# Gymnastics Scoring Software

The purpose of this paper is to define the design for the proposed gymnastics scoring software.

**10.0 Software**

Design Document

CS 461

Senior Software Engineering Project

Fall 2018

**Abstract**

This document archives the in depth technical and design choices made by 10.0 Software for our competitive gymnastics scoring software. Continuing, this document shows systems through the use of UMLs to visually show the classes, objects, systems, and how all of the systems are interconnected. Moreover, this document touches on the topics of conceptual models, design descriptions, and design viewpoints to further show and document how the system works.

# Contents

# 1 Overview

## 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.

## 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.

## 1.3 Intended audience

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

## 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.

# 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 |
| Moderator (authentication level) | The highest level of authentication. Moderators have the ability to set-up teams, meets, and delete data. |
| Judge (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 Conceptual model for software design descriptions

## 3.1 Software design in context

For the gymnastics scoring software project, an object oriented design method will be used. Self-contained entities such as gymnast, team, and event will be contained in objects. Design relationships such as encapsulation, specialization, and composition will be used to implement the scoring application. We will construct a design viewpoint to show design elements. A design elements consists of four categories: design constraints, design relationships, design attributes, and design entities. The design viewpoints will specify the design elements and conventions for the design views that it governs. Each design view contains multiple entities, attributes for the entities, relationships between entities, and constraints on those design elements. Stakeholders of the gymnastics scoring software project can obtain the information they need through these design views.

### 3.2 Software design descriptions within the life cycle

#### 3.2.1 Influences on SDD preparation

The most important factor to the SDD is the requirements document.The design decisions and design constraints are based on the contents of the requirements document.

#### 3.2.2 Influences on software life cycle products

The SDD can influence change in the requirements document and test documentation. During the process of creating the SDD, we may discover through design decisions or design constraints that the requirement document needs changes. When we create test cases and test procedures, we will need to consider the SDD.

#### 3.2.3 Design verification and design role in validation

The SSD will tested through design verification to find out if the design properly meets the stakeholders' design requirements and covers demanded design choices. Design concerns may be discovered through verification. The verification process can done through analysis, inspection, or review. The design role in validation is presenting an overview of how the system will work, the reasoning behind the design choices, and how the design traces to the requirements document.

## 4 Design description information content

### 4.1 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

### 4.2 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 1st draft.

- Scope
The scope of the 10.0 Software design description is the design of the web portal, API, database, and score board server, 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 and 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.

## 4.3 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.

## 4.4 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.

## 4.5 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.

### 4.6 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 upload a .JSON file for each team. The .JSON contains information on the team for team setup.
- **API**: The API is the logical process responsible for reacting to user requests.
- **Database**: The SQL database hosted on Gill Colosseum's server will store all team and gymnast related data.

### 4.7 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.

### 4.8 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 Microsoft SQL because the information technology staff informed us it is required for their scoring system. The fast set up time design concern was addressed by allowing users to input a JSON file with the team rosters.

### 4.9 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.

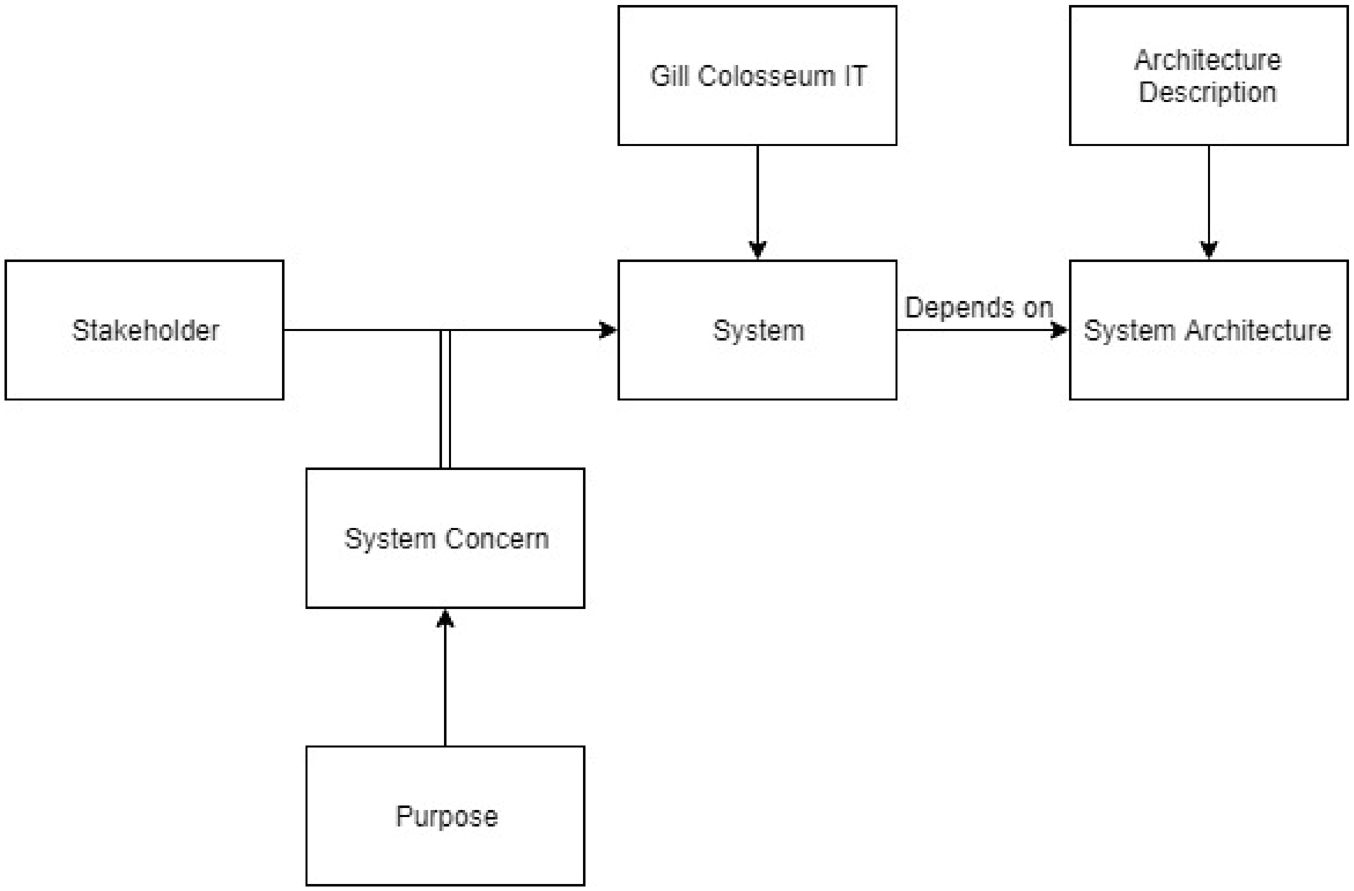
# 5 Design viewpoints

## 5.1 Introduction

**Table 1—Summary of design viewpoints**

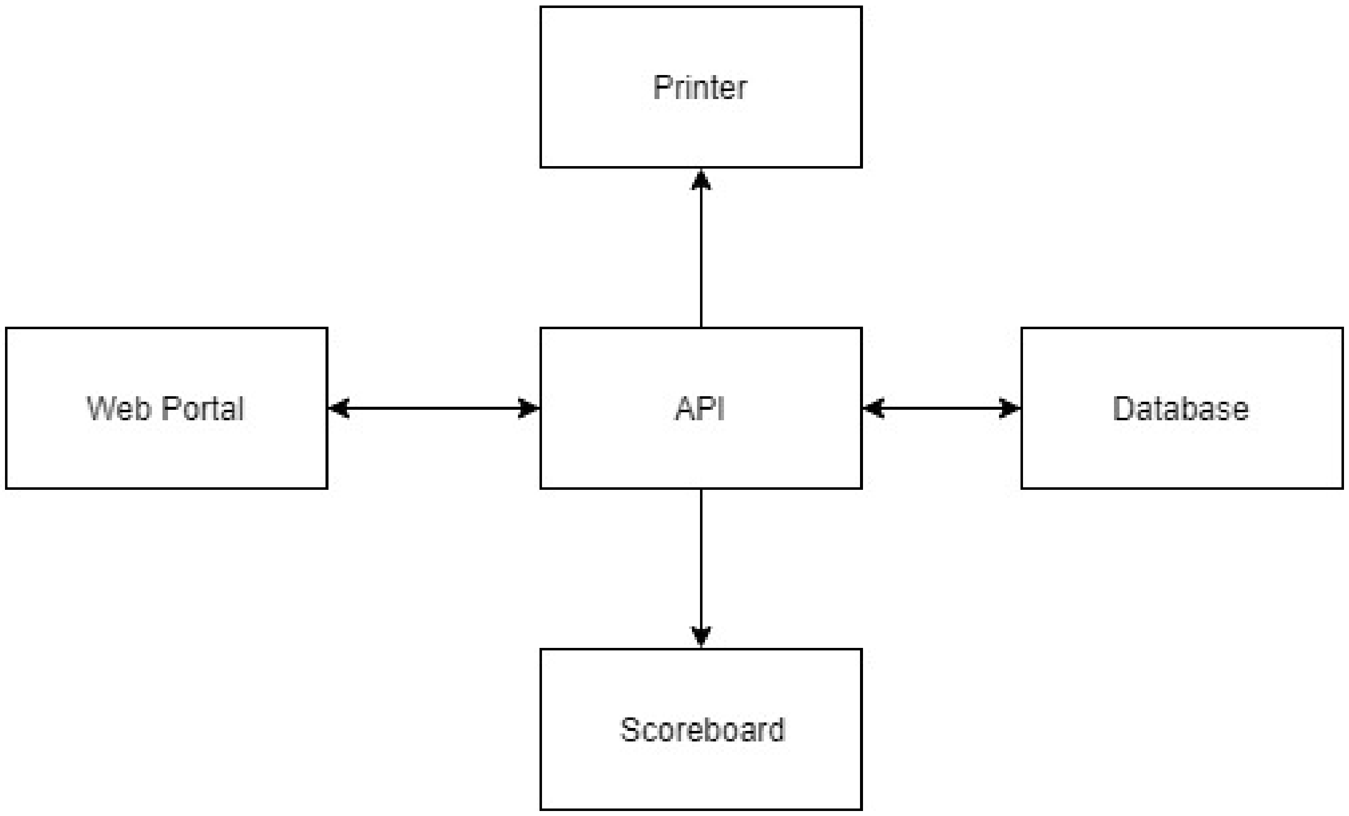
| **Design Viewpoint** | **Design concerns** | **Example design languages** |
|---|---|---|
| Context (5.2) | Gill coliseum IT people, judge assistants/OSU gymnastic data inputters | UML use case diagrams |
| Composition (5.3) | SQL database, API server, client side web portal, scoreboard server | UML 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 |

## 5.2 Context viewpoint



## 5.3 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.



### 5.4 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.

### 5.4.1 /teams

| **/teams** |
|---|
| - gymnast[]: gymnasts<br>- int: teamScore<br>- varchar: teamName<br>- int: vaultScore<br>- int: barScore<br>- int: beamScore<br>- int: floorScore<br>- gymnast[]: vaultLineup<br>- gymnast[]: barLineup<br>- gymnast[]: beamLineup<br>- gymnast[]: floorLineup<br>- gymnast[]: vaultRotation<br>- gymnast[]: barRotation<br>- gymnast[]: beamRotation<br>- gymnast[]: floorRotation |
| + GET /teams<br>+ GET /teams/:teamname<br>+ GET /teams/:teamname/teamscore<br>+ GET /teams/:teamname/:gymnastname<br>+ GET /teams/:teamname/:event/lineup<br>+ GET /teams/:teamname/:event/rotation<br>+ POST /teams (team object)<br>+ PUT /teams/:teamname (team object)<br>+ PUT /teams/:teamname/:event/lineup (gymnast[])<br>+ PUT /teams/:teamname/:event/rotation (gymnast[])<br>+ DELETE /teams/:teamname |

| **Name** | **Return** | **Authentication level** | **Definition** |
|---|---|---|---|
| GET /teams | team[] | Moderator or Judge | Returns a list of all of the teams in the meet |
| GET /teams/:teamname | team | Moderator or Judge | Returns a team object specified by the :teamname |
| GET /teams/:teamname/teamscore | int | Moderator or Judge | Returns the teamscore for the team specified by :teamname |
| GET /teams/:teamname/:gymnastname | gymnast | Moderator or Judge | Returns the gymnast object specified by :gymnastname in the team object specified by the :teamname |
| GET /teams/:teamscore/:event/lineup | gymnast[] | Moderator or Judge | Returns the lineup of gymnasts for the event specified by :event from the team specified by :teamname |
| GET /teams/:teamname/:event/rotation | gymnast[] | Moderator or Judge | Returns the rotation of gymnasts for the event specified by :event from the team specified by :teamname |
| POST /teams | Status code 201 | Moderator | Creates a team object in the database based on the object sent in the request header of the packet. Request must fill all data needed for the team |
| PUT /teams/:teamname | Status code 204 | moderator | Updates a team object |
| PUT /teams/:teamname/:event/lineup | Status code 204 | Moderator | Updates the lineup of gymnasts that are participating for the team specified by :teamname in the event specified by :event |
| /teams/:teamname/:event/rotation | Status code 204 | Moderator | Updates the rotation of gymnasts that participating for the team specified by :teamname in the event specified by :event |
| DELETE /teams/:teamname | Status code 204 | Moderator | Deletes the team specified by :teamname in the database |

#### 5.4.2 /event

| **/event** |
|---|
| - result: [gymnasts] |
| + GET /result/:event<br>+ PUT /result/:event |

| **Name** | **Return** | **Authentication level** | **Definition** |
|---|---|---|---|
| GET /result/:event | gymnast[] | Moderator or Judge | Returns the result of the event specified in :event |
| PUT /result/:event | gymnast[] | Moderator or Judge | Updates the result by getting the full list of all gymnasts that participated in the event then organizes in order of highest score |

### 5.4.3 /gymnasts

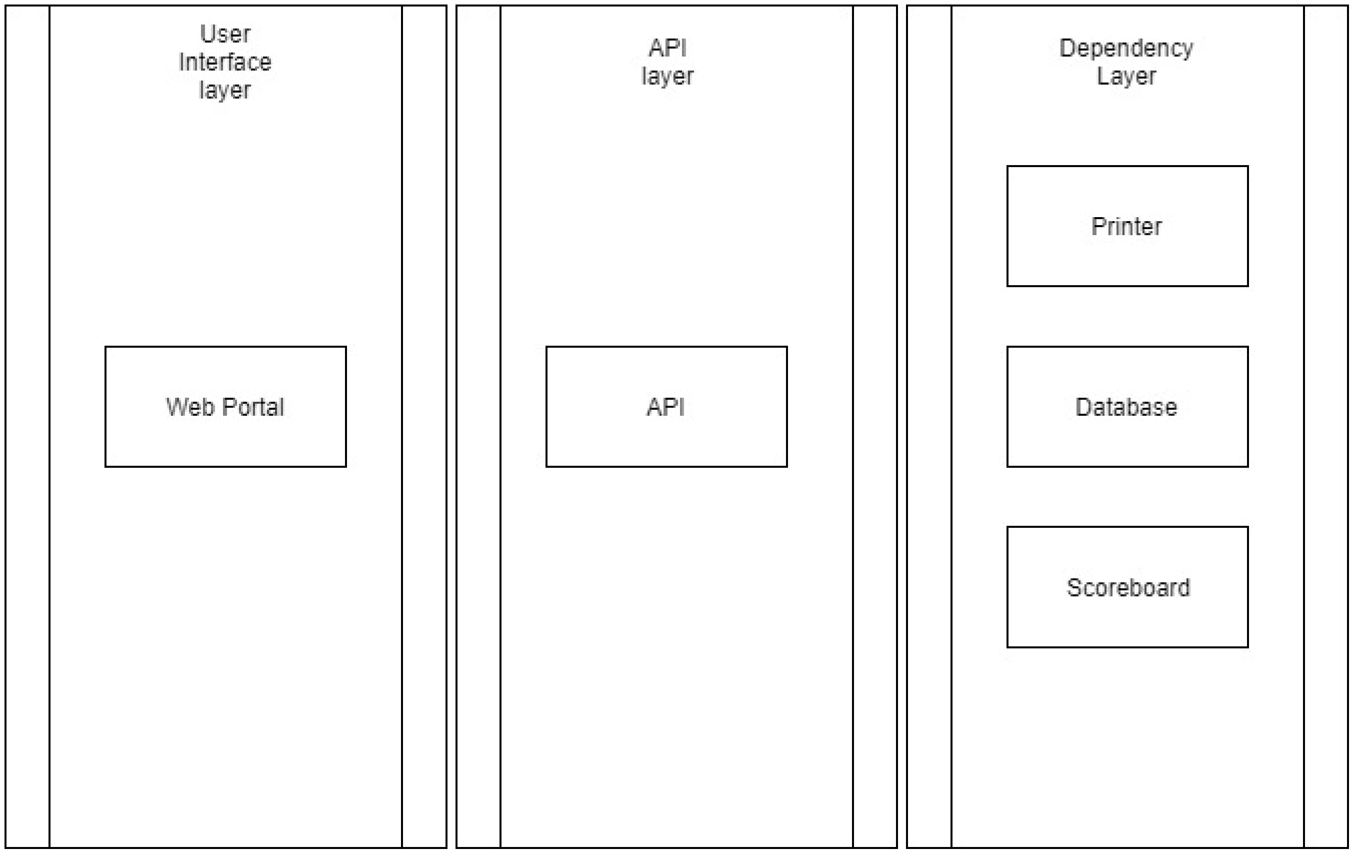
| **/gymnasts** |
|---|
| - gymnastName: string<br>- vaultScore: int<br>- barScore: int<br>- beamScore: int<br>- floorScore: int<br>- allAroundScore: int<br>- vaultExhib: bool<br>- barExhib: bool<br>- beamExhib: bool<br>- floorExhib: bool<br>- team: string |
| + GET /gymnasts/:gymanstname<br>+ GET /gymnasts/:gymnastname/:event<br>+ GET /gymnasts/:gymnastname/aascore<br>+ POST /gymnasts/:gymnastname/:event (int[] scores)<br>+ PUT /gymnasts/:gymnastname/:event (int[] scores)<br>+ DELETE /gymnasts/:gymnastname/:event |

| | | | |
|---|---|---|---|
| GET /gymnasts/:gymanstname | gymanst | Moderator or Judge | Returns the data for the gymnast specified by :gymnastname |
| GET /gymnasts/:gymnastname/:event | int | Moderator or Judge | Returns the score for the gymnast specified by :gymnastname in the event specified :event |
| GET /gymnasts/:gymnastname/aascore | int | Moderator or Judge | Returns the all around score for the gymnast specified by :gymnastname in the event specified :event |
| POST /gymnasts/:gymnastname/:event | Status code 201 | Moderator or Judge | Takes the array of scores given, runs the scoring algorithm then creates a score for the gymnast specified by :gymnastname for the event specified by :event |
| PUT /gymnasts/:gymnastname/:event | Status code 204 | Moderator or Judge | Updates the for the gymnast specified by :gymnastname for the event specified by :event by running the scoring algorithm with the scores given |
| DELETE /gymnasts/:gymnastname/:event | Status code 204 | Moderator | Deletes the score for the gymnast specified by :gymnastname for the event specified by :event |

## 5.5 Dependency viewpoint

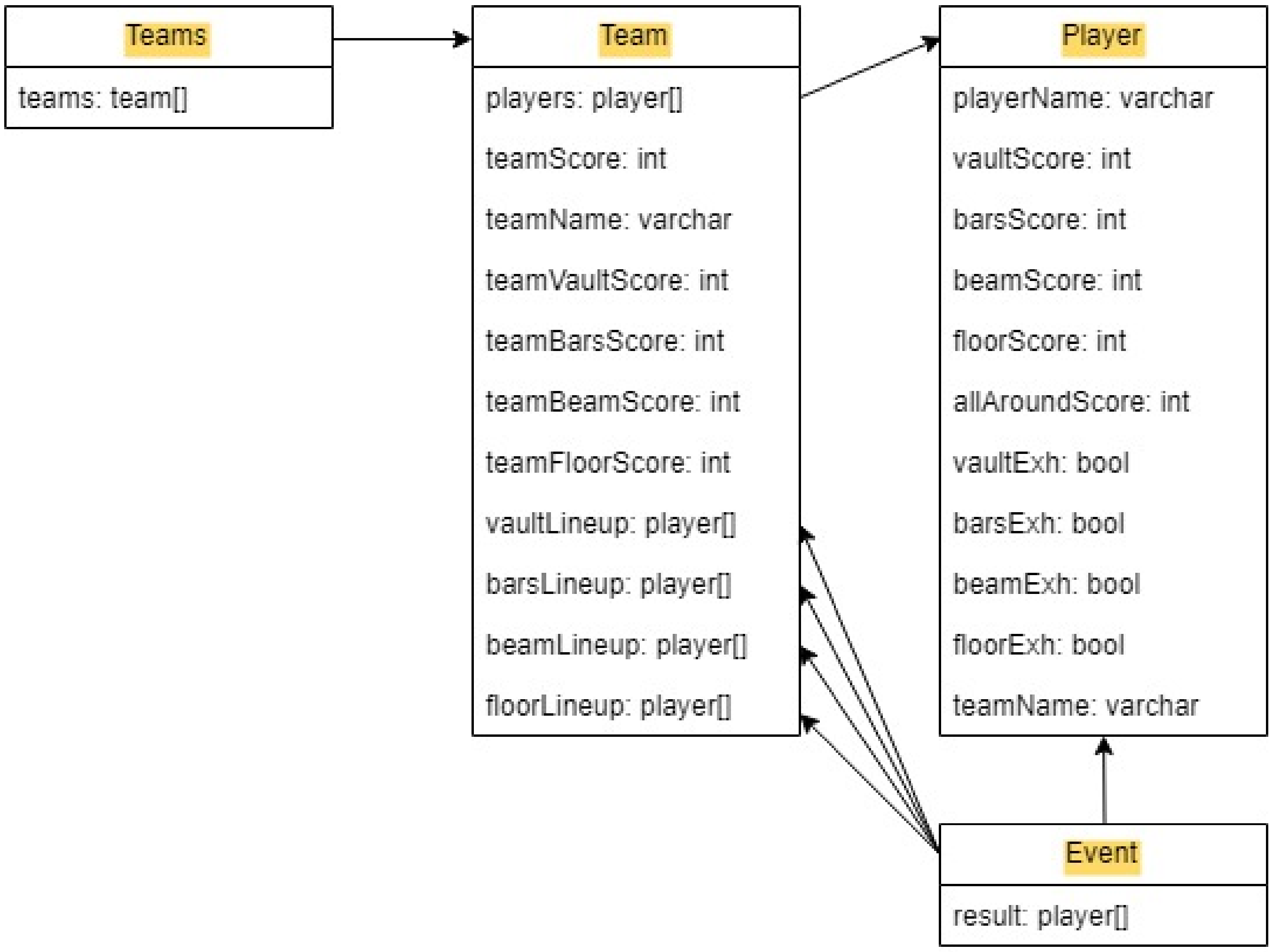
This system implements a three layer architecture in its design. This three layered system allows for modularization of each layer, allowing upgrades and changes to be more easily implemented.
The three layered architecture contains the user interface layer, the API layer, and the dependency layer.

- 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.



## 5.6 Information viewpoint

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



## 5.7 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.

## 5.8 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 chose 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.

## 5.9 Structure viewpoint

The structure of this project can be exemplified by a client-server architecture. In essence, a client-server architecture is 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.

## 5.10 Interaction viewpoint

Whenever the user want's 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. The API also has functions to print scorecards and send XML data to the scoreboard which the scoreboard parses into usable data.

## 5.11 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 the 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.

## 5.12 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.

## 5.13 Resource viewpoint

This section will contain every npm module we used and the purpose for why it was used. Furthermore it will contain information on the scoreboard server, sql database, and printing service once more is known.