

Recreation Center Equipment Manager by figuring it out

- Adarsha Subedi
- Estuardo Marroquin
- Rob Pierce
- Shishir Paudel
- Skylar Shaffer

Introduction

About Team

Adarsha, an international Graduate student.

Estuardo, a Junior in Computer Science.

Rob, a Junior in Computer Engineering with a minor in VLSI.

Shishir, an international Graduate student.

Skylar, a Senior in Computer Science.

Project Overview

Recreation Center Equipment Management System (RCEMS) will allow users to book and manage recreational equipment online. Users will be asked to register and then be able to check availability and book items like bicycles and footballs online, while the system ensures real-time updates and reduces wait times. The administrators can easily track bookings and handle reports of damage. With our focus on convenience and efficiency, RCEMS will improve the overall experience for both users and administrators.

Elevator pitch

Recreation Center Equipment Management System (RCEMS) allows users to book and manage recreational equipment online. Users are asked to register and then can check availability and book items like bicycles and footballs online, while the system ensures real-time updates and reduces wait times. The administrators can easily track bookings and handle reports of damage. With our focus on convenience and efficiency, RCEMS will improve the overall experience for both users and administrators.

Overview - Recreation Center Equipment Management System (RCEMS)

Purpose: Recreation Center Equipment Management System aims to create an efficient platform for managing recreational equipment. It is designed for both students and admins processes related to equipment booking, availability tracking, and damage reporting.

Key features:

User login:

- Both students and admins can securely login with the respective credentials.
- Role-based access: Different access is granted based on the user role, ensuring appropriate functionality and data for the user.

Admin features:

- Inventory management: Add, update, remove equipment.
- Damage report handling: View and update equipment status based on reports.

Student features:

- Equipment booking: Access available equipment and reserve it remotely.
- Damage reporting: Report damaged equipment with descriptions and photos.

Overview - Architecture

1. Frontend (Next.js) (Estuardo, Skylar, Rob)

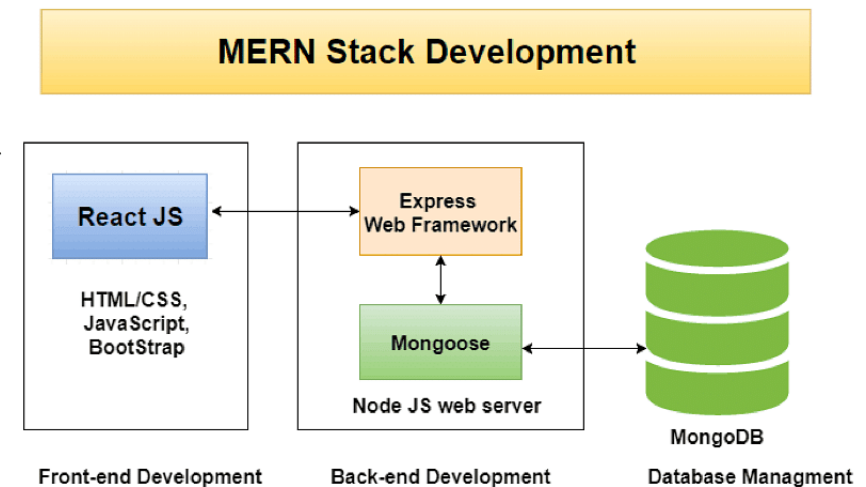
1. Components: handles UI rendering, user interactions and state management.
2. State Management Tools: Context API would be used for managing application states.
3. APIs: Using Frontend, components make requests to the backend through API to fetch and update data from the database.

2. Backend (Adarsha, Shishir)

1. Node.js : Runtime environment for JavaScript that is used to run server-side code.
2. Express.js: Framework used for handling HTTP requests and building API.

3. Database (Adarsha, Shishir)

1. Mongo DB: NoSQL database used to store the data for application.



Requirement 1 - Backend

User story: *As a Backend Developer, I want to manage equipment availability data through Create, Read, Update, and Delete (CRUD) operations, so the system can accurately reflect the current availability of equipment.*

1. Requirement 1.1: Data Input and Output for Equipment Availability
 1. Requirement 1.1.1: API to create new equipment availability records.
 2. Requirement 1.1.2: API to list of available equipment, including details like equipment ID, name, and availability status.
 3. Requirement 1.1.3: API to update and delete the availability of equipment.
2. Requirement 1.2: Database Integration for Equipment Availability
 1. Requirement 1.2.1: Store all equipment data in the MongoDB database.
 2. Requirement 1.2.2: Track equipment details in the database.
3. Requirement 1.3: Validation and Error Handling
 1. Requirement 1.3.1: Validate input data for equipment operations.
 2. Requirement 1.3.2: Return error messages for invalid data and return relevant status code.

Implementation

Node.js with Express.js: Handles backend functionality.

MongoDB: Stores and manages the data.

System Design

- Middleware: Checks input data and handles errors.
- Controllers: Manage CRUD operations and connect the APIs to the database.

MongoDB Collections:

- equipment: Stores data about all equipment, including ID, name, type, and availability status.
- admins: Tracks admin user details, login credentials and permissions.
- bookings: Logs information about equipment booking, the user, equipment ID, and booking dates.
- students: Contains student details, the name, and permissions.
- damagereports: Records reports about damaged equipment, including equipment ID, reported issue, and status of the report.

API flow

- Frontend Request: Sends a request to create, read, update, or delete equipment data.
- Middleware: Validates the request and ensures the data is correct.
- Controller Logic: Processes the request and communicates with the database.
- Response: Returns the results or error messages to the frontend.

Implementation Screenshot

```
const addEquipment = async (req, res) => {
  try {
    const { name, description, availabilityStatus, itemCount, equipmentID } =
      req.body;

    if (!name || !description) {
      return res.status(400).json({
        success: false,
        message: "Please provide name and description of the equipment.",
      });
    }

    const newEquipment = new equipment({
      name,
      description,
      itemCount,
      equipmentID,
      availabilityStatus:
        availabilityStatus !== undefined ? availabilityStatus : true,
    });

    const savedEquipment = await newEquipment.save();

    res.status(201).json({
      success: true,
      data: savedEquipment,
    });
  } catch (error) {
    console.error("Error adding equipment:", error);

    res.status(500).json({
      success: false,
      message: "Server Error: Unable to add Equipment.",
    });
  }
};
```

What we did

POST http://localhost:5000/api/admin/addEquipment

Params Authorization Headers (9) Body Scripts Tests Settings

☐ none ☐ form-data ☐ x-www-form-urlencoded ☒ raw ☐ binary ☐ GraphQL

```
1 {
2   "id": "674679ea482cbbdf75eee325",
3   "name": "cycle",
4   "description": "Lets go for a ride",
5   "itemCount": "20",
6   "equipmentID": "9"
7 }
```

Body Cookies Headers (8) Test Results ↺

Pretty Raw Preview Visualize JSON ↕

```
1 {
2   "success": true,
3   "data": {
4     "name": "cycle",
5     "description": "Lets go for a ride",
6     "equipmentID": "9",
7     "itemCount": "20",
8     "availabilityStatus": true,
9     "_id": "674e81513d4497c252f30a3e",
10    "createdAt": "2024-12-03T03:56:01.178Z",
11    "updatedAt": "2024-12-03T03:56:01.178Z",
12    "v": 0
  }
```

What we get

Requirement 2 - User authentication

User story: *As a registered user, I want to log in so I can view the available equipment and be able to book a reservation.*

1. Requirement 2.1: User Authentication

1. Requirement 2.1.1: Login form that accepts a username and password.
2. Requirement 2.1.2: Validate the credentials with the database.

2. Requirement 2.2: Viewing Available Equipment

1. Requirement 2.2.1: Display a list of available equipment post-login.
2. Requirement 2.2.2: The equipment list shall include details such as name, description, availability status, and booking options.

3. Requirement 2.3: Booking a Reservation

1. Requirement 2.3.1: An interface to select equipment, dates, and times for the reservation.

Design

Sign-Up Page:

- Clear labels for user inputs.
- Required fields: student email, full name, password, and confirm password.
- A "Sign Up" button is provided to submit the form.
- Option to redirect to the login page for existing users.

Login Page:

- Interface with input fields for email/username and password.
- Buttons for "Login" and "Create New Account."
- Includes a "Forget Password?" link for recovery.

A wireframe of a login page within a web browser window. The browser's address bar shows "http://". The page header includes "Company Logo" on the left and navigation links "Home", "Equipments", "Bookings", and "Settings" on the right. The main content area is divided into two sections. The left section, titled "Login", contains a text box with the placeholder "To log in, please enter your email address and password.", followed by input fields for "Email/Username" and "Password". Below these are two buttons: "Login" and "Create New Account", and a link "Forget Password?". The right section is a large square placeholder with a diagonal 'X'. The footer contains the word "Footer" on the left and a block of small, illegible text on the right.

A wireframe of a sign-up page within a web browser window. The browser's address bar shows "http://". The page header includes "Company Logo" on the left and navigation links "Home", "Equipments", "Bookings", and "Settings" on the right. The main content area is divided into two sections. The left section, titled "Sign up", contains a text box with the placeholder "Create a User Account", followed by input fields for "Student Email", "Full Name", "Password", and "Confirm Password". Below these are a "Signup" button and a link "Already have an account? Click here". The right section is a large square placeholder with a diagonal 'X'. The footer contains the word "Footer" on the left and a block of small, illegible text on the right.

Implementation

Sign up page:

- Form containing text inputs, a drop down for user role, and a submit button.
- Passwords are hashed using bcrypt, and a new user is created.
- If the form is valid, then we try to store the new user into the database.

Sign Up

Username

Password

Confirm Password

Role
Student ▼

[Sign Up](#)

[Already have an account?](#)

Login

To log in, please enter your email address, password, and select your role.

Email/Username

Password

Role
Student ▼

[Login](#) [Create New Account](#)

[Forget Password?](#)

```
// Hash the password
const hashedPassword = await bcrypt.hash(password, 10);

// Create the new user
const createdUser = await userModel.create({
  username: username,
  password: hashedPassword,
  role: role,
});
```

```
const handleSubmit = async (e) => {
  e.preventDefault();

  // Validate form
  if (!validateForm()) {
    return; // Don't submit the form if validation fails
  }

  try {
    const response = await axios.post(
      "http://localhost:5000/api/auth/signUP",
      {
        username: formData.username,
        password: formData.password,
        role: formData.role,
      }
    );
  }
};
```

Requirement 3 - Frontend

***User story:** As a Frontend developer, I want to know what information I'm going to receive and send back to design an effective and appropriate User Interface.*

1. Requirement 3.1: Data Input and Output for Login

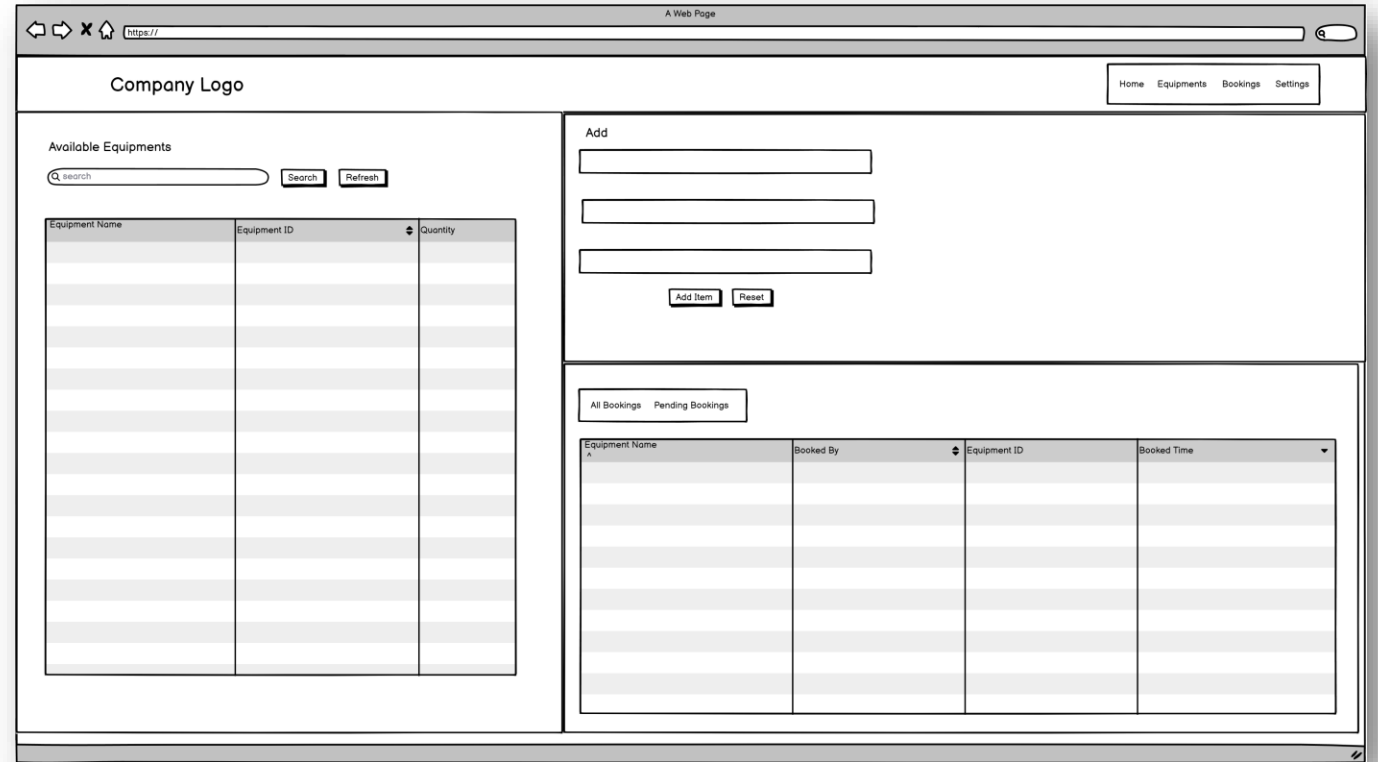
1. Requirement 3.1.1: Send the username and password to the backend for authentication.
2. Requirement 3.1.2: Display a success or failure response based on the login attempt.

2. Requirement 3.2: Storing Data for Available Equipment

1. Requirement 3.2.1: Provide a response that includes a list of available equipment with names, descriptions, and availability Status.
2. Requirement 3.2.2: The frontend shall be able to process and display this data in an organized way.

Design

- Equipment:
 - Our original design for the admin panel included the Available Equipment table, Add Equipment form, and bookings list.
 - During the development process these were split up.
 - The Available Equipment table and Add Equipment were placed into the Equipment Dashboard page.
 - The bookings table was moved to the Equipment Bookings page.
 - The separation of these entities allowed for the website to seem less cluttered.



Implementation

Add Equipment:

- Form with text inputs for the equipment name, description, ID, and quantity.
- Equipment name, human readable as opposed to just an ID.
- Add/Cancel buttons, to submit or discard the entered equipment information.

Available Equipment:

- Table with Equipment Name, ID, Quantity, and actions.
- Equipment Name and Quantity, showing what and how many pieces of equipment are available.
- Edit/Delete actions, allow admins to change the quantity or full remove a type of equipment.

Available Equipments

Add Equipment

Equipment Name	Equipment ID	Quantity	Actions	
Soccer Ball	123	10	Edit	Delete
Tennis Racket	124	10	Edit	Delete
Bicycle	125	5	Edit	Delete

Add Equipment

Cancel

Add

Requirement 4 - Equipment Booking System

User story: As a student user, I want to be able to check the availability of equipment online so that I book them remotely.

1. Requirement 4.1: Accessing the Equipment listing

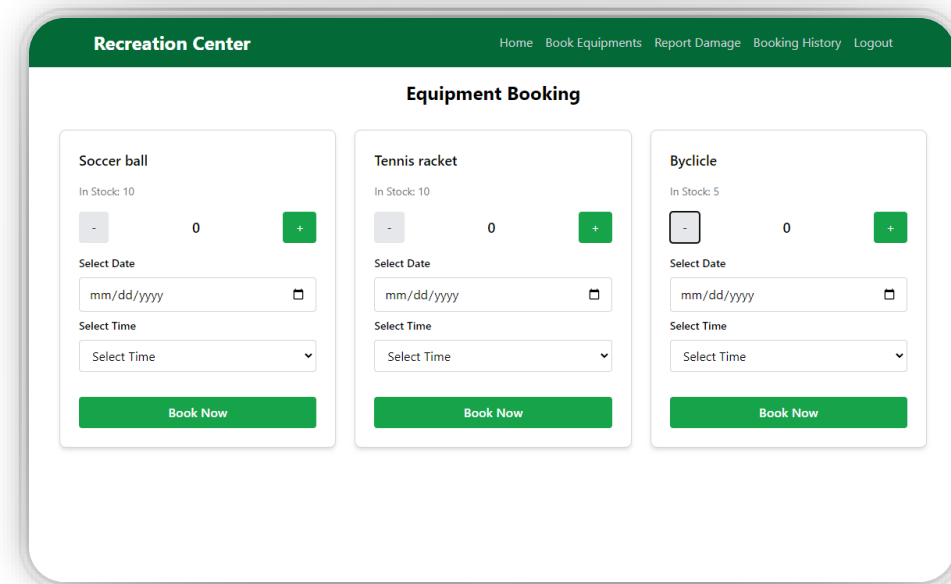
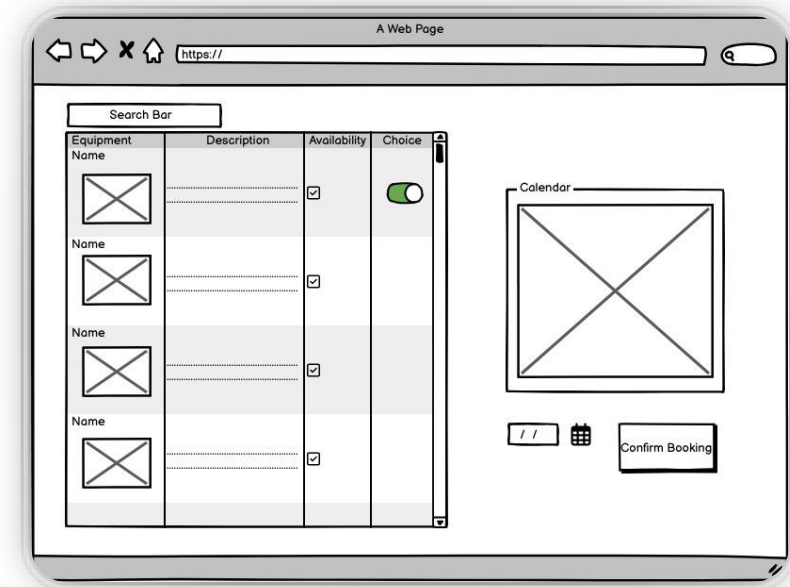
1. Requirement 4.1.1: The user shall be required to log in with student credentials to access the Interface

2. Requirement 4.2: Booking an equipment

1. Requirement 4.2.1: The Interface shall show the list of available equipment
2. Requirement 4.2.2: The Interface shall have a button to book the equipment
3. Requirement 4.2.3: The Interface shall show which equipment is currently booked by the student
4. Requirement 4.2.4: The Interface shall show the deadline to submit the booked equipment

Design

The first design was confusing because it didn't clearly label things or visually separate different parts of the page. It was hard for users to understand what each section was for and how to interact with it. For example, it wasn't clear what the checkboxes and buttons were for. The new design is much better. It's easy to understand, visually appealing, and provides all the necessary information for users to book equipment.



Implementation

Equipment Booking Page:

- Includes date and time fields, and quantity selection.
- Date and Time Selection: Date and time selection is provided, allowing for easy selection of booking times.
- Quantity Selection: Select the desired quantity of equipment.
- Book Button: A button is provided to start the booking process.
- Error Handling: The page displays informative error messages if the user tries to book unavailable equipment or selects invalid dates/times.
- Confirmation Messages: After a successful booking, a confirmation message is displayed, providing details of the booking.

Recreation Center Home Book Equipments Report Damage Booking History Logout

Equipment Booking

Soccer Ball
In Stock: 10
- 0 +
Select Date
mm/dd/yyyy
Select Time
Select Time
Book Now

Tennis Racket
In Stock: 10
- 0 +
Select Date
mm/dd/yyyy
Select Time
Select Time
Book Now

Bicycle
In Stock: 10
- 0 +
Select Date
mm/dd/yyyy
Select Time
Select Time
Book Now

! Please select both date and time!

✓ Booking successful! Equipment must be returned by 6:00:00 PM on the selected day. Late returns will incur a fine of \$20.

! You have already booked this equipment for the selected time slot.

Requirement 5 -Breakage/Damage Reporting

- **User story:** *As a student user, I want to be able to report damages so I can ensure maintenance fixes equipment for the next student.*

1. Requirement 5.1: Damage Reporting Interface

1. Requirement 5.1.1: The website shall provide a damage reporting form that allows users to select the equipment they want to report damage for.
2. Requirement 5.1.2: The form shall include fields for users to describe the type of damage, its severity, and an optional photo upload.
3. Requirement 5.1.3: The form shall require users to submit their student identification information with the report.

2. Requirement 5.2: Submission and Confirmation

1. Requirement 5.2.1: Upon form submission, the website shall confirm the report by displaying a message that the damage has been successfully reported.
2. Requirement 5.2.2: The damage report shall be sent to the backend, which stores the report in the database for review by the maintenance team.

Design

Damage Reporting form:

- **Equipment ID:** A text input field where the user would enter the ID of the damaged equipment.
- **Damage Description:** A text area where the user would describe the damage in detail.
- **Upload Image:** A file input field where the user could optionally upload an image of the damage.
- Once the user has filled out the form, they would click the "Submit Damage Report" button.

A wireframe of a web browser window titled "A Web Page" showing a "Report Damage" form. The form is contained within a rectangular box and includes the following elements: a title "Report Damage", a text input field labeled "Equipment ID", a text area labeled "Damage Description", a file input field labeled "Upload Image (Optional)" with a "Choose File" button, and a "Submit Damage Report" button.

A mockup of a web application titled "Recreation Center" with a green header bar. The header contains navigation links: "Home", "Book Equipments", "Report Damage", "Booking History", and "Logout". The main content area is light gray and features a "Report Damage" form. The form is styled with a white background and rounded corners. It includes the following elements: a title "Report Damage", a text input field labeled "Equipment ID" with placeholder text "Enter Equipment ID", a text area labeled "Damage Description" with placeholder text "Describe the damage...", a file input field labeled "Upload Image (Optional)" with a green "Choose File" button and the text "No file chosen", and a green "Submit Damage Report" button.

A conceptual image featuring a glass hourglass with white sand, positioned over a calendar. The hourglass is tilted, showing sand falling from the top bulb into the bottom bulb. The calendar is partially visible on the right side, showing dates 22, 23, 24, 29, 30, and 31. The entire scene is overlaid with a semi-transparent dark grey filter.

Testing Time

Testing 1

Functional Testing Using Jest and Super Test for User Auth Functionality

```
const request = require("supertest");
const mongoose = require("mongoose");
const { MongoMemoryServer } = require("mongodb-memory-server");
const app = require("express")(); // Minimal Express instance for testing
const authRouter = require("../routes/auth.router");
require("dotenv").config();

// Use middleware to parse JSON
app.use(require("express").json());
app.use("/auth", authRouter);

let mongoServer;

beforeAll(async () => {
  // Start in-memory MongoDB
  mongoServer = await MongoMemoryServer.create();
  const uri = mongoServer.getUri();

  // Connect Mongoose to the in-memory server
  await mongoose.connect(uri, { useNewUrlParser: true, useUnifiedTopology: true });
});

afterAll(async () => {
  // Close Mongoose connection and stop in-memory MongoDB
  await mongoose.disconnect();
  await mongoServer.stop();
});

describe("Admin Controller", () => {
  const adminSignup = {
    username: "adminUser",
    password: "adminPass",
    role: "admin",
  };

  const studentSignup = {
    username: "studentUser",
    password: "studentPass",
    role: "student",
  };

  const loginData = {
    username: "adminUser",
    password: "adminPass",
    role: "admin",
  };
});
```

```
it("POST /auth/signup - Should create a new admin user", async () => {
  const res = await request(app).post("/auth/signup").send(adminSignup);
  expect(res.status).toBe(201);
  expect(res.body.success).toBe(true);
  expect(res.body.user).toHaveProperty("username", adminSignup.username);
});

it("POST /auth/signup - Should create a new student user", async () => {
  const res = await request(app).post("/auth/signup").send(studentSignup);
  expect(res.status).toBe(201);
  expect(res.body.success).toBe(true);
  expect(res.body.user).toHaveProperty("username", studentSignup.username);
});

it("POST /auth/signup - Should not create a duplicate user", async () => {
  const res = await request(app).post("/auth/signup").send(adminSignup);
  expect(res.status).toBe(409);
  expect(res.body.success).toBe(false);
  expect(res.body.message).toBe("User already exists with this username");
});

it("POST /auth/login - Should authenticate admin with correct credentials", async () => {
  const res = await request(app).post("/auth/login").send(loginData);
  expect(res.status).toBe(200);
  expect(res.body.success).toBe(true);
  expect(res.body).toHaveProperty("token");
});

it("POST /auth/login - Should reject login with invalid credentials", async () => {
  const res = await request(app).post("/auth/login").send({
    username: "adminUser",
    password: "wrongPass",
    role: "admin",
  });
  expect(res.status).toBe(200);
  expect(res.body.success).toBe(false);
  expect(res.body.message).toBe("Credentials Not Valid");
});

it("POST /auth/reset - Should update admin password with correct current password", async () => {
  const res = await request(app).post("/auth/reset").send({
    username: "adminUser",
    password: "adminPass",
    newPassword: "newAdminPass",
    role: "admin",
  });
  expect(res.status).toBe(200);
  expect(res.body.success).toBe(true);
  expect(res.body.message).toBe("Updated successfully");
});
```

Test Outcome

PASS `__tests__/adminController.test.js`

Admin Controller

- ✓ POST /auth/signup - Should create a new admin user (111 ms)
- ✓ POST /auth/signup - Should create a new student user (59 ms)
- ✓ POST /auth/signup - Should not create a duplicate user (5 ms)
- ✓ POST /auth/login - Should authenticate admin with correct credentials (58 ms)
- ✓ POST /auth/login - Should reject login with invalid credentials (54 ms)
- ✓ POST /auth/reset - Should update admin password with correct current password (112 ms)
- ✓ POST /auth/reset - Should reject update with incorrect current password (54 ms)

Test Suites: **1 passed**, 1 total

Tests: **7 passed**, 7 total

Snapshots: 0 total

Time: 2.691 s, estimated 3 s

Ran all test suites.

Testing 1: Integration Testing for user Auth

Feature	Test Case ID	Test Case Description	Test Steps	Result	Remarks
Signup	TC001	Test if the Signup page works properly	1. Clicking 'Create New Account' button redirects you to signup page 2. Username field is a required field and takes string 3. Password field is an required field and takes string 4. User can select to create either student account or admin account 5. Should not create a duplicate user	1. Clicking 'Create New Account' redirected to signup page 2. Username field did not take an empty input 3. Password field did not take an empty input 4. User could select to create either student or admin account 5. Throws error while tryin to create a duplicate user	Pass
Login	TC002	Test if the Login page works Properly	1. Username and Password field should not take an empty input. 2. User Should not be logged in if the user selects incorrect role. 3. Should authenticate admin with correct credentials 4. Should reject login with invalid credentials 5. Should update admin password with correct current password 6. Should reject update with incorrect current password	1. Username and Password field did not take an empty input 2. User did not get logged in if the user selected incorrect role. 3. Authenticated admin with correct credentials 4. Rejected with invalid credentials 5. Updates admin password with correct current password 6. Rejects update with incorrect password	Pass
Access of Functionality	TC003	Test if Admin and Student get different functionality	1. Check if the site redirects to different page according to the role	1. Site redirecded to different page according to the role	Pass
	TC004	Test if Any other functionality of site cannot be accessed without logging in first	1. Redirect to the login page if the user is trying to access the functionality without login in	1. Redirects to login page if not logged in	Pass

Testing Outcome Screenshot

Sign Up

Username

Username is required


Password

Password is required


Confirm Password


Please confirm your password

Role





User already exists with this username






Login Success!





Access denied: Admins cannot access student pages.



Login

To log in, please enter your email address, password, and select your role.

Email/Username

Username is required

Password

Password is required

Role

[Forget Password?](#)

Login

To log in, please enter your email address, password, and select your role.

Email/Username

Password

Role

User Not Found !!

Testing 2

Feature	Test Case ID	Test Case Description	Test Steps	Result	Remarks
Add Equipment	TC005	Test if Admin can add the equipment	1. Any Empty Field should not allow admin to add equipment 2. Added equipment should be reflected on the list 3. The added equipment should be shown on the Student portal as well	1. Empty field did not allow admin to add new equipment 2. The added equipment got reflected on the list 3. The added equipment showed up on the student portal	Pass
Edit Equipment	TC006	Test if Admin can edit the equipment	1. The system should show the form containing the current data that should be edited 2. The list should be updated with the new data after saving 3. The edited data should also be reflected on the student portal	1. The system displayed form populated with previous data 2. The list got updated with the new data 3. The updated equipment was properly reflected on the student portal	Pass
Delete Equipment	TC007	Test if Admin can delete the equipment	1. The user should be prompted with pop up warning before deleting 2. The user should be able to click the button for the pop-up 3. After approving the deletion, the equipment should be removed from the table 4. Deleted equipment should not be shown	1. The user got prompted with pop warning after clicking delete button 2. The user was able to click the button on the pop-up 3. The equipment got removed from the table 4. The deleted equipment did not show on the student portal	Pass

Testing Outcome Screenshot

Edit Equipment

Name is required

Description is required

Equipment ID is required

Quantity is required

Cancel

Update

Edit Equipment

Cancel

Update

✓

Equipment added successfully!

✕

Are you sure you want to delete this equipment?

Cancel

Delete

Available Equipments

Add Equipment

Equipment Name	Equipment ID	Quantity	Actions
Car	2224	33	Edit Delete
Byecycle	0012	2	Edit Delete

Conclusion

What we've created

- A simple website to book and manage recreational equipment
- Easy to use for both students and admins

Project impact

- Simplifies booking and tracking, it makes it easier for students to book equipment and admins to keep track of their inventory.
- Improves user experience, it has an easy to navigate interface.
- Enhances maintenance efficiency, the damage reporting feature prevents damaged equipment from being used
- Real-time updates, students and admins will receive up-to-date information about equipment, which allows smooth operations and reduces scheduling conflicts



Questions?

Contact us:

- Adarsha – subedi.34@wright.edu
- Estuardo – marroquin.4@wright.edu
- Rob – pierce.60@wright.edu
- Shishir – paudel.29@wright.edu
- Skylar – shaffer.125@wright.edu