DVI is a digital video interface developed in 1999 by the Digital Display Working Group (DDWG) [8]. It is most closely associated with personal computers and monitors designed for computers [8]. Almost every personal computer video card has at least one DVI output port. The most versatile type of DVI cable is the DVI-I cable [8]. It is capable of transmitting either a digital to digital signal or analog signal to signal [8]. A single link DVI-I cable can transmit a video resolution of 1920x1200 at up to 60Hz while a dual link DVI-I cable can transmit a video resolution of 2560x1600 at up to 60Hz [8]. In general, DVI cables do not perform over long lengths as well as DisplayPort and HDMI [8]. Typically, DVI cables can maintain a signal at up to 25 feet [8].

References

- [1] A. Mullis, "Android Studio tutorial for beginners", Android Authority, 2017. [Online]. Available: <https://www.androidauthority.com/android-studio-tutorial-beginners-637572/>. [Accessed: 02- Nov- 2018].
- [2] A. Mullis, "Android NDK Everything you need to know", Android Authority, 2017. [Online]. Available: <https://www.androidauthority.com/android-ndk-everything-need-know-677642/>. [Accessed: 10- Nov- 2018].
- [3] "Best 10 Android Frameworks for Building Android Apps", Medium, 2018. [Online]. Available: <https://medium.com/@MasterofAndroid/best-10-android-frameworks-for-building-android-apps-44ceb3756880>. [Accessed: 02- Nov- 2018].
- [4] T. Klosowski, "I Want to Write iOS Apps. Where Do I Start?", Lifehacker, 2018. [Online]. Available: <https://lifehacker.com/i-want-to-write-ios-apps-where-do-i-start-1644802175>. [Accessed: 02- Nov- 2018].
- [5] R. Vries, "How To Develop iOS Apps On A Windows PC LearnAppMaking", LearnAppMaking, 2018. [Online]. Available: <https://learnappmaking.com/develop-ios-apps-on-windows-pc/>. [Accessed: 02- Nov- 2018].
- [6] M. McInerney, "Architectural overview of Cordova platform - Apache Cordova", Cordova.apache.org, 2018. [Online]. Available: <https://cordova.apache.org/docs/en/latest/guide/overview/index.html>. [Accessed: 02- Nov- 2018].
- [7] M. Brown, "HDMI vs. DisplayPort: Which display interface reigns supreme?", PCWorld, 2018. [Online]. Available: <https://www.pcworld.com/article/2030669/laptop-accessories/hdmi-vs-displayport-which-display-interface-reigns-supreme.html>. [Accessed: 02- Nov- 2018].
- [8] "A Complete Guide to the Digital Video Interface", Datapro, 2018. [Online]. [Accessed: 02- Nov- 2018].

4.2 Zech Tech Review

Database

The system will be incorporating an API for potential stretch goals. As such, we must incorporate a database to save all the gymnastics meet data. The options for the database that we will be examining are MySQL, MongoDB, and Firebase.

Firebase

Firebase is a mobile and web application platform. One feature of the platform is the Firebase database. The database features a NoSQL, real-time database, which allows for the data to be updated without the page needing to be refreshed.[1] The major pros to this database is that its very easy to use and the real-time data synchronization is exactly what we need with multiple judges all entering scores at the same time. The database also being cloud-based means that large amounts of traffic would place the load on Googles servers rather than the server at Gill. This is not as much of an advantage for our system however as the only people on the server will be the judges and their helpers. Because there is data being collected on the athletes however, their names, athlete ID, and scores, our database must be run locally to avoid any potential legal issues. Additionally, Firebase is a Google service and there is a cost associated with the product. If we were to use Firebase as our database a monthly fee would be introduced to the costs.

MongoDB

MongoDB is an open-source NoSQL or non-relational database, very similar to Firebase. It features a good scalable database system that can be hosted locally on the Gill servers. MongoDB allows a lot more flexibility, such as custom queries, when compared to Firebase at the cost of MongoDB being more complex.[2] With MongoDB being a non-relational database, it simplifies horizontal scaling, which is the addition of additional nodes or access points. This specific project would benefit from horizontal scalability as a potential stretch goal is to create a mobile app to access data from the database. The major cons to MongoDB would be that someone in the group would need to learn to use it.

Additionally, the server at Gill runs everything using MySQL so implementing a MongoDB database would annoy to the people who work IT in Gill Coliseum.

MySQL

MySQL is a Structured Query Language (SQL), meaning it organizes the data into tables featuring standard rows and columns. Each row is given an identifier that ties all the data in the row together. This type of database would work perfectly with the gymnastics scoring software as all the data can be associated with the athletes unique identifier. The databases managed by the IT team in Gill also use MySQL so it would be better for us to follow their standards as they will be the primary ones using it.

Scoreboards

The scoreboard is the most important piece of hardware in this project because it is the focal point of the software we are going to develop. There are many factors that go into a scoreboard such as size, type of scoreboard, and costs, but the main focus will be on the methods data can be sent to the scoreboard.

Nevco

One popular brand of scoreboards are the Nevco scoreboards. They offer a very wide variety of different sized scoreboards. One thing they emphasize is their easy to setup and use systems. Their scoreboards come with either a wired or wireless console to control the scoreboards. This means that connecting the scoreboard console to the scoring software would be very difficult and require us to build a custom adapter. To further add to the difficulty of this option, there is very little documentation on how data is interpreted by the scoreboard.

Daktronics

The daktronics scoreboard is the one currently inside Gill. Daktronics scoreboards are often cheaper than Nevco. They offer nearly similar features but have less options to choose from in terms of sizing and color. The main reason for choosing Daktronics however is the documentation online, as their website features manuals on every scoreboard. Daktronics scoreboards can take in serial data as well as XML data to display information. Another reason to use Daktronics is because the previous software is designed for their boards and we have templates for exporting the data.

Data Format

Because the main component of the system will be the collection and storage of data, the format we use is important. The different data formats have different methods of organization as well as special attributes to make searching easier. The three formats we will examine are plain text, JSON, and XML.

Plain Text

Plain text is a data format that is simply a long string. This string of data consists entirely of characters and so the data must be parsed and interpreted manually. Plain text is often used with loose data and it can be used to solve incompatibility issues involving endianness. Because we want our system to pair data such as names and scores, plain text is a poor option. Plain text offers no real advantages and instead adds complexity to create and interpret the data.

JSON

JSON, short for JavaScript Object Notation, is a human readable data format derived from JavaScript. Should we choose Node, a javascript based framework, the implementation will be extremely easy. JSON is however a language independent format, meaning any language can use this data. Because of its popularity many programming languages include functions to generate and parse JSON data automatically. JSON data is very concise and often takes up much less space when compared to other formats such as XML.[3] This reduction in size generally leads to faster transmission and processing making it a very fast format.

XML

XML is a markup language and is another viable option for this project. XML organizes data with opening and closing tags. One of the main advantages of XML is their metadata support, which is the ability to add descriptions such as title, author, etc., to name-value pairs.[4] XML also features a variety of schema or rules the data must follow, such as order of organization. Many browsers also commonly feature a XML viewer which creates visual representations of XML data, typically in the form of a tree-diagram. Because the primary access point to the data will be through a web application, this build in feature would be beneficial in creating a good format for the data.

REFERENCES

- [1] Sonam Khedkar and Swapnil Thube. Real time databases for applications, 2017.
- [2] David Connelly. Firebase vs mongodb vs mysql.
- [3] Nurzhan Nurseitov, Michael Paulson, Randall Reynolds, and Clemente Izurieta. Comparison of json and xml data interchange formats: a case study. *Caine*, 9:157–162, 2009.
- [4] RestfulAPI. Json vs xml, 2018.

4.3 Mary Tech Review

Containers

Docker

Docker was developed from LXC, which allowed multiple Linux systems to run on one computer isolated from the other systems. Docker is an open source project that was made to make an LXC container for a single application. Docker is now a Linux container runtime environment that can make, deploy, and run a container. [11] As of recently, Docker is getting a lot of attention. There are many reasons to use Docker. Docker is the current industry standard. Docker is the main reason containers are currently so popular in the software engineering industry. Docker is lightweight. Instead of requiring an operating system for an application, Docker shares the operating system kernel with the computer. Docker is secure and trusted. The application will be safe inside the Docker container. [5] Docker provides all the benefits of an extra layer of abstraction without very much overhead at all. Docker also allows for portable shipment of applications. Docker is focused primarily on the application as opposed to the machine or other things. Another convenient part of Docker is that it provides source control which doesn't affect the decision of using it for this project because using Github is required but it is a very good feature anyway. [1] Because of all these features Docker provides and because it is trusted by so many, Docker seems like the safe choice.

Open Shift

Open Shift is an application container platform. Open Shift is a different option for the use of a container but it still can utilize Docker containers. Open Shift is not as well known or as commonly used as Docker but it does have some appealing features. One of the more exciting parts of Open Shift is that it is built to work along side with Github which is ideal for the project since Github must be used. Not only that, most software developers are very familiar and comfortable with using Github so it makes the learning curve of Open Shift not as steep. Open Shift makes it easier to deploy applications because applications can be deployed using git push or pushing a button. Another helpful feature of Open Shift is that it has templates for a few platforms and languages including Node and Python. [9] Open Shift is free to use for just one project so that would suffice. After some research, it does not seem like the features added by using Open Shift are worth the extra time it would take to set it up. The Open Shift features are meant to save time but they may not save enough time to choose this option.

Kubernetes

Kubernetes is a Google open source project that manages containers. Using a container is not one of the requirements for this project but a container may be used if the benefits outweigh the work of setting it up. Containers at Google, like other containers, provide an application environment that does not change, as well as the ability to run applications anywhere, and a layer of abstraction from other parts of the machine and other applications. Containers also have a very small over head and do not take up much memory. Kubernetes automates the process of managing a container while still providing all the benefits of using a container. Kubernetes runs best on Google Cloud Platform because it is integrated in but can be run elsewhere. Kubernetes' "goals are to build on the capabilities of containers to provide significant gains in programmer productivity and ease of both manual and automated system management. [6]

Frameworks

Node.js

This project will definitely require using a framework due to time constraints and because it is more efficient. Node.js is an open source, server side run-time environment built on the Google Chrome's JavaScript V8 engine. Node.js works on the main operating systems (OS X, Windows, and Linux). Node.js is event driven and asynchronous. That will help with waiting for data to be returned. Node.js is also very fast and won't slow down the code execution. Node.js is single threaded but because it is event driven, it is still scalable and will suffice for this project. The Node.js applications return the data in pieces so it never has to buffer any data. Node.js also partners well with JSON APIs which might be an option for this project. Node.js is also very trusted. Some of the companies that use Node.js are Uber, eBay, Microsoft, and Paypal. [12]

Django

Django is a web platform based on Python that is open source and free to use. The three main reasons developers usually choose to use Django and why we might choose it is its speed, security, and scalability. Django supports very fast development. Django also helps with having sufficient security. And because of Django's superior scalability, "Some of the busiest sites on the Web leverage Django's ability to quickly and flexibly scale." [3] Django has some added features that would help specifically with this project. One of the more helpful extra features that Django includes is that it takes care of authentication. That is important for this project since it requires different levels of authentication to allow

only authorized staff to input scores. Django is also a well trusted, popular, and versatile framework. "Companies, organizations and governments have used Django to build all sorts of things from content management systems to social networks to scientific computing platforms." [4] Django would definitely work for this project so it really depends on if Python is chosen as a language since the framework and language decisions really depend on each other.

Beego

Beego is an open sourced application framework built on the Go language. Beego is one of the more commonly used frameworks for the Go language which is growing rapidly in popularity. There are four main reasons why developers would choose Beego. Beego is easy to use, intelligent, modular, and has high performance. Beego has automated testing, packing, and deploying. Beego has some interesting monitoring features as well. Beego can monitor memory and CPU usages as well as other things. Beego works for most types of applications. "With powerful built-in modules including session control, caching, logging, configuration parsing, performance supervising, context handling, ORM supporting, and requests simulating. You get the powerful foundation for any type of applications." [2] Applications built with Beego can also handle lots of traffic which does not specifically apply to this project unless the IOS and Android applications stretch goal is completed. If the language Go is decided upon, the Beego framework would be a good choice for this project.

Languages

Javascript

This project will require the use of a language and whatever language is chosen will affect which framework will be chosen. Javascript is used for so many web applications and would be an easy choice with the use of Node.js. There are many aspects of Javascript which make it an appealing option. Javascript is easy to use because a large amount of documentation exists for it. Javascript is built into the browser which eliminates set up time. Javascript is very widely used. Javascript is used in web browsers, server side, mobile, desktops, games, and more. The reason for this is because Javascript is "the native language of the web browser" and not necessarily because it is the best choice. [10] Javascript does work really well for forms. If Javascript is used correctly, it can enhance a web page and make it friendlier for the user. [13] Overall, Javascript makes sense to choose because it is so well known and used and because of the quality of Node.js.

Python

Python is considered an easy to use language by most developers. "Python is an interpreted, interactive, object-oriented programming language. Python incorporates modules, exceptions, dynamic typing, very high level dynamic data types, and classes. Python combines remarkable power with very clear syntax." [8] Python also has many libraries that can be included. Python is extremely versatile and is used in many different types of applications. Due to the large standard library, Python is very good for working with strings unlike many other languages which makes code much simpler. Also included in the standard library are ways to easily use internet protocols such as HTTP and XML which will be needed for this project. [8] Python also has very good documentation which is so important for finding the best ways to use the language and solving problems. Overall, Python seems like the easiest language to use.

Go

Go was created at Google and is an open source language. "Go was actually developed as an alternative to C++." [7] Go was designed to better serve software engineers and not to research programming languages. Go is somewhat similar in syntax to C and is therefore easy to learn for developers who already know C. Go has many strengths since it was built to solve current software engineering problems. Go is statically typed which C/C++ developers can really get behind. Go has a very fast compilation speed and execution speed which was one of the big problems that the language was created to solve. The code compiles directly to machine code which not only makes the execution speed very fast, but it also makes Go portable. Go has automatic garbage collection so pointers don't need to be freed unlike C. Go has many good libraries that can be included that make things easier for the developer such as HTTP client and server implementations, SQL databases, and JSON. "JSON is treated as a first class member of the standard language." [7] Go would be a good choice because it is based off C with the typical complaints about C fixed and because it is an exciting language to learn.

Containers, Frameworks, Languages Conclusion

After researching the three pieces that relate to my role in the 10.0 software project and each of the options for each piece, the reasons for choosing each of the options are far more clear. For containers, I recommend using Docker because it is lightweight, secure, and trusted. I recommend using Kubernetes to manage the Docker container because it is easy to use and automates much of the process. For the frameworks, it depends on which language is chosen because Node.js, Django, and Beego all seem usable. For languages, I recommend using Javascript because it has a good reputation, would be able to accomplish what we need, and has the most documentation out of the three languages. Along with javascript, I recommend using Node.js as the framework because it is the most trusted framework for Javascript.

REFERENCES

- [1] Aater Suleman. <https://www.airpair.com/docker/posts/8-proven-real-world-ways-to-use-docker>.
- [2] Beego Framework. <https://beego.me/>.
- [3] Django Project. <https://www.djangoproject.com/>.
- [4] Django Project. <https://www.djangoproject.com/start/overview/>.
- [5] Docker. <https://www.docker.com/resources/what-container>.
- [6] Google. <https://cloud.google.com/containers/>.
- [7] Kanishk Dudeja. <https://hackernoon.com/the-beauty-of-go-98057e3f0a7d>. Hackernoon, 2017.
- [8] Python. <https://docs.python.org/3/faq/general.html>.
- [9] Red Hat. <https://www.openshift.com/products/features/>.
- [10] Richard Kenneth Eng <https://medium.com/javascript-non-grata/the-five-top-reasons-to-use-javascript-bd0c0917cf49>. 2016.
- [11] Roderick Bauer. <https://www.backblaze.com/blog/vm-vs-containers/>. Black Blaze, 2018.
- [12] Tutorials Point. https://www.tutorialspoint.com/nodejs/nodejs_introduction.htm.
- [13] Stephen Chapman. <https://www.thoughtco.com/why-javascript-2037560>. ThoughtCo, 2018.

4.4 Nick Tech Review

INTRODUCTION

Our project is centered around design competitive scoring software for the Oregon State University Gymnastics Team. The OSU Gymnastics Team has many ideas in mind for how the software should work and has asked the team to create a new interface that is specifically designed to work at Gill Coliseum where the gymnastics meets are held. My role in this project is to work on the security of the application that will be built, for the software needs to be secure in order to ensure that the scores that are submitted to nationals are correct and were accurately recorded. If the software does not accurately keep score then it would lead to scores being taken by handed then double checked by the NCAA which would cause both a poor fan experience and take a lot of manpower to keep track and double check each score. Some major security concerns that need to be handled are protection of data, protection of user accounts, and lastly securing data for being sent across the internet. The three aspects this tech review will cover are hash functions, encryption ciphers, and transfer protocols.

HASH FUNCTIONS

The main purposes for hash functions are to store passwords, and due to the necessity of administrator accounts to ensure that data can only be used by authorized individuals. To begin, the purpose of a hash function is to manipulate data into a different string of a set length, usually 128, 256 or 512 bits. By using hashing, programmers can hide resources and reduce the size of the resource by hashing. For the password example, a user will enter a password which will then be sent into the hash function and the result is sent to the server; from there, the server checks if the hash password stored is the same as the hash received. One issue with hashes however, is that a hash of two separate strings can return the same output which is known as a hash collision. Hashes have multiple uses in security, but there are two major type of hash functions, cryptographic and non-cryptographic [?].

Cryptographic hash functions

The major difference between cryptographic and non-cryptographic hash functions are the end goal of the hash. For cryptographic hash functions, the end goal is to create a hash that can not be easily worked back into the original string; for example, if a user inserts the password `password`, hashes the password to get a new string

`5BAA61E4C9B93F3F0682250B6CF8331B7EE68FD8`, and from there sends that hash over the internet it will be very difficult for an adversary that would want to figure out the password to successfully work back the hash into the original password. The ability for a hash function to create a string where it would be unfeasible to recreate the original string is what separates cryptographic hashes from non-cryptographic hashes [?].

SHA-2

One of the most well known and secure hash functions is SHA-2. It was developed by the National Security Agency, or NSA, of the United States of America. SHA-2 works by implementing numerous bit wise operations in order to mutilate and differentiate data [?]. The SHA-2 algorithm takes in 64 characters at a time, and separates those 64 characters into eight blocks each containing eight characters. From there SHA-2 performs bit wise operations, operations where the characters are turned into their binary forms and logical operators are used with them, on all of the blocks and uses the outcomes of those bit wise operations to create the new string [?]. A quick example is that for a message with blocks A, B, C, D, E, F, G, H the new block G is found by taking $G = (E \text{ AND } F) \text{ XOR } ((\text{NOT } E) \text{ AND } G)$. This shows the amount of operations that are done on each block in order to completely change the output, for this was done just to find one of eight blocks, but due to the amount of operations done SHA-2 is seen as a slow and expensive hash due to the amount of time and the resources it needs to compute a hash.

Non-cryptographic hash functions

Unlike cryptographic hashes, non-cryptographic hashes do not primarily care about the ability to hide the original input, and instead care about reducing the number of hash collisions that can occur. Now, since the main goal of non-cryptographic hashes is to stop collisions, they tend to be more efficient and quicker to run then their cryptographic cousins. This causes the primary use of non-cryptographic hashes to be used in local programs, or programs where the hash does not get sent over the internet. Overall non-cryptographic hashes can be seen as weaker than cryptographic hashes, but they make up for it with speed and simplicity [?].

MurmurHash

MurmurHash is a non-cryptographic hash that takes a message and outputs a 64 bit hash. It does this by taking the first 32 bits of the message and essentially moving the bits either right or left x-places, think of the number 8 which is `1000` in binary. If one was to rotate the number 8 left by two bits the result would be `100000` which is equal to 32. In general MurmurHash is a very simple hash function that can quickly create a hash that would be good for local use; however, this hash can easily be reversed to find the original message making it weak for being sent over unsecured channels [?].

ENCRYPTION CIPHERS

Encryption will be used when passing data from the client web app to the API server, nad ensuring that the data is protected. The basis of encryption is to take a message, and perform an operation on each character in order to

completely change the message. Encryption has been used for centuries and one of the earliest encryption ciphers can be attributed to the ancient Romans [?]. Continuing, the only goal of an encryption cipher is to create a cipher text that cannot be deciphered by an adversary, but can be deciphered by a friendly agent. There are two major types of ciphers in modern cryptography, stream ciphers and block ciphers, and block ciphers are primarily used due to stream ciphers needing to know the exact length of a message [?]. Block ciphers on the other hand are not limited to knowing the length of message, and instead break the message up into x-bytes and encrypt each block as it goes. Moreover, there are many attributes that are desired in an encryption cipher such as self healing properties, the self healing property is a property where if a bit gets flipped on accident during sending, it will only hurt part of the encryption and not the whole thing, and avalanche effect, where if even a single character is different then at least half of the encryption cipher is different [?].

ECB

ECB, Encryption Cipher Block, is one of the first block ciphers that was created in modern cryptography, and as such it is now outdated and it is highly warned against using this cipher [?]. The way that ECB works is simple, it takes the first block of the message and encrypts to start the cipher text. It proceeds to do the same process for each of the blocks and appends the blocks to the ciphertext. This creates a ciphertext that can easily be decrypted by running the decryption algorithm of the cipher for each block. One upside to ECB is that it can be run in parallel, meaning that each block can be decrypted at the same time, and has self healing properties, for the only block effect by a flipped bit would be the block with the flipped bit [?]. However, where ECB fails is the avalanche effect. The major problem with ECB is that there is no difference in a message of the same thing over and over again. So, if an adversary knew the encryption for password, then they could easily find every instance of that block. And due to that major failure of the avalanche effect ECB is considered one of the worst block ciphers to use.

CTR

CTR mode, which stands for counter mode, heavily focuses on two things: the ability to run in parallel, and the avalanche effect [?]. CTR mode works by first creating a random string known as an initialization variable, or IV. From there, it encrypts the IV then runs a bit wise XOR with the plaintext in order to get the first ciphertext block [?]. For the second ciphertext block, it adds one to the IV in order to get a completely different IV, encrypts the IV, and then runs a bit wise XOR with the plaintext to get the second ciphertext block. So, for each block CTR mode simply adds $N - 1$ to the IV, encrypts the IV, then XOR the plaintext and encrypted IV, and once all blocks have been encrypted CTR mode appends the un-encrypted IV to the begging of the ciphertext [?]. Now, the main reason for the IV is to stop the same message from having the same ciphertext; moreover, the IV also helps to create an avalanche effect since adding one to the IV will result in a very different binary string that will cause a message composed of the same blocks to come out with very different results. Another big advantage of CTR mode is that it is considered a forward only encryption, which means that a decryption is not needed to retrieve the plaintext, instead the decryption process encrypts the IV and uses XOR with the ciphertext to then get the plaintext, and because of the forward encryption CTR can easily be run in parallel by simply sending in the cipher text, the IV, and the amount to increment the IV. Overall CTR mode is considered one of the best block ciphers to use when resources, such as storage space and processor speeds, are a constraint.

CBC

The last mode that will be talked about is CBC, or Cipher Block Chaining, which was developed with avalanche and self healing properties in mind[?]. CBC works by first creating a random IV, much like CTR mode. Then, CBC uses XOR with the first plaintext block and the IV, and then encrypts the result to get the first ciphertext[?]. For the second block, CBC once again uses XOR on the plaintext, but instead of using XOR with the IV, CBC instead uses the first ciphertext as the other end of the XOR, and after the XOR the result is encrypted to get the next ciphertext block[?]. So the encryption algorithm for any ciphertext after the first is to XOR plaintext N with ciphertext N - 1, then encrypt N the result, and once the message has been fully encrypted the IV is appended to the beginning of the ciphertext[?]. Decryption works very similarly to encryption for CBC, first it decrypts the ciphertext, then it takes the XOR of the decrypted ciphertext and the previous ciphertext to get the plaintext message[?]. Due to the amount of chaining that CBC does, it is a very secure encryption method. CBC also shows avalanche effect because of the random IV in the beginning, and since the IV effects the first ciphertext which effects the second ciphertext and so on, it creates a completely different ciphertext at each stage. Next, CBC has self healing properties due to its decryption algorithm, for the mistake in a ciphertext would only persist for the block the mistake was made in, and the block after it. CBC is a very strong encryption block cipher and is the current standard for most security protocols.

TRANSFER PROTOCOLS

Transfer protocols are necessary for sending information over a connection. The base purpose of a transfer protocol is to send a message from point A to point B; however, more modern transfer protocols have started to care about making sure that each part of a message arrives, and that the message is going to the correct receiver and not being intercepted. With transfer protocols, the modern internet would not exist as they are a fundamental structure that the internet was built upon.

UDP

UDP, or User Datagram Protocol, is the most basic transfer protocol available. It simply takes the message and sends it. It does this technically by getting the IP address of the receiver, then formats the header of the file to be sent to that IP address[?]. If the message is not received by the recipient, UDP does nothing to ensure that the packets are received or that the correct message is received either[?]. The best use for UDP is when data needs to be sent over a secure line in an efficient manner that has no excess.

TCP

TCP, or Transmission Control Protocol, can be viewed as the more secure bigger brother of UDP. TCP works by sending a message to the receiver followed by the receiver sending a message back to the sender that they received the message[?]. This is done through the use of acknowledgement, or ACK. To start the sender sends the message and starts a timer, and when the receiver gets the message, it will send back an ACK to the sender that means the message was received[?]. If the message never makes it to the receiver then after a set amount of time passes, TCP assumes the message was lost and will send it again and will repeat this process until the receiver sends back an ACK[?]. Overall, TCP cares a lot more about the package arriving to the correct place, and that the entire message gets sent.

HTTP

HTTP, or Hyper Text Transfer Protocol, is a request-response protocol that is more commonly used with servers[?]. What request-response means, is that a client will send a request to a server, and the server will fulfill the response with data[?]. The first thing that happens is the client will send a request to the web server of either GET, which grabs data from the server; PUT, which updates data in the server; POST, which creates new data on the server; and lastly DELETE, which deletes data on the server[?]. The client sends one of these four commands, sometimes with a message body that has more information, and the server will then attempt to fulfill the request. The server will process the request and send a response to the client with an HTTP status code. Usually the requests will be processed correctly and send a 200 level code, and the response to the client will have the data or the action request[?]. If the server fails it will send either a 400 level code which indicates that the operation was not allowed, or a 500 level status code which means the server failed to send the data correctly[?]. HTTP was specifically built for the internet and servers, and because of this HTTP is very well optimized to enhance the transfer protocol for data being sent and received from servers.

CONCLUSION

In conclusion, the needs of hash functions, encryption ciphers, and transfer protocols are very important for our project to make gymnastics scoring software. The current plan for the project is to make an API that acts as the server to deal with requests of authorized devices, and because of this the project needs to be very security minded in order to make sure that data isn't stolen or falsified. So, with that in mind the best options for creating a secure API is to use a cryptographic hash function, and in particular SHA-2, in order to successfully hash and securely store passwords. Next, the API will be sending data over wifi which can be captured by another device, so it is important to use a secure block cipher for encryption, and due to the amount of resources our server can have it will be best to use CBC mode. Lastly, since our API server is a server, the best transfer protocol to use is HTTP since it was designed to be used for web servers.

REFERENCES

- [1] Sonam Khedkar and Swapnil Thube. Real time databases for applications, 2017.
- [2] David Connelly. Firebase vs mongodb vs mysql.
- [3] Nurzhan Nurseitov, Michael Paulson, Randall Reynolds, and Clemente Izurieta. Comparison of json and xml data interchange formats: a case study. *Caine*, 9:157–162, 2009.
- [4] RestfulAPI. Json vs xml, 2018.

5 WEEKLY BLOG POST

5.1 Thomas' Weekly Blog Posts

5.1.1 Fall Term

Week 3: I wrote my first draft of the problem statement this week. After that, I peer reviewed someone else's problem statement. Writing my problem statement allowed me to better learn the objectives of my senior project. In addition, peer reviewing a problem statement helped me see common struggles with writing the first draft of the problem statement. Getting a 1000 word count seemed to be a struggle not only just for me. I did not have an in depth talk with my client yet about exactly what they want, so writing a 1000 word problem statement that is supposed to clearly understand the project was very difficult. I also have not written much for a while (especially technical writing), so getting back

into writing was somewhat difficult for me. I plan to communicate with my client more so I can write a better problem statement and understand what they want.

Week 4: During this past week, my group combined our problem statements to make a group final problem statement. I realized some of the others had similar struggles to me on writing the problem statement. However, some people actually had a much better problem statement than me. Getting the opportunity to read my group mates' problem statements helped me learn how to make a better problem statement myself. Some problems were coordinating a time to meet with the client and coordinating a time to communicate about how to finish our group final problem statement. In the end, we managed to figure it all out. Next up is working on our requirements document. We may have to meet up with our client again in order to get a better understanding of the requirements.

Week 5: This week was a slow week. Many of us were busy with midterms and other homework assignments, so our group mostly just planned on how to divide up the work for the requirements document. This will be the first assignment where we truly write an assignment together from the ground up, so agreeing on how to approach it may produce some problems.

Week 6: This week, our group completed our requirements document. This was definitely more challenging than the group problem statement because we had to write everything from scratch. Coordinating who would work on which section was another obstacle. In addition, we also completed our individual tech review drafts today. Personally, this was quite a challenge for me to write 1500 words. Finding topics for everyone to write was hard and researching. It was definitely out of my comfort zone researching technologies because I have never done anything like this before. My group plans to do biweekly meeting with our client and to attend a gymnastics meet in order to get a grasp of how the scoring system works.

Week 7: I finished the final draft for my tech review this week. After the peer edits, I took the criticism to mind and improved my paper. I also made sure to find credible sources. The biggest problem I had is I'm not sure if my topics in the tech review are good enough. My group plans to attend a practice gymnastics meet on the 15th of November to get a better idea of the how the scoring system works.

Week 8: This was a slower week. Our TA told us missing the Gantt chart would cost us 25 points. We added a Gantt chart to our requirement document because the draft we turned it did not have it. We attended a gymnastics meet on the 15th so we could understand the scoring process better. In class, we started to get an idea of what a design document would look like.

Week 9: This was a less active week. There were no assignments due this week, however I have started to plan my design document. Each member has an individual component in the design document that has to cover the pieces of the project that they are responsible for in the project. Our group will have to get together and discuss how we will divide up the project. We will also have to talk about how our system will work and the timeline of our work. To me, it seems like planning so far ahead will be a challenge, but hopefully this should make completing the project easier in the long run.

5.1.2 Winter Term

Week 1: We still have not started coding the project yet. In addition, we still have not made contact with our client since winter break. No one has initiated anything so it will probably be up to us to initiate contact and begin planning. Next week, we will make contact with our client and begin our project.

Week 2: Our group went to the TA meeting this week. I have had some trouble overcoming winter break laziness. However this week, I have begun to review HTML, CSS, and SQL in preparation for the project. Within the next week, our group should be meeting up to work together.

Week 3: Our group started setting up the database for our project last weekend. We completed the poster rough draft and continue redesigning our database schema. In the following week, we plan to start on the front end portion of the project. The only slight obstacle is finding a good time for our group to meet.

Week 4: We have continued programming this week. Along with another group member, I worked on developing some of the web pages with HTML and CSS. Next week, I think I'll start to work on the client side JavaScript.

Week 5: My group is continuing to split between frontend and backend. This week I have continued on HTML/CSS for the webpages and have also worked on client side JavaScript. Next week, hopefully we will have an alpha release ready.

Week 6: We have managed to get the front end and back end talking to each other. Right now we are making changes to our database (we found problems with our original plan) and trying to complete our alpha version. On Monday, we will meet with our client to keep them updated.

Week 7: We are working on catching up database portion so we can start working on security. The main hurdle is completing the database so we can work on other parts of the project. We plan to meet on Sunday.

Week 8: We finished our winter term first draft report. Before that, we did a user study with our client. Overall he was satisfied with our progress. We are working on fixing concerns he had and are continuing to finish our database. We plan to hopefully finish databases and continue on to security and our create meets page.

Week 9: We are continuing work on the back end portion of the project. It continues to be the main hurdle to our progress. Next week, we will work on the final written report and video as well as launching our beta release.

Week 10: This week our group implemented cookies and finishes our create meet page. We also finished our end points. Now we need to work on printing scoresheets, writing our report, and making our video.

5.1.3 Spring Term

Week 1: My group met with the TA this week. We have not met up as a group to work further on the project, but we plan to meet next week on Monday. One concern we have is we are having lots of trouble printing the score sheet (in their specific format) for gymnastics meets. We discussed our difficulties with the TA and she said she would talk to Kirsten. We may have to edit our requirements document and verify it with our client.

Week 2: We signed up for the project expo. In addition, we completed our project recently and we are ready for the code freeze. Next, we need to edit some documents and verify with our client.

Week 3: Our group worked on the document verification document this week. A significant hurdle we had was one of our group members is a gymnast and she had to travel for her competition. Our client was also busy with gymnastics so we were unable to obtain the signatures on time. We hope to get signatures by next week.

Week 4: Our group got our documents verified by our client. We also finished our poster and had our client look at it to make sure they were satisfied with it.

Week 5: We submitted our final poster submission for printing this week. It took a few revisions to match the template, but we managed to get it approved.

Week 6: Our group has been coasting. All major work on the project is done. We are waiting for expo next week. I went to the mandatory class today.

5.2 Zech's Weekly Blog Posts

5.2.1 Fall Term

Week 4: Progress: This week the team met with the client to discuss the minimum specifications of the project as well as features and stretch goals. In addition we met with Gill Colosseum's IT team and got information regarding hardware specifications as well as the current running model. We also discussed a very broad picture design with the client and IT team which included moving from the current software to a webapp as well as the possibility of a mobile app. Problems: Not exactly a problem but more of an inconvenience is that we still don't know exactly how the hardware works, we're waiting on a follow up meeting with someone who knows how the data needs to be formatted and how to send the data to the scoreboards inside Gill. Plans: Going forward the only plans regarding the entire team is to meet with the person who knows how the data gets translated to the scoreboards. My personal plan moving forwards is to brush up on SQL as it is how we plan on implementing the database for the gymnastics scores.

Week 5: Progress: This week was mostly about figuring out how to complete the assignments. Because our project is simple in terms of requirements and there are not a whole lot of choices to make in terms of tech options we met up in class to come up with topics for the tech review as well as how to divide the topics. Problems: Similar to what is mentioned above there is not a lot of information to write about regarding the specific topics required for the writing assignments. Plans: Plans for next week include meeting with the client again as well as another member of Gill Colosseum IT to learn how data gets transmitted currently. We also plan to try and make contact with the creator of the former software, who happens to be a Corvallis resident. Next week we also plan to have the requirement doc and tech reviews in.

Week 6: Progress: This week we completed the tech review as well as the requirement docs. We met with the client about the requirement doc and everything looked good so far. We also met with more IT people at Gill for further information regarding the type of data to send to the scoreboard. Problems: We still don't know what framework we are going to use (Hopefully whoever did the tech review on that has some good ideas) so we will try and decide by this week. Plans: Currently we don't have any immediate plans besides getting more familiar with the things we know we will be using. We also have planned to all go to the mock meet on Nov 15th to get a better idea how the previous system works as well as how the judges will be inputting scores.

Week 7: Progress: Finished the tech review this week Problems: No problems Plans: Plans to congregate tech reviews into single tech review. We're also going to try to decide what framework to use. Next week is a gymnastics meet so we can finally see how the old system works.

Week 8: Progress: Went to the mock meet to see how the old system worked as well as took note of what features they really liked. Problems: none Plan: We have a general plan on how this project will go from start to finish, now we just need to meet with the client/IT people of Gill and ask any remaining questions we have (Planned for next Wednesday)

Week 9: Progress: Didn't do a whole lot with it being Thanksgiving week. Talked about general plans for design doc and the video. Problems: None this week Plans: Plans to do design doc on Monday and start to think about video

5.2.2 Winter Term

Week 1: Progress- Agreed upon TA meeting times Problems- None Plans- Planned to meet with TA on Wednesdays

Week 2: Progress- Began working on code. created database and began API Problems- No problems this week Plan- Fully implement database this week

Week 3: Progress- Plan to implement some end cases for the API by Monday Problems- None this week Plans- Finish some end cases and fill database with test data

Week 4: Progress- Finished a majority of endpoints. Planning to work on implementing the database w docker this weekend. Problems- None this week Plans- Once the database is up we want to start testing endpoints with the database.

Week 5: Progress - Implemented database and many end points. Also began testing. Problems - Lots of issues getting the database working, slowed progress down a little but we're not too far off track. Plans - Once the frontend is more finished we will begin connecting the front and back ends. Until then we will keep adding more end points and create a test suite in Postman

Week 6: Progress: Completed alpha version. Completed lineup and scoring pages with working html, css, js, and working GET, POST, PUT, and DELETE methods. Problems: The structure of the database caused some problems so we re-arranged the database to work better with this project.

Week 7: Progress- Restructured database for better scalability. Reconnected endpoints to work with new database. Problems- None besides being delayed slightly for having to re-structure the database.

Week 8: Progress- Completed the progress report and user study Problems- Still having issues with the docker + foreign keys keeping us from making any real progress. Plans- We are going to try and meet a professor to help us solve the issue with foreign keys.

Week 9: Progress- Completed most of endpoints. Completed top 5 scores from lineup and avg between judges. Problems- Database issues have been resolved. Plan- Printing and security

Week 10: Progress- Basically done at this point, Still don't have a solid solution to printing. Hopefully, Gill IT or Road To National's will provide a better solution. Plan- Finish Beta/Progress reports Problems- Printing is an extremely painstaking task and would take far too many hours to complete with the current method.

5.2.3 Spring Term

Week 1: Progress- Did research on potential method for implementing the printing. Plans- Meeting Monday to work on polishing some features and implement sessions. Problems- No solution to printing

Week 2: Progress- Finished Logins, Sessions, Cookies, and Authentication. Plan- Get ready for expo Problems- None.

Week 3: Progress- Finished all major requirements/deliverables before code freeze Plans- Complete the client verification

Week 4: Completed the poster board and took a team picture and got client signature. Plan: None, we finished Problems-

Week 5: Progress- Submitted final poster to engr.media Plan- none Problems- none

Week 6: Nothing this week.

5.3 Nick's Weekly Blog Posts

5.3.1 Winter Term

Week 2 Progress: We planned out what needs to be done before our week 3 meeting with the TA. Our goal is to write out DB initialization script, and our dockerfile that initializes our sql image and runs our API server. Problems: None so far Plans: Complete the DB initialization script and finish the DB schema. Create the dockerfile that initializes the sql image and runs our API server.

Week 9 Progress: We finally were able to get foreign keys to work and the flood gates have been unleashed. We have finished all of the backend endpoints. Problems: None at the moment, but potential issues with cookies. Plans: Finish the frontend and printing, then we should be ready for beta

5.3.2 *Spring Term*

Week 2 With code freeze on the horizon, our entire team put in a bunch of work to get the project ready for Monday. Problems: Finishing all of the tasks, exporting to csv, and lastly printing out scoresheets. Progress: Got the entire application to a point where we are happy. The only thing we could not figure out was printing scoresheets. Thomas was put in charge of exporting the data to a csv, has yet to finish.

Week 3 This week Mary went to nationals for gymnastics and as such was not able to help too much. Before she left we put in a fair amount of work on to the second draft of the design and requirement documents. The issue with the OSU Gymnastics team going to Nationals is that Michael, our sponsor, is super busy and has not been able to look at our second drafts to sign them. Overall we all put in a good amount of work towards this deadline. Lastly, I finished the largest point assignment of my career by submitting yes.

Week 5 Nothing really exciting happened this week. We finished the poster revisions as a group and turned in the rest of the necessary documentation for Expo. For the future weeks we hope to work on what we plan to present to the people who visit our table at Expo.

Week 6 Not much is actually happening outside of practicing my pitches for expo. I have not really talked much with my group since last week.

5.4 **Mary's Weekly Progress**

5.4.1 *Fall Term*

Week 4 Progress: We met with the client and two Gill IT people to better understand what the requirements are for the project and also to tell them what we could do. We are on the same page and everyone is excited about the project. On Sunday, I created a github repo and added the whole group. On Thursday, each group member emailed me their individual problem statement and I compiled them into one group problem statement. On Friday (today), each group member looked over the problem statement and changed anything that needed to be changed. It is ready to be turned in now. Problems: The client originally didn't think we could meet this week because recruits are here and it is a busy week but we found a time that worked for him so it ended up working out. Besides that it was a smooth week. Plans: Now that we have more information, we will write the requirements and email them to the client and IT to approve next week. We also plan to add everything we've done so far to the github repo since we haven't done that yet.

Week 5 Progress: We are still working on the requirements document. I added my individual and the group problem statement latex source codes and makefiles to the github repo. I also added our TA to the repo. Problems: The only problem we ran into was having enough parts of the project for each of us to have 3 parts for tech reviews. We are still working on this. I believe we've thought of 6 or 7 so far. Plans: We will finish the requirements by Tuesday. We will also split up which parts each of us will do for the tech review. We have another meeting with the client and Gill IT tentatively scheduled for Wednesday to talk about how the scores are displayed.

Week 6 Progress: We finished the first draft of the requirements document and went over it in a meeting with the client and Gill IT. We also gained some more information about the project in that meeting. We researched different options for the project based on the requirements. Each team member finished their individual tech reviews. Problems:

We had a problem finding enough pieces in the project for each group member to have three pieces to review different options for but after multiple brainstorming conversations, we figured it out. Plans: Now that we have researched different tech options, the next step is deciding which option to use for each piece as a group. We will then decide what each group member is in charge of.

Week 7 Progress: Each team member has finished the final draft of their tech review. We have started discussing who will be in charge of what parts of the project. We did not have a meeting with the client or Gill IT this week. Problems: It is hard to decide who will be in charge of which part because we want to make sure everyone has enough to work on and that they get to work on what they want to but I'm sure we'll figure it out. Plans: We want to nail down who will be working on which parts this week. We also want to decide as a team which tech options to choose based on the recommendations in the tech reviews that we wrote.

Week 8 Progress: We made a Gantt chart for the requirements with a timeline. We also all attended an Orange and Black exhibition in Gill for gymnastics yesterday. I was competing but the group (besides me) shadowed Gill IT to see how the scoreboards and software are used during a meet. Problems: I still need to be updated on what was learned since I was doing gymnastics. Plans: We plan to start on the end of fall term checklist as much as we can during Thanksgiving break and start on the design.

Week 9 Progress: This week was somewhat slow because of Thanksgiving. But on Monday we got to talk about design choices for the project after having learned more because of the practice meet we attended. Problems: We didn't have specific problems because of the break but we are excited to have some more time soon to start implementing. Plans: We will put our design ideas into writing for the design document next week and then finalize it for the final draft.

5.4.2 *Winter Term*

Week 1 Progress: This first week of winter term was not a good week for progress. I did get some research done on implementing an API and got node.js on my computer but I need to really get going next week. Problems: After going through my winter term schedule for one week, I've realized I don't have enough time to devote to this so I will need to make some changes. Plans: I am going to quit a couple responsibilities I have at night to free up more time to work on the project.

Week 2 Progress: I started working on the API without a database set up. Not sure if the code will help later but I am getting used to coding in javascript. Our team met and got organized. Problems: I've never made an API before so I am learning. It is hard to code unless we meet as a group because our system is so connected. Plans: We decided we need to be done with the database by the end of next week so we can move on with the other parts.

Week 3 Progress: We set up docker, the database, and the file system of the API. Problems: We were thinking of having arrays in our database in the design but we realized when implementing it that relational databases don't work like that so we set it up a different way. Plans: Zech and I will be doing the endpoints of the API this week.

Week 4 Progress: Zech and I wrote and tested some endpoints for the API. We still need to set up more but right now we have one endpoint for player and five for team. Problems: This is all new to us so we are working hard to learn as we go along. Plans: We already have the database initialized with the tables so we want to connect the database this week to the endpoints.

Week 5 Progress: Zech and I connected the database to the endpoints this week so the backend is working together nicely. Problems: It took a long time to get everything connected so now we are going to have to get a lot done next

week for the alpha deadline. Plans: We plan to connect the frontend to the backend next week. We are meeting as a group for much of the development time next week to speed the process up since half our team is frontend and half is backend.

Week 6 Progress: We got our frontend and backend working with each other for the scoring and line up pages so it is ready for the alpha. I reworked the database a little to include some information we realized we needed. Problems: We realized we needed each score from a judge to be an object so we could keep track of the judges for the score sheet so that took some time to rework the database. We also realized we needed a meet table in the database to keep track of different meets. Plans: Add endpoints for the newly added tables and get those working with the frontend. We are also meeting with our client this week to show progress and talk about security.

Week 7 Progress: We got all the web pages connected to the backend. I've been working reworking the database to make the endpoints better for having multiple meets. But it would work for a single meet right now. Problems: I am having trouble getting some of the foreign keys to work in the database. Plans: We will be having other members of the team look at the database to see if they can figure out the problem.

Week 8 Progress: We met with our client and Gill IT on Monday and had our client use the software. He made a vault lineup and then inputted all the scores for the vault lineup. He actually didn't have any trouble using it. Nick and Thomas did a good job of making the front end very user friendly so far. The only part we realized needed to be fixed was we need to add a confirmation after creating the lineup because that confused the client a little. We did have one after inputting the scores for the lineup though. Charles (from IT) was very happy with how the project was looking so far and really excited about how it would make it easier to score meets in the future. He was really happy that it didn't rely on network connection and that it was all containerized with docker. He wants to work with us on using the server in Gill now. Problems: As far as problems go, we are still having trouble with the changes in the database for the beta version. Zech and I worked on it for a long time on Sunday and we think the problem lies in the order that docker is making the tables but we haven't figured out how to change the order yet. Nick is going to help us out today because he knows more about docker so hopefully that will get resolved today because that could impede progress on the beta version. Plans: After the database problem gets resolved, we will be working on connecting the meet setup pages since we cannot do that until the meet and judge tables that we added work. We will also start working on the print functionality. Week 9 Progress: We finally got passed our database problem thanks to Nick. As we suspected, it was caused by us not knowing how docker created the tables. Docker was creating separate volumes for each table so they couldn't reference each other. So the fix was to put all the database init stuff in one volume. After that was resolved, Zech and I quickly went to work and finished and tested all the endpoints except for one so the backend is almost completely done. Problems: The problem this week was the database creation which got resolved. Hopefully, there are not anymore big problems left since we are almost done. Plans: Zech and I will finish the last endpoint and start on printing functionality. Nick will start on cookies. We will also meet on Sunday to work on connecting the last few pages to the backend. Week 10 Progress: We finished the endpoints and they all worked for Nick to use to finish the last of the pages. We have gotten all the features done except for exporting and printing the score sheets. I emailed Road to Nationals and asked for ideas on how to handle the score sheet and they responded. I also emailed Gill IT to see if they have any ideas but I haven't gotten a response yet. Thomas researched and set up cookies to use so the software has the current meet id. Problems: I struggled working on the score sheet printing this week and realized we wouldn't have time to finish that feature the way we were approaching it. So I reached out for help with finding another way to

do it. I really hope we find a doable way. Plans: We plan to meet for a long meeting on Monday to complete all the end of term progress reporting. I plan to keep following up with Road to Nationals and Gill IT to hopefully find an easier way to export and print the score sheet.

5.4.3 Spring Term

Week 1 Progress: I fixed a problem we had with judging on the backend so now we just need to update how the scores are put in on the front end. The average scores endpoint now checks if there are more than three scores and if so, drops the high and low before averaging the scores.

Problems: The score sheet feature is still a problem. We are not sure if we are able to finish exporting or printing the score sheet in time but we talked to Alex about it.

Plans: On Monday, we are working on sessions to resolve waiting for all the judge scores to be in. We will also work on authentication.

Week 2 Progress: We implemented sessions and authentication this week. Sessions allow for the users to get the current meet id in a cookie. Authentication only allow the score inputters to put in scores and admins are able to do everything. Both need a token. Problems: We were never able to get printing working. Plans: I made a list of features / requirements that should be added in case we pass this project on to another group for next year.

Week 3 Progress: We finished all we were going to do on our project before code freeze. The big thing I did this week was set up a testing suite on postman. It took longer than I expected to make requests for all the endpoints but it turned out well. All the endpoints work as expected after I fixed a couple of small things that showed up while testing. Problems: The client and I are in Texas right now for NCAA championships. So we aren't available for the document revisions and client verification which is due tomorrow. I sent an email about it but haven't gotten a response yet. Plans: We plan to fix up our poster for expo next.

Week 4 Progress: We finished revising the requirements and the design and got them approved and signed by our client. We also finished revising our poster this week. I continued to add to the list of things left to do for the software that could be used by a group in the future that takes on this project because there is still more to be done. Problems: Our client was away for NCAA championships and was unable to sign off on the documentation until today so we resubmitted it with his approval and signature today. Plans: We are planning to finish all the remaining assignments for this class and are excited for Expo.

Week 5 Progress: We got our poster signed off by the client and Professor Winters. We sent our poster to be printed in time. I am still adding tasks for the project to a list to be done by the next group. Problems: We had so much trouble understanding the poster instructions. We didn't get any feedback on ours besides some padding issues so we thought ours was good. We fixed the padding and then resubmitted it and then received feedback that the content on our poster was not good the day of the deadline to be sent to the printers. It did get resolved in time but it was a stressful situation. Plans: I plan to keep adding to the task list mentioned above and finish all the assignments for this class as they come. Next week, we will be preparing for expo.

Week 6 Progress: I went to class and heard all the details I need about expo. Problems: Thankfully, there weren't many problems to run into this week since nothing was due. Plans: I will be at expo between 9 and 10 am to sign in and set up our project.

6 FINAL POSTER

COLLEGE OF ENGINEERING
Electrical Engineering and Computer Science
CS18

PROJECT DESCRIPTION

- Our project consists of two main components: the API and the web page.
- The purpose of the API is to handle requests made by the webpage. The API also helps abstract away calls to the server, allowing for less powerful hardware to work as well as more powerful hardware.
- Our API is containerized, and handles requests based off of the express middleware architecture.
- We use Docker Toolbox to run our software.
- The purpose of the web page is to act as an easy method for scores to interact with the API. The webpage will essentially make requests to the API server and display that information in an easy and intuitive way for users.
- This project also uses a Microsoft SQL database to store data on teams and gymnasts.
- The software backend has 37 endpoints.
- The web application requires the user to be authenticated through login.

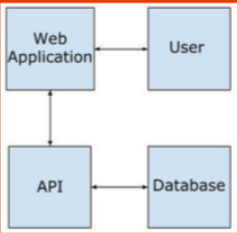




Figure 1: UML diagram of the components



OSU GYMNASTICS SCORING

Competitive Scoring Software for OSU Gymnastics CS Senior Capstone Project



Gymnastics photos by Ralph Greene

INTRODUCTION

The gymnastics meets held in Gill Coliseum are fast paced, usually televised, and attended by many fans. In order for competitions to run smoothly and provide a fun experience for fans, athletes, coaches, and staff, there must be a way for the scores judges give to each routine to be recorded, organized, and displayed using the hardware that is already in Gill.


The Oregon State Gymnastics Team would like new scoring software. The pre-existing software used to display and keep scores has become outdated and has issues displaying on the current hardware. Because of this, it is important for new, custom software to be built that can accurately and elegantly display scores on Gill Coliseum's hardware in a readable manner that is intuitive for fans to understand while providing all the necessary functionality.

BACKGROUND

- Oregon State Gymnastics is looking to improve the quality of experience for athletes, fans, and staff in scoring gymnastics meets.
- Gill Coliseum holds five to six gymnastics meets per year against other PAC-12 and NCAA opponents.
- The current scoring software is slow to set up, outdated, and has reliability issues.
- Post season competitions, such as regionals, add more complexity to scoring.

RESULTS

- The web application has user login authentication.
- The user starts using the application by setting up a meet.
- The user sets the number of teams, names of teams, number of players, names of judges, number of judges, and names of judges.
- The user can then set lineups for each event.
- After the lineups are set, judges can score each event.
- After scoring is complete, scoring and lineup data can be exported to csv files.



Team members (From L to R):
Zedariah DeCleene,
Thomas Huynh,
Mary Jacobsen,
Nick Giles

Contact information:
decleenz@oregonstate.edu
huynhthom@oregonstate.edu
jacomary@oregonstate.edu
gilesni@oregonstate.edu

Sponsor: Michael Chaplin,
Oregon State Gymnastics
Associate Head Coach

7 PROJECT DOCUMENTATION

7.1 10.0-Software-Group-18

The gymnastics meets held in Gill Coliseum are fast paced, usually televised competitions attended by thousands of fans. In order to run the competitions smoothly and provide a positive experience for fans, athletes, and staff, there must be a way to record, organize, and display the scores that the judges give to each routine using the hardware already in Gill Coliseum. The Oregon State Gymnastics Team would like new scoring software. The pre-existing software used to display and keep scores has become outdated and has issues displaying on the current hardware. Because of this, it is important for new, custom software to be built that can accurately and elegantly display scores on Gill Coliseum's hardware in a readable manner that is intuitive for fans to understand while providing all the necessary functionality.

7.2 How to run code

Since our project calls for a containerized API based web server, Docker is needed to run the code. You can install Docker [here](#), but it is important to note that if you are running any version of Windows that isn't enterprise edition you will instead need to download [Docker Toolbox](#). Once installed, open the 'Code' folder and use the command 'docker-compose up' this will run the server, to run as a background process use the command 'docker-compose up -d'. Once the

server is running open your favorite web browser and navigate to the [dashboard](#) or on [docker toolbox](#). The username is "admin" and the password is "hunter2". Note, if you are using docker toolbox, 'localhost:8000' will not work, and instead you will need to grab the ip address given by the virtual machine which can be found underneath Moby Dock (Docker's whale logo). Our current build does not have the print page implemented. We also have many endpoints that can be accessed. These can be found by looking at the created endpoints in the 'api' folder inside the 'Code' folder. We recommend an application called [postman](#) to query the API.

7.3 Using Postman to Test

To test endpoints with Postman you must enter the endpoint and the HTTP method and click on send. Data must be sent using x-www-form-urlencoded format. Ex. 'GET http://localhost:8000/team/20/meet' or 'GET http://192.168.99.100:8000/team/20/meet' will return a list of all teams that are participating in the meet where the meet ID = 20

We have made testing collections using postman that can be used to put information into the database and to test each endpoint. To import the collections, copy a link, click the import button on the top left inside postman, and then click "Import From Link" and paste. The endpoint collections work in the following order:

[user](#) [meet](#) [team](#) [player](#) [lineup](#) [judge](#) [score](#)

7.4 Endpoints

To view the endpoint documentation.

- 1) Start server using 'docker-compose up --build'
 - 2) Wait for server to start.
 - 3) Go to 'http://localhost:8000/endpoints.html'
- or if you are using docker toolbox 'http://192.168.99.100:8000/endpoints.html'

8 RECOMMENDED TECHNICAL RESOURCES FOR LEARNING MORE

8.1 Postman

[Postman Documentation](#)

9 CONCLUSIONS AND REFLECTIONS

9.1 Mary

9.1.1 What technical information did you learn?

I learned how to develop the back end of an API using javascript and express to make endpoints for a web application and a sql database on a server.

9.1.2 What non-technical information did you learn?

I learned more about how scoring gymnastics works during competitions. I also learned how to work with a client and gather information.

9.1.3 *What have you learned about project work?*

I learned about how to set realistic development goals and deadlines each week.

9.1.4 *What have you learned about project management?*

I learned how to organize meetings and when it is better to meet in person rather than meeting remotely. I also learned how to divide up work and that it is not always the most efficient to divide work up evenly because everyone works at different paces.

9.1.5 *What have you learned about working in teams?*

If everyone on the team is willing to do what they can to help the team, it works out so much better than if people are worried about contributing too much. Also, it builds up trust from your teammates if you put in extra work when possible.

9.1.6 *If you could do it all over, what would you do differently?*

I would have started coding in Fall term and done more of an agile approach. I also would have retrieved the Road to Nationals API documentation before the project started so we could have used that.

9.2 **Zech**

9.2.1 *What technical information did you learn?*

I learned how to build a server using the nodejs javascript framework. Additionally, I learned to build a RESTful API with endpoints that can effectively fetch and insert data into a MySQL database.

9.2.2 *What non-technical information did you learn?*

I learned a lot about how gymnastics and gymnastics scoring works.

9.2.3 *What have you learned about project work?*

I learned how to set realistic goals and how to work on a large scale project in a group setting.

9.2.4 *What have you learned about project management?*

I learned how to break up the project into smaller sections and how to divide those small projects among the team.

9.2.5 *What have you learned about working in teams?*

There needs to be a good balance of working remotely and working in-person.

9.2.6 *If you could do it all over, what would you do differently?*

Two things that I would do differently with this project is to start writing code sooner, and use more of an agile strategy. The other thing that I would have done differently is plan some of the larger structures more thoroughly (I'm looking at you original database design)

9.3 Nicholas

9.3.1 *What technical information did you learn?*

I learned a lot about how much work goes into creating a full stack web application. Throughout my studies I was able to work on all of the aspects of a web application, but this project helped teach me the methods of how to put everything together and all of the complications that can happen in the process of making a backend and frontend work together.

9.3.2 *What non-technical information did you learn?*

There was a lot of stuff I learned about when it comes to managing a project and working with teammates. I really enjoyed all of my teammates and felt very lucky that I got a group that gelled from the start, and it helped me learn how important it is to be a helpful and productive team member, for a single negative Nancy could run the entire group dynamic.

9.3.3 *What have you learned about project work?*

When it comes to project work, I learned how difficult it is to make all of the pieces come together and have different people work on different aspects of a project that need to work together. All of the syncing and standardizing that needs to be done to have a group work efficiently was surprising and took our team a while to get the hang of and learn.

9.3.4 *What have you learned about project management?*

Project management is something that has always come natural to me due to experience with project management through scouts at a young age; however, applying those techniques to independent adults is a very different beast than fellow scouts. Overall, I garnered a much better understanding of how to apply different teaching, leadership, and management techniques through working with my team on this project.

9.3.5 *What have you learned about working in teams?*

The major takeaway for teamwork that I learned was how much fun and how fulfilling working in a team can be. Throughout college I constantly got placed in sub par groups that did not work well together, and it taught me to dread working in groups or teams, but luckily this team really helped show me how a good group can work.

9.3.6 *If you could do it all over, what would you do differently?*

I would have started coding a little bit earlier, probably over winter break or even in fall term. Second, I would have looked more into client side javascript libraries like React and Angular JS.

9.4 Thomas

9.4.1 *What technical information did you learn?*

I learned to develop the frontend of a web application and adding features through JavaScript such as cookies and exporting to csv files. In addition, I learned how to use Docker Toolbox.

9.4.2 *What non-technical information did you learn?*

I learned how a gymnastics meet runs and how a gymnastics meet is scored.

9.4.3 What have you learned about project work?

I learned a project can seem overwhelming initially, but once you get your work rhythm going, it is not so overwhelming anymore.

9.4.4 What have you learned about project management?

I learned breaking up a large project to smaller pieces is crucial to making progress.

9.4.5 What have you learned about working in teams?

I learned communication and team chemistry is essential to a team's success.

9.4.6 If you could do it all over, what would you do differently?

I would be more proactive in starting work earlier instead of waiting for someone else to initiate. We did not start until the end of week 2 in winter term so it would have been nice if we started earlier.

APPENDIX

Server Code

```
1  const express = require('express');
2  const morgan = require('morgan');
3  const bodyParser = require('body-parser');
4  const exphbs = require('express-handlebars');
5  const mysql = require('mysql');
6  const api = require('./api');
7  const path = require('path')
8  const sessions = require("client-sessions");
9  const cookieParser = require('cookie-parser')
10
11  const app = express();
12  const port = process.env.PORT || 8000;
13
14  app.engine('handlebars', exphbs({defaultLayout: 'main'}));
15  app.set('view engine', 'handlebars');
16  app.use(morgan('dev'));
17  app.use(bodyParser.urlencoded( {extended: true }));
18  app.use(express.static('public'));
19  app.use(cookieParser());
20  app.use(sessions({
21      cookieName: 'meetSession', //key name added to request onject
22      secret: "process.env.SESSIONS",
23      duration: 12 * 60 * 60 * 1000,
24      activeDuration: 1000 * 60 * 60
25  }));
26
27  app.listen(port, () => {
28      console.log("== Server is running on port", port);
29  });
30
```



```
31 const mysqlHost = process.env.MYSQL_HOST;
32 const mysqlPort = process.env.MYSQL_PORT || '3306';
33 const mysqlDBName = process.env.MYSQL_DATABASE;
34 const mysqlUser = process.env.MYSQL_USER;
35 const mysqlPassword = process.env.MYSQL_PASSWORD;
36
37 const maxMySQLConnections = 10;
38 v app.locals.mysqlPool = mysql.createPool({
39 v   connectionLimit: maxMySQLConnections,
40 v   host: mysqlHost,
41 v   port: mysqlPort,
42 v   database: mysqlDBName,
43   user: mysqlUser,
44   password: mysqlPassword
45 });
46
47 // app.use(express.static(path.join(__dirname, 'public')));
48
49 app.use('/', api);
50
51 v app.use('*', function (req, res, next) {
52 v   res.status(404).json({
53     error: "Requested resource " + req.originalUrl + " does not exist"
54   });
55 });
```

API

```

1  const router = require('express').Router();
2  const validation = require('../lib/validation');
3  const { requireAuthentication, requireAdmin } = require('../lib/auth');
4
5  //schema for required and optional fields for a score object
6
7  const scoreSchema = {
8    id: { required: false }, //medium int
9    playerId: { required: true }, //medium int
10   judgeID: { required: true }, //medium int
11   score: { required: true }, //Decimal
12   event: { required: true }, //varchar
13   exhibition: { required: true }, //medium int (1=True/0=False)
14   meetID: { required: true } //medium int
15 };
16
17 /*
18 /-----
19 / Get all scores
20 /-----
21 / gets all scores
22 /
23 */
24 router.get('/', requireAdmin, function (req, res, next) {
25   // console.log("/team/teams");
26   const mysqlPool = req.app.locals.mysqlPool;
27   getScores(mysqlPool)
28   .then((scores) => {
29     if (scores) {

```

```
27     getScores(mysqlPool)
28     .then((scores) => {
29       if (scores) {
30         res.status(200).json(scores);
31       } else {
32         next();
33       }
34     })
35     .catch((err) => {
36       console.log(err);
37       res.status(500).json({
38         error: "Unable to fetch scores. Please try again later."
39       });
40     });
41   });
42
43   function getScores(mysqlPool) {
44     return new Promise((resolve, reject) => {
45       mysqlPool.query('SELECT * FROM score', function (err, results) {
46         // console.log(results);
47         if (err) {
48           reject(err);
49         } else {
50           resolve(results);
51         }
52       });
53     });
54   };
55
```

Frontend

```

1  var teams = [];
2  var judges = [];
3  var numTeams;
4  var itr = 0;
5
6  function getNumTeams() {
7      numTeams = parseInt($('#num-teams').val(), 10);
8      if (isNaN(numTeams)) {
9          alert("Please enter a valid non-negative whole number");
10         $('#number-teams-accept').one('click', getNumTeams);
11     } else {
12         teamNameAccept = $('#name-accept');
13         for (var i = 0; i < numTeams; i++) {
14             $('#name-accept').before("<input type='text' id='team-" + i + "' placeholder='Insert Name of Team'>");
15         }
16         $('#team-names-box').removeClass('hidden');
17     }
18 }
19
20 $('#number-teams-accept').one('click', getNumTeams);
21
22 function submitPlayers(name) {
23     playersVal = $('#.' + name);
24     for (var i = 0; i < playersVal.length; i++) {
25         $(playersVal[i]).val();
26     }
27 }
28
29 function allFilled(name) {
30     var bool = true;
31     playersVal = $('#.' + name);
32     for (var i = 0; i < playersVal.length; i++) {
33         console.log($(playersVal[i]).val());
34         if ($(playersVal[i]).val() == '')
35             bool = false;
36     }
37     if (bool) {
38         itr++;
39         getTeamData();
40     } else {
41         alert("Please make sure every gymnast field is filled in.");
42         $('#' + name + '-players-accept').one('click', () => {
43             allFilled(name);
44         });
45     }
46 }
47
48 function addPlayerBoxes(name) {
49     var numPlayers = parseInt($('#' + name + '-num-players').val(), 10);
50     var htmlStr = "<h3 class='form-text'>Gymnasts for " + name + "</h3>";
51     for (var i = 0; i < numPlayers; i++) {
52         htmlStr += "<input type='text' class='" + name + "' placeholder='Insert Name of Gymnast'>";
53     }
54     htmlStr += "<input type='submit' id='" + name + "-players-accept' value='Accept'>";
55
56     $('#' + name + '-num-accept').after(htmlStr);
57     $('#' + name + '-num-accept').addClass('hidden');
58     $('#' + name + '-players-accept').one('click', () => {

```

```
133 function postTeam(name, meet) {
134     var url = window.location.origin;
135     var teamObj = {
136         teamName: name,
137         meetID: meet
138     }
139     $.ajax({
140         url: url + '/team',
141         method: 'POST',
142         data: teamObj,
143         headers: {
144             "Authorization": 'Bearer ' + Cookies.getJSON('credentials').token
145         },
146         dataType: 'JSON',
147         success: (data, res) => {
148             console.log("postTeam-Succeeded in creating team: " + name);
149             console.log("postTeam-Data: " + data);
150             var players = $('.' + name);
151             for (var i = 0; i < players.length; i++) {
152                 console.log()
153                 postPlayer($(players[i]).val(), meet, data.id);
154             }
155         }
156     });
157 }
```

Authentication

```
3  const secretKey = 'hunter2';
4
5  function checkAuthToken(token, authLevel) {
6    return new Promise((resolve, reject) => {
7      jwt.verify(token, secretKey, (err, payload) => {
8        if (!err && payload.auth >= authLevel) {
9          resolve();
10         } else {
11           reject(403);
12         }
13       });
14     })
15   }
16
17  function generateAuthToken(userID, auth) {
18    return new Promise((resolve, reject) => {
19      const payload = {
20        sub: userID,
21        auth: auth
22      };
23      jwt.sign(payload, secretKey, {expiresIn: '12h'}, (err, token) => {
24        if(err) {
25          reject(err);
26        } else {
27          resolve(token);
28        }
29      });
30    });
31  }
```

```
33 function requireAuthentication(req, res, next) {
34   const authHeader = req.get('Authorization') || '';
35   const authHeaderParts = authHeader.split(' ');
36   const token = authHeaderParts[0] === 'Bearer' ? authHeaderParts[1] : null;
37   jwt.verify(token, secretKey, (err, payload) => {
38     if (!err) {
39       req.user = payload.sub;
40       next();
41     } else {
42       res.status(401).json({
43         error: "Invalid authentication token"
44       });
45     }
46   });
47 }
```

Database

```

1  DROP TABLE IF EXISTS `meet`;
2
3  CREATE TABLE `meet` (
4    `id` mediumint(9) NOT NULL AUTO_INCREMENT,
5    `name` varchar(255),
6    PRIMARY KEY (`id`)
7  ) ENGINE=InnoDB AUTO_INCREMENT=20 DEFAULT CHARSET=latin1;
8
9  LOCK TABLES `meet` WRITE;
10 INSERT INTO `meet` VALUES (null, 'test meet');
11 UNLOCK TABLES;
12
13 DROP TABLE IF EXISTS `team`;
14
15 CREATE TABLE `team` (
16   `id` mediumint(9) NOT NULL AUTO_INCREMENT,
17   `teamScore` DECIMAL(13,10),
18   `teamName` varchar(255) NOT NULL,
19   `vaultScore` DECIMAL(13,10),
20   `barsScore` DECIMAL(13,10),
21   `beamScore` DECIMAL(13,10),
22   `floorScore` DECIMAL(13,10),
23   `meetID` mediumint(9) NOT NULL,
24   PRIMARY KEY (`id`),
25   KEY `fk_meet_team` (`meetID`),
27   CONSTRAINT `meet_fk_1`
28   FOREIGN KEY (`meetID`)
29   REFERENCES `meet` (`id`)
30   ON UPDATE CASCADE
31   ON DELETE NO ACTION
32 ) ENGINE=InnoDB AUTO_INCREMENT=20 DEFAULT CHARSET=latin1;
33
34 LOCK TABLES `team` WRITE, `meet` READ;
35 INSERT INTO `team` (`teamName`, `meetID`)
36 VALUES ('Oregon State University',
37         (SELECT id FROM `meet` LIMIT 1));
38 UNLOCK TABLES;

```



```

1 <body>
2   {{> header}}
3
4   <main class="form-container">
5     {{> vbbfSelectBox}}
6
7     <div class="form-box hidden" id="team-select-box">
8       <h3 class="form-text" id="team-select-text"></h3>
9       <select class="team-picker" id="team-select">
10        <option value="" selected disabled hidden>Choose here</option>
11        {{#each teams}}
12          <option value="{{id}}">{{teamName}}</option>
13        {{/each}}
14      </select>
15    </div>
16
17    <div class="form-box hidden" id="player-name-box">
18      <center><h3 class="form-text" id="player-name"></h3></center>
19    </div>
20
21    <div class="scoring-box-container hidden">
22      {{#each judges}}
23        {{> scoringBox}}
24      {{/each}}
25    </div>
26
27    <div class="form-box hidden" id="accept-box">
28      <center><h3 class="form-text" id="accept-text">Please double check each Judge's scores are correct.</h3></center>
29      <input type="submit" id="score-accept" value="Submit Score">
30    </div>
31
32    <div class="form-box hidden">
33      <center><h3 class="form-text" id="scoring-complete-text"></h3></center>
34    </div>
35  </main>
36 </body>
37
38 <script src="https://cdnjs.cloudflare.com/ajax/libs/handlebars.js/4.0.10/handlebars.runtime.js"></script>
39 <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js"></script>
40 <script src="https://cdn.jsdelivr.net/npm/js-cookie@2/src/js.cookie.min.js"></script>
41 <script src="/scoring.js" charset="utf-8"></script>

```

Compose

```
1  version: '3.6'
2  services:
3    api:
4      build: .
5      image: gymnastics-api
6      restart: always
7      ports:
8        - 8000:8000
9      environment:
10        MYSQL_DATABASE: ${MYSQL_DATABASE}
11        MYSQL_USER: ${MYSQL_USER}
12        MYSQL_PASSWORD: ${MYSQL_PASSWORD}
13        MYSQL_HOST: mysql
14        MYSQL_PORT: 3306
15
16    mysql:
17      image: mysql:5
18      restart: always
19      volumes:
20        - mysql-data:/var/lib/mysql
21        - ./db-init:/docker-entrypoint-initdb.d
22      environment:
23        MYSQL_RANDOM_ROOT_PASSWORD: 'yes'
24        MYSQL_DATABASE: ${MYSQL_DATABASE}
25        MYSQL_USER: ${MYSQL_USER}
26        MYSQL_PASSWORD: ${MYSQL_PASSWORD}
27        MYSQL_ROOT_HOST: '%'
28
29
30  volumes:
31    mysql-data:
32      name: gymnastics-mysql-data
33
```

Style

```
1  /*
2  ** Color scheme hex codes:
3  ** Orange: #EE801E
4  ** Red Orange: #E75B10
5  ** Black: #000000
6  ** Dark Grey: #D6D1CE
7  ** Light Grey: #E3E0DD
8  ** White: #fff
9  */
10
11  body {
12      font-family: Roboto, Helvetica, sans-serif;
13      margin: 0;
14      background-color: #E3E0DD;
15  }
16
17  a {
18      text-decoration: none;
19  }
20
21  .hidden {
22      display: none;
23  }
24
```

```
25 .site-title {
26     margin: 0;
27     padding: 10px 20px;
28     font-size: 30px;
29     font-weight: 100;
30     background-color: #EE801E;
31     color: #fff;
32 }
33
34 .site-title a {
35     color: #fff;
36 }
37
38 .site-brand {
39     float: right;
40 }
41
42 .navbar {
43     background-color: #D6D1CE;
44     font-size: 18px;
45     line-height: 22px;
46 }
47
48 .navlist {
49     margin: 0px;
50     padding: 0;
51 }
```