COS 497: Capstone II

Code Inspection Report

UMaine Athletic Department Inventory Management System

Version 1.0

Collin Rodrigue, Brennan Poitras, Graham Bridges, Gabe Poulin, Sean Radel

Jude Killy - University of Maine Athletic Department

4 March 2024

UMaine Athletic Inventory Management System

Code Inspection Report

Table of Contents

Page

1. Introduction

This is a capstone project for the University of Maine Athletics Department. Our group of five computer science students is developing a scalable cloud-based solution to assist in the management and tracking of the University of Maine Athletic Department equipment and item inventory. The key stakeholders for our project are Jude Killy, Nick Fox, and Kevin Ritz. Jude Killy submitted the project on behalf of the athletic department to combat the issue of an inefficient inventory management process. Currently, Nick Fox and Kevin Ritz are tasked with managing inventory orders and distribution of items to players and teams. Currently, this process is only documented on a spreadsheet. Our group is developing a solution that can increase the efficiency of entering data items, storing inventory data, and tracking the distribution to athletes and teams. Our solution replaces the old inventory management solution Front Rush and the currently used strategy that's a combination of Excel spreadsheets and word of mouth to track inventory.

1.1 Purpose of This Document

The purpose of this document is to detail the key artifacts and processes of our code inspection meetings. This document describes our coding and commenting conventions, defect checklist, code inspection process and impressions, modules inspected, and list of defects. Our findings serve as a tool to categorize our work within a table which we can then use to conventionally reconfigure our codebase.

References

1. IMSG. "System Requirements Specification" November 1 2023, https://docs.google.com/document/d/1LnOj2DEyu8DPbKXBTDBm2y6UbePr_AXC/edit
2. IMSG "System Design Document" November 15, 2023, https://docs.google.com/document/d/1_dkbb3zXQzOxVbdIr3nbizQc0r-f6yC2/edit
3. IMSG. "IMSG_Capstone" March 22 2024, https://github.com/gsb02/IMSG_Capstone

1.2 Coding and Commenting Conventions

Examples of coding conventions can be found in Appendix A.

Commenting Convention

We do not follow a standard commenting convention, but we are adopting our own. All developers must add comments at the module level to describe the group of related procedures. Developers must also add comments at the procedure level to describe the purpose of the procedure itself. Procedure-level comments should describe parameters, any "clever" logic where it isn't obvious what the code is doing, and finally, the outputs of the procedure or function. Modules should have a block of comments after the list of imports that lists at least the following information: Author, Latest Update Date, and Purpose of Module.

Coding Conventions

Front-end Coding Convention

The front-end follows a coding convention that fits each of our own personal styles, as well as being readable to an outside audience. The front-end imports only the modules that would be needed for the present code, maintains consistent indentation per line of code, try/catch functions to catch errors, navigate functions to navigate the page, camelCase for variable names, styling both inside and outside of the page, and similar names for functions across pages. An example of the latter would be: handleClick, this is a function that most pages have to handle on click for specific things.

Back-end Coding Convention

The back-end follows a coding convention of our own style that includes the following: camelCase variable naming, importing only the necessary modules, consistent indentation, async/await syntax for handling promises, try/catch blocks for error handling, object destructuring to extract values from req.body and req.params, HTTP status codes for server success and failures, responses in JSON format, and string interpolation to construct SQL queries.

Database Coding Convention

The database schema follows a coding convention of our own style that includes the following: consistent indentation, uppercase keywords, standard SQL data types, constraints defined for tables, table locking, and consistent table dropping.

1.3 Defect Checklist

Give a comprehensive tabular checklist of possible defects that you used during the inspection process. Create categories for the defects such as Coding Conventions, Logic Errors, Security Oversights, and Commenting (you may create your own categories). Remember to take into consideration defects that are programming language-specific. (I expect to see at least **15** possible defects.)

List of Defects

Error
Syntax Error
Edge Case Error
Improper Error Handling
Security Oversights
SQL Insecurity
Naming Convention

Unused import
Unused variable
Data Inaccuracies
Failure to meet Use Case
Requirement
Usability Issues
Performance Issues
Commenting convention
Data Sanitization
Code Style

Table 1.1

2. Code Inspection Process

Author: The author wrote the software being inspected and is present to answer questions. The author does not defend their work. The author is responsible for reading their code line-by-line, or paraphrasing where necessary.

Recorder: The recorder describes any found defect in writing so that it can be included in the inspection report. The recorder will list in the code inspection table what file and what line the defect is found in.

Inspector: The inspectors raise questions and suggest problems within the code. Anyone but the author can be an inspector.

- **Preparations**: Each author creates a branch on the GitHub repository for their code inspection.
- **Role Selection**: There will be one recorder, three inspectors, and one author
    - The inspectors should include at least one back-end developer and at least one front-end developer at the same time
- **Reading**: Author reads all code modules that they wrote.
    - Inspectors call our what defect, what file, and what line, for the recorder to note down in the table
    - The recorder will note it down

- **Switch Roles**: After the author finishes reading through their modules, they will switch to a new role, and another author will conduct a review of their modules.
- **Goals**: Knowledge transfer of modules.
    - Find bugs/errors
    - Report on bugs
    - Communicate a generalized coding standard

2.1 Description

The process we used for inspecting each of our own code was similar for each person. We came up with a complete process before starting any real code inspection, to ensure that everyone's code was evaluated equally and effectively.

First, we decided who's code we were going to review. We would have them both pull up a screen stream on Discord and create a branch in GitHub named according to who's code review it was. Afterward, we decided who would be the recorder, as everyone else participating would be an inspector. The recorder can also inspect, ask questions, and point out issues, but their main role is to record everything. These roles are decided before any code is inspected and are maintained throughout the author's code inspection. The roles can change between authors.

Following role selection and preparation for the code inspection, the code can start being inspected. The author would pull up their code and explain how things work line by line. Comments would do what the authors explaining would do, but since we had a lack of comments across all code, verbal explanations were needed.

Whenever someone noticed a defect, it was noted what line, module and what the defect was. We also made sure to give a brief explanation of why the defect was there, for both documentation and future fixing of the defect. These notes are in a table (section 4), for each person, listed with the author, recorder and the inspectors. This process went on repeat until all code inspections were completed.

2.2 Impressions of the Process

In general, the code inspection meetings we held were really effective for our group. As a team, we were able to evaluate our current codebase by looking for any inconsistencies, defects, and unused code. We held multiple meetings over the course of our current development period, each member acting as a different role to make sure that everyone within our group could contribute effectively and ensure clean and consistent software. We were able to use these meetings to identify and standardize the flaws within our code. As a group, we agreed that each meeting served as a milestone for what we had accomplished and what needed to be improved. The time we spent critiquing our work gave insight into the skills we have learned and what we could do better.

These meetings are necessary for every software development team. They give a fundamental structure to the code review process, which most tech companies have adopted. Alternating roles for each meeting gave each group member a chance to study and learn the parts of our system that they did not formally contribute to. For our group, these meetings allowed us to compare our initial ideas with our current solution. We plan to fix the found problems to deliver a complete product.

We feel that our software's "best" modular units involve our database. The database that our system functions with has consistency and abstractness. However, our front-end react components are also developed and working. We have successfully integrated both units to meet many of our preliminary requirements. Our worst modular units have to do with the styling of our front-end pages. We have multiple files that use inline styling, and others that do not. We think this is because there was a differentiation in ideas about conventional styling techniques. Overall, our application is operational and usable.

2.3 Inspection Meetings

This is a list of meetings that we held for our inspection review process.

Meeting 1

Date: 3/5/24
Time Started: 5:26PM
Time Ended: 6:30PM
Location: Discord
Who: ALL
Roles:
- Author: Gabe Poulin
- Recorder: Collin Rodrigue
- Inspector: Sean Radel, Brennan Poitras, Graham Bridges

Modules:
- equipment.jsx
- addEquipment.jsx

Meeting 2

Date: 3/5/24
Time Started: 6:34PM
Time Ended 7:56PM
Location: Discord
Who: ALL
Roles:
- Author: Sean Radel
- Recorder: Gabe Poulin
- Inspector: Collin Rodrigue, Brennan Poitras, Graham Bridges

Modules:

- teamRoutes.js
- sportsRoutes.js
- OrderRoutes.js
- Team.js
- Sports.js
- Orders.js
- teamsControllers.js
- sportsControllers.js
- ordersControllers.js
- Docker-compose.yml
- docker-api-test.yml

Meeting 3

Date: 3/7/24
Time Started: 2:05PM
Time Ended: 2:51PM
Location: Discord
Who: Graham Bridges, Collin Rodrigue, Brennan Poitras, Gabe Poulin
Roles:

- Author: Graham Bridges
- Recorder: Collin Rodrigue
- Inspector: Brennan Poitras, Gabriel Poulin

Modules:

- Players.jsx
- Players.css
- addPlayer.jsx
- addPlayer.css
- Modal.jsx
- Modal.css

Meeting 4

Date: 3/19/24
Time Started: 12:45PM
Time Ended : 1:10PM
Location: Discord
Who: Collin Rodrigue, Brennan Poitras, Graham Bridges
Roles:

- Author: Collin Rodrigue
- Recorder: Graham Bridges
- Inspector: Brennan Poitras

Modules:
Home.jsx
Teams.jsx
addTeams.jsx

Home.css
teams.css

Meeting 5

Date: 3/19/24
Time Started: 1:15PM
Time Ended: 1:48PM
Location: Discord
Who: Sean, Collin, Brennan, Graham
Roles:
- Author: Brennan Poitras
- Recorder: Sean Radel
- Inspector: Collin Rodrigue, Graham Bridges

Modules
- index.js
- dbConfig.js
- equipmentRoutes.js
- playerRoutes.js
- loggingRoutes.js
- equipmentControllers.js
- playerControllers.js
- loggingControllers.js
- Player.js
- Equipment.js
- Log.js
- Apparel.js
- Shoe.js
- Jersey.js

3. Modules Inspected

The following section contains the names and descriptions of all source code modules that were inspected.

Front-end Modules:

All front-end code modules correspond to the views in our MVC architecture. These pages send information to the back-end controllers for processing and are then updated based on data received from the controllers.

home.jsx: This is the initial page that the user is greeted with after first logging into our system.

team.jsx: This module is used to view, add, and edit current sports teams.

addTeams.jsx: This module acts as a submission mechanism for users to add new sports teams to our system.

players.jsx: This module is the page where the user can view, delete, and add players. Users can be navigated to this page from the teams page.

addPlayers.jsx: This module is the front-end page where users can add players to a team in the database. All fields must be filled out before the player can be added to a team.

Modal.jsx: This module is a pop up created for confirming deletion of a player. The user can either confirm or cancel the deletion in the pop up.

equipment.jsx: This module is where the user can view the equipment of a given team. Users can also be redirected to addEquip and addEquipToTeam module pages from this page.

addEquip.jsx: Users can add equipment to a team from this page. As of now, the only type of equipment that can be added is apparel.

addEquipToTeam: Users can assign equipment to a specific team on this page.

Back-end Modules:

index.js: This is the starting point for our API. It specifies the middleware we are using and starts the server.

Routes:

All routes are part of the back-end API. They connect the front-end to the controllers in our back-end. These modules contain no logic or use case functionality, but specify the request URL and parameters needed to send and retrieve sets of data.

equipmentRoutes.js: This file specifies the API endpoints used to send and retrieve data related to equipment.

loggingRoutes.js: This file specifies the API endpoints used to retrieve data related to history and logging.

playerRoutes.js: This file specifies the API endpoints used to send and retrieve data related to players and their assigned equipment.

orderRoutes.js: This file specifies the API endpoints used to send and retrieve data related to orders

teamRoutes.js: This file specifies the API endpoints used to retrieve data related to teams and their assigned equipment.

sportsRoutes.js: This file specifies the API endpoints used to retrieve data related to sports.

Controllers:

All controllers are part of the Model, View, Controller architecture specified in our SDD. These controllers serve as the midpoint between our front-end and database and are used to send information between the views and the models.

equipmentControllers.js: This file defines all use case functionality related to equipment including creating, updating, and deleting equipment.

loggingControllers.js: This file defines all use case functionality related to equipment including creating log items based on the action history in the application.

playerControllers.js: This file defines all use case functionality related to players including creating, updating, deleting, and assigning equipment.

orderControllers.js: This file defines all use case functionality related to ordering including creating, updating, and deleting orders.

teamsControllers.js: This file defines all use case functionality related to teams including creating, updating, deleting, and assigning equipment.

sportsControllers.js: This file defines all use case functionality related to teams creating and reading sport information.

Models:

These classes represent the models in the MVC architecture. All models in our system are a one-to-one mapping of the tables in our database. These models serve as the point of contact between our back-end and database and are used to actually perform the MySQL operations requested by the controller. These models return any data to the controllers to be further processed if necessary.

Equipment.js: This is a class representing equipment and defines the basic database operations that can be done on equipment.

Log.js: This is a class representing the log and defines the basic database operations that can be done with the logging history.

Apparel.js: This is a class representing apparel and defines the basic database operations that can be done on apparel.

Shoe.js: This is a class representing shoes and defines the basic database operations that can be done on shoes.

Jersey.js: This is a class representing jerseys and defines the basic database operations that can be done on jerseys.

Player.js: This is a class representing players and defines the basic database operations that can be done on players.

Team.js: This is a class representing teams and defines the basic database operations that can be done on teams.

Order.js: This is a class representing orders and defines the basic database operations that can be done on orders.

Sport.js: This is a class representing sports and defines the basic database operations that can be done on sports.

Config:

dbConfig.js: This file is used to connect to our MySQL database.

keys.js: This file is used to pull in our MySQL environment information that is used in the database configuration.

4. Defects

This section contains all defects that were found as part of the Code Inspection Process. Table 4.1 lists all of the possible defect types. The following tables contain all the defects found for each member's authored code modules. Each table has a header describing the author of the code modules, the recorder of the defects, and the inspectors of the code modules. The defects table contains the module where the defect was found, the type of defect from Table 4.1, and a description of the deficiency.

Error Number	**Errors**
1	Syntax Error
2	Edge Case Error
3	Improper Error Handling
4	Security Oversights
5	SQL Logic Error
6	Naming Convention
7	Unused import

8	Unused variable
9	Data Inaccuracies
10	Failure to meet Use Case
11	Requirement
12	Usability Issues
13	Performance Issues
14	Commenting convention
15	Data Sanitization
16	Code Style
17	Logic Error
18	Package error

Table 4.1

Inspection 1
Roles: Author: Gabe Poulin
Recorder: Collin Rodrigue
Inspector: Sean Radel, Brennan Poitras, Graham Bridges

File	Line Number	Error #	Description
addequip.jsx	1,2,5	16	Inconsistent semicolon on imports
addequip.jsx	15,16,17,20,21	9	UseState starting value inconsistent between numbers and strings
addequip.jsx	134,135	16	Inconsistent use of “” vs ‘’
addequip.jsx	163-190	14	163 - 190 need commenting
equipment.jsx	1-45	16	Inconsistent semicolon on imports/useState
all files	all lines	14	lack of commenting in general

Table 4.2

Inspection 2
Roles: Author: Sean Radel
Recorder: Gabe Poulin
Inspector: Collin Rodrigue, Brennan Poitras, Graham Bridges

File	Line Number	Error #	Description
docker-compose.yml	8-9	4	Hard-coded credential - database credentials
docker-compose.yml	20-23	4	Hard-coded credential - database (api) credentials
docker-compose.yml	21	4	Running as root user by default
teamRoutes.js	3	16	Router variable should be with rest of code, not imports
routes folder	all files	14	give a deeper explanation of what is going on
sportsRoutes.js	9	16	missing semicolon at end of line
Team.js	90-125	10	equipment quantity not included during assignment
Player.js	97-125	10	equipment quantity not included during assignment
Player.js	61-78	1	improper use of function declaration (async and static)
Orders.js	66-76	17	failure to use correct level of variable for specific function use
controllers folder	all lines	15	lack of data sanitation for security
controllers folder	all lines	9	lack of security over data variable types
teamsControllers.js	31,44,57, 78	14	unused comments/bloat
teamsControllers.js	43-54	10	failure to delete players when deleting a team
teamsControllers.js	56-65	17	failure to read request parameters correctly
teamsControllers.js	77-98	17	failure to read all necessary parameters
sportsControllers.js	33-34	17	failure to read name as part of sport creation
sportsControllers.js	60-61	17	failure to read request parameters correctly
orderControllers.js	56-71	17	failure to read request parameters correctly
lint.yml	whole doc	18	package error

docker-api-test.yml	27	18	package error
all files	all lines	14	lack of commenting in general

Table 4.3

Inspection 3
Roles: Author: Graham Bridges
Recorder: Collin Rodrigue
Inspector: Gabe Poulin, Brennan Poitras

File	**Line Number**	**Error #**	**Description**
players.jsx	9	7	Navigate not used
players.jsx	16	6	Inconsistent use: class used in app/grade used in database
players.jsx	103	17	TeamID might not be needed
addPlayer.jsx	60	6	Input asks for grade not class
All			Commenting

Table 4.4

Inspection 4
Roles: Author: Collin Rodrigue
Recorder: Graham Bridges
Inspector: Brennan Poitras

File	**Line Number**	**Error #**	**Description**
home.jsx	3	16	Missing semicolon on import
teams.jsx	4	16	Missing semicolon on import
All files		16	Inconsistent styling(in-line vs separate css file)
Create React App files		8	Unused Create React App files

All files		14	Commenting

Table 4.5

Inspection 5
Roles: Author: Brennan Poitras
Recorder: Sean Radel
Inspector: Collin Rodrigue, Graham Bridges

File	**Line Number**	**Error #**	**Description**
index.js	1-9	16	Missing semicolons on import
index.js	16	16	Hardcoded value, should be an environment variable
dbConfig.js	3	14	Unnecessary comment
equipmentRoutes.js	all	16	Unconventional Whitespace
equipmentRoutes.js	All	14	Lack of commenting
loggingRoutes.js		14	Comments that aren't needed
Equipment.js	1,3	7	Unused imports
Log.js	31	16	Unnecessary console.log statement
Shoe.js	6-23	16	Poor naming convention of variables
playerControllers.js	1	16	Missing semicolon at the end of import
playerControllers.js	42	10	Should delete the equipment that's assigned to that player
Logging Controller.js	6,21	16	Unnecessary console.log statement
equipmentControllers.js	92-125	10	Deleting a piece of equipment should remove its assignment to players and teams
equipment.js	104-111	8	Unused function
equipmentControllers.js	197-205	17	Case 3 and case 4 don't have breaks in the switch statement which could cause an infinite loop
playerController.js	42-54	17	Code could be refactored to just call deletePlayerByID function

Table 4.6

Appendix A - Coding and Commenting Conventions

The code examples below show the standards that we are trying to uphold. All code modules can be found in our GitHub repository [3], as cited in section 1.

Commenting Convention

```
/*
Sean Radel
March 22, 2024
The purpose of this module is to define the Team class that represents -
properties like Team Name, Description of Team, Gender, Season, and SportID.

The class includes methods that can interact with the databse using SQL Queries

*/
```

Figure 1. Commenting Convention

Coding Convention

Front-end Convention example:

```
import React, { useEffect, useState } from "react";
import axios from "axios";
import { useNavigate, Link, useLocation } from "react-router-dom";
import './players.css';
import Modal from './Modal.jsx';

const Players = () => {
    const location = useLocation();
    const navigate = useNavigate();
    const { teamId, teamName } = location.state || {};

    const [showModal, setShowModal] = useState(false);
    const [deletePlayerId, setDeletePlayerId] = useState(null);

    const [players, setPlayers] = useState([]);
    const [classFilter, setClassFilter] = useState('All');
    const [nameFilter, setNameFilter] = useState('');
    const [ageFilter, setAgeFilter] = useState('');

    const initiateDelete = (playerID) => {
        setShowModal(true);
```

```
        setDeletePlayerId(playerID);
    };

    const fetchPlayers = async () => {
```

Figure 2. Front-end code example

Back-end Controllers example:

```
export const getAllTeams = async (req, res, next) => {

    try{
    let [team, _] = await Team.getAllTeams();

    res.status(200).json(team);
     } catch (error) {

        res.status(500).json({ error: 'Failed to get all teams' });
    }
}

export const createNewTeam = async (req, res, next) => {
    try{
    let { teamName, teamDesc, sportID, gender, season } = req.body;

    let team = new Team(null, teamName, teamDesc, sportID, gender, season);

    team = await team.createTeam();
    console.log(team)
    res.status(200).json(team);
    } catch (error) {

        res.status(500).json({ error: 'Failed to create new team' });
    }
}
```

Figure 3. Back-end code example - Controller

Back-end Models example:

```
async createTeam() {

        let sqlQuery = `
        INSERT INTO teams(
            teamName,
            teamDesc,
            sportID,
            gender,
            season
        )
        VALUES(
            '${this.teamName}',
            '${this.teamDesc}',
            '${this.sportID}',
            '${this.gender}',
            '${this.season}'
        )
        `;

        return promisePool.execute(sqlQuery);
    }
```

Figure 4. Back-end code example - Model

Back-end Routes coding convention example:

```
import express from 'express';
import { getAllTeams, createNewTeam, getTeamByID, updateTeam, deleteTeam,
getEquipmentByTeamID, assignEquipmentToTeam, removeEquipmentFromTeam } from
'../controllers/teamsControllers.js';
const router = express.Router();

router.route("/").get(getAllTeams).post(createNewTeam);
router.route("/:teamID").get(getTeamByID).delete(deleteTeam).put(updateTeam);
//equipment
router.route("/:teamID/equipment").get(getEquipmentByTeamID).post(assignEquipment
ToTeam);
router.route("/:teamID/equipment/:equipmentID").delete(removeEquipmentFromTeam);

export default router;
```

Figure 5. Back-end code example - Route

Database Schema example:

```
DROP TABLE IF EXISTS `apparel`;
/*!40101 SET @saved_cs_client     = @@character_set_client */;
/*!50503 SET character_set_client = utf8mb4 */;
CREATE TABLE `apparel` (
  `ID` smallint NOT NULL AUTO_INCREMENT,
  `brandName` varchar(255) NOT NULL,
  `equipmentID` smallint NOT NULL,
  `quantitySmall` int NOT NULL,
  `quantityMedium` int NOT NULL,
  `quantityLarge` int NOT NULL,
  `quantityXL` int NOT NULL,
  `quantity2XL` int NOT NULL,
  `quantity3XL` int NOT NULL,
  PRIMARY KEY (`ID`),
  CONSTRAINT `apparelEquipmentID` FOREIGN KEY (`equipmentID`) REFERENCES
`equipment` (`equipmentID`)
) ENGINE=InnoDB AUTO_INCREMENT=4 DEFAULT CHARSET=utf8mb4
COLLATE=utf8mb4_0900_ai_ci;
/*!40101 SET character_set_client = @saved_cs_client */;
```

Figure 6. Database code example

Appendix B – Team Review Sign-off

Team Agreement Sign Off

Team IMSG has thoroughly reviewed the Code Inspection Review document for the Athletic Inventory System and has agreed that the following information is accurate and achievable. Collectively we have no major contentions in the information stated in the document. By signing this agreement, one acknowledges all the terms and conditions outlined in the document and understands the importance of effective team collaboration, communication, and shared accountability when achieving the goals of the project. By signing below, we pledge our dedication to the success of the team and the project we plan to undertake. We agree to work collaboratively, and support each other to uphold the guidelines and expectations set forth in the agreement.

Signature:	Date:	Printed:
X: *Collin Rodrigue*	3/21/2024	Collin Rodrigue
X: *Gabriel A. Poulin*	3/21/2024	Gabriel A. Poulin
X: *Brennan Poitras*	3/21/2024	Brennan Poitras
X: *Sean Radel*	3/21/2024	Sean Radel
X:*Graham Bridges*	3/21/2024	Graham Bridges

Appendix C – Document Contributions

Name	Date	Contribution	Version
Gabriel Poulin	3/21/24	2.1 Description	1
Gabriel Poulin	3/21/24	Front-end coding style	1
Brennan Poitras	3/20/24	Back-end modules inspected	1
Collin Rodrigue	3/22/24	Front-end modules inspected	1
Graham Bridges	3/22/24	Front-end modules inspected	1
Sean Radel	3/05/24	Code convention	1
Sean Radel	3/05/24	Code Inspection, Section 1,	1
ALL	3/22/24	Appendix A, B, C	1
ALL	3/22/24	Impressions of the process	1
ALL	3/22/24	Defects	1