

D424 – SOFTWARE CAPSTONE
TASK 3



Capstone Proposal Project Name:

Employee Management System (EMS)

Student Name:

Bradley Summers

TABLE OF CONTENTS

D424 – Software Capstone Task 3	1
Application Design and Testing.....	4
Class Design.....	4
UI Design	5
Unit Test Plan	9
Introduction	9
Purpose	9
Overview	9
Test Plan.....	9
Items	9
Features	9
Deliverables.....	10
Tasks.....	10
Needs	10
Pass/Fail Criteria	10
Specifications	10
Procedures	13
Results.....	13
Hosted Web Application	13
Hosted Web Application Link.....	13
GitLab Repository & Branch History.....	13
GitLab Repository Link	13
GitLab Branch History	13
User Guide – Deployment & Setup	14
Introduction	14
Database Deployment	15
Backend Deployment.....	15
Dockerizing the Backend Application	15
Deploying the Backend to Render.com	17
Frontend Deployment.....	18
Custom Domain Configuration	20
Setup	21

User Guide – Application Usage	22
Login.....	22
Employees.....	23
Viewing Employee List	23
Adding a New Employee	24
Editing an Employee	25
Deleting an Employee	27
Departments	28
Viewing Department List.....	28
Adding a New Department	29
Editing a Department.....	30
Deleting a Department	31
Reports.....	32
Viewing the Organizational Chart	32
Application Navigation and User Profile	34
Using the Navigation Menu	34
Viewing Your User Profile	34
Logging Out	35

APPLICATION DESIGN AND TESTING

CLASS DESIGN

The Employee Management System (EMS) is built using a layered architecture approach with clear separation of concerns. The class design follows object-oriented principles with entities representing the core domain objects, controllers handling HTTP requests, services implementing business logic, and repositories managing data access.

The system is centered around two primary entities: Employee and Department. The Employee entity represents users of the system with attributes such as name, email, role, and department affiliation. The Department entity represents organizational units that employees belong to within the company structure.

The system implements a hierarchical employee structure where employees can have managers, creating a reporting relationship that forms the basis of the organizational chart visualization. Authentication is handled through a JWT-based system that provides secure access with role-based permissions. The application supports two distinct roles: ADMIN and EMPLOYEE, each with appropriate access levels to system functionality. The class diagram below illustrates the backend Java/Spring Boot architecture of the system. The frontend is implemented using React and TypeScript, which follows a component-based architecture rather than a traditional class structure, making it less suitable for representation in a class diagram format.

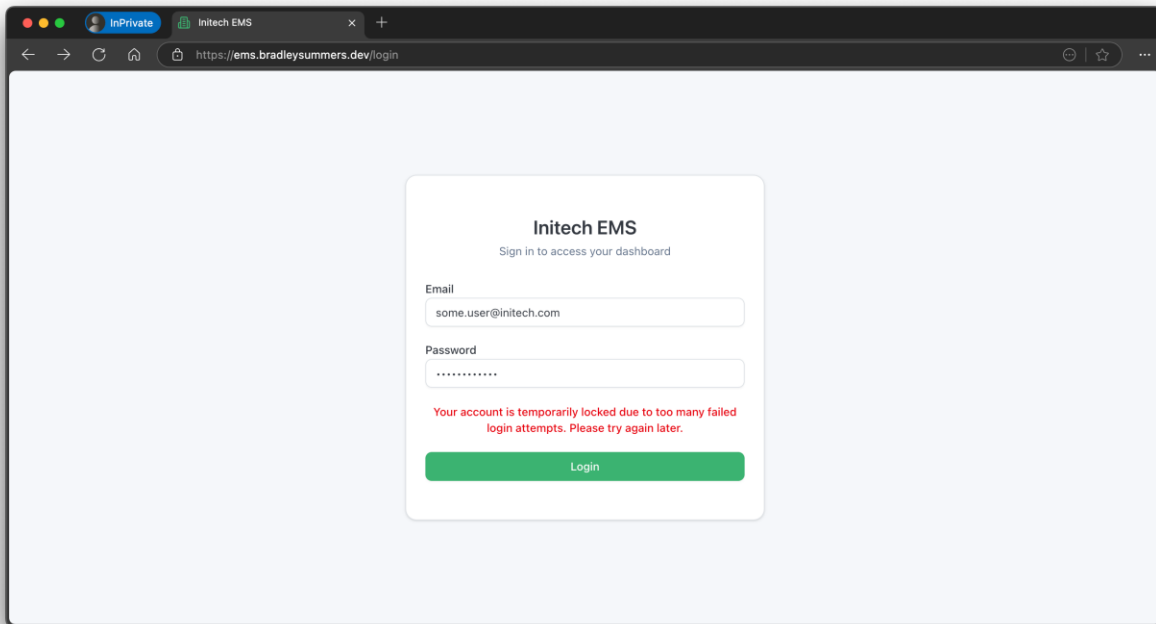


UI DESIGN

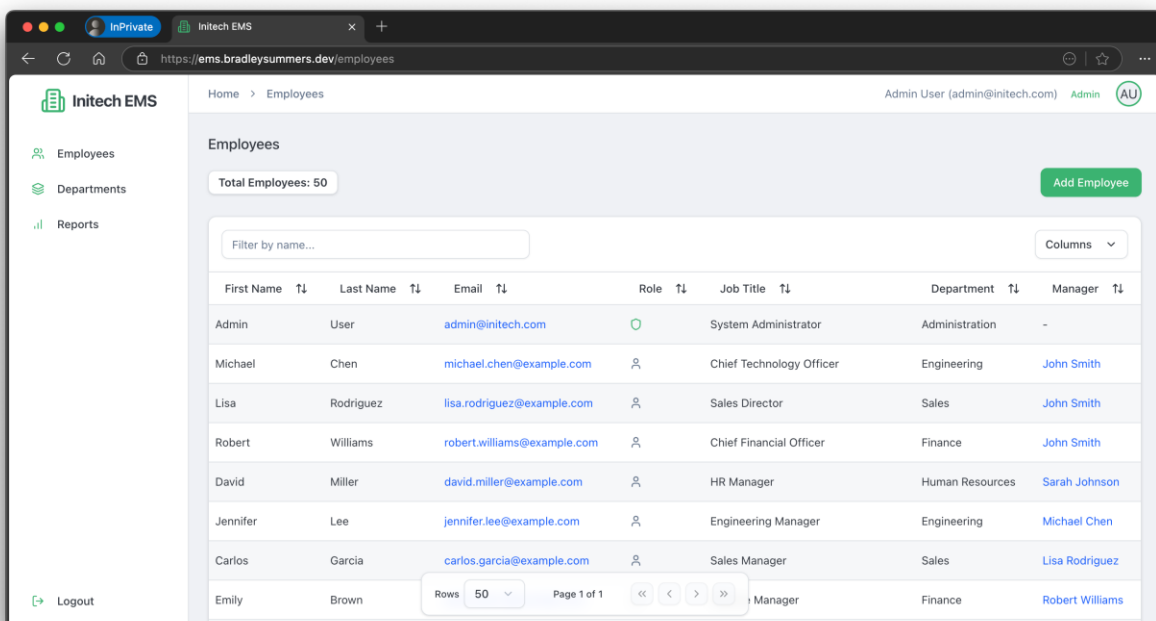
The Initech Employee Management System features a clean, functional frontend interface built with React and TypeScript. It uses Shadcn UI components to ensure visual consistency across the application and reduce development time by leveraging customizable, pre-built elements.

The login screen presents a minimal and focused entry point to the application. It features a centered card layout with clearly labeled fields for email and password, along with a prominent green "Login" button. To maintain a secure and controlled environment, account registration and password recovery options are intentionally omitted, as all user access is managed directly by administrators. The system includes error handling for common authentication issues such as invalid credentials, account lockouts

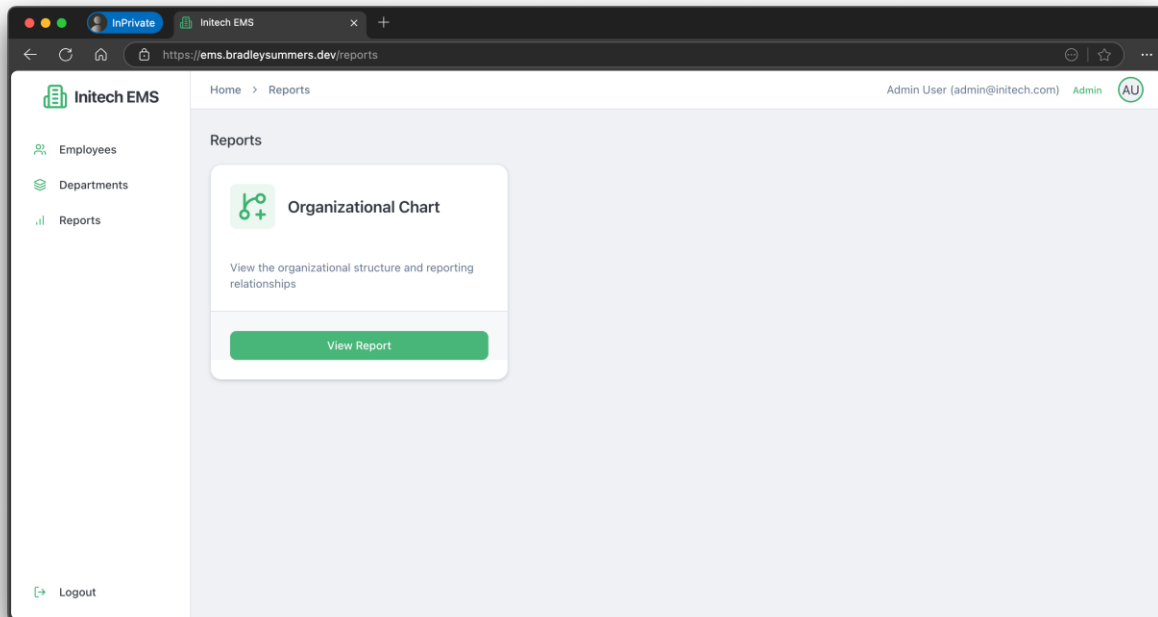
due to too many failed login attempts, and disabled accounts, all with clear, user-facing messages.



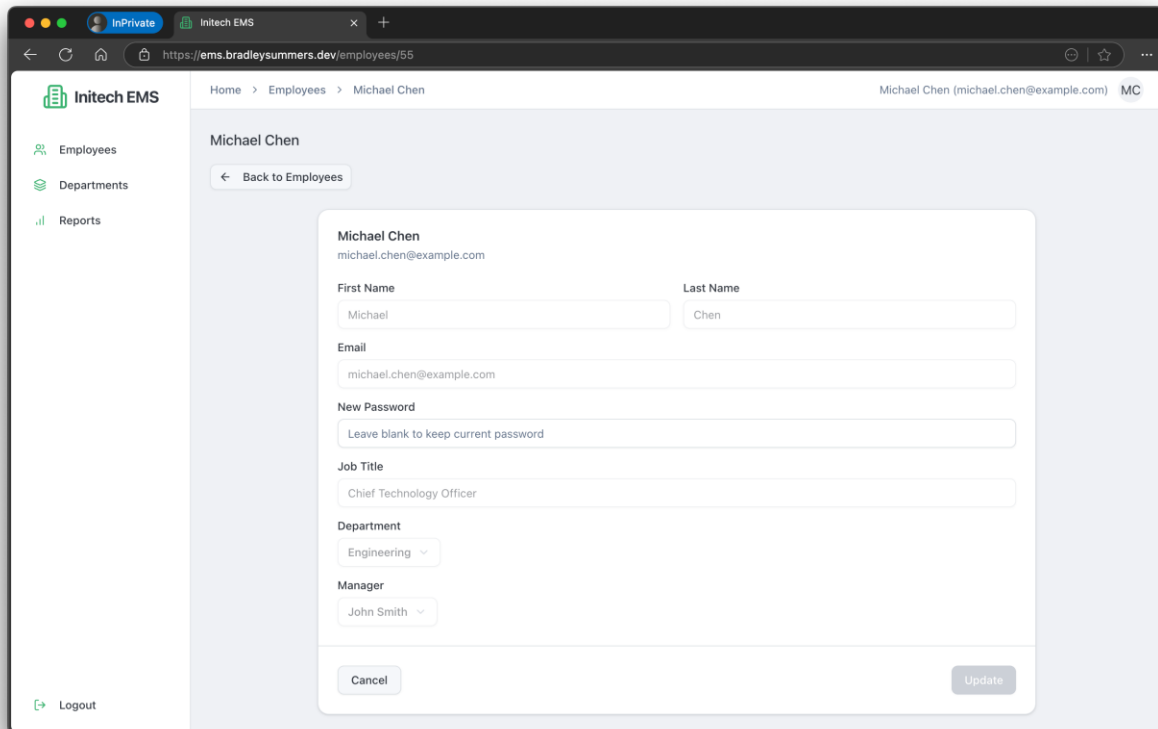
Navigation is organized around a persistent sidebar that provides access to three main sections: Employees, Departments, and Reports. The Employees page serves as the default landing screen, displaying a comprehensive list of all personnel. Like the Departments page, it uses a custom DataTable built on Shadcn's implementation of the TanStack Table library, enabling powerful data handling features such as sorting, filtering, and pagination. Both the Employees and Departments pages follow a consistent layout and interaction model, promoting a seamless experience when working with different types of data.



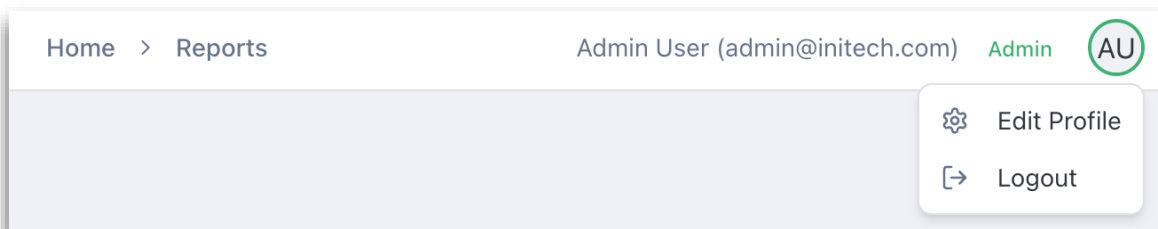
The Reports page uses a modular "pod" layout, allowing users to select from different report types. Currently, it includes the Organizational Chart Report, which visually represents the company's hierarchy, making reporting relationships easy to understand. Additional pods for new reports can easily be added to this page to extend this functionality.



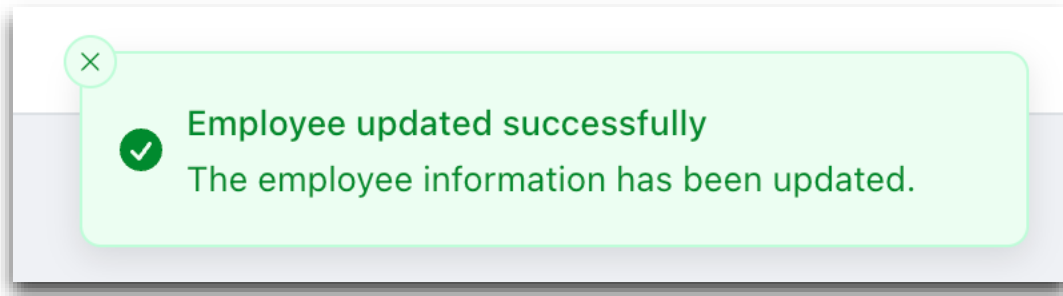
A comprehensive role-based access control system dynamically adapts the interface based on the user's permissions. ADMIN users have full system access, including exclusive features like "Add Employee" and "Add Department" buttons, which are hidden from regular users. In data tables, ADMINS can view all records, while EMPLOYEE users see only their own information and directly related data. This access model extends to detail pages, where form fields are rendered as editable or read-only depending on the user's role. For example, EMPLOYEE users can change their own password but cannot modify other profile details like name, job title, or reporting structure, which require administrative privileges. This granular control ensures that users only interact with the parts of the system relevant to their role.



To support intuitive navigation, breadcrumbs are positioned at the top of each page, helping users understand their current location and easily move backward. In the top-right corner, the interface shows the logged-in user's name and email alongside a profile badge with their initials. This badge includes a dropdown menu with options to update profile details or log out.



To provide immediate feedback throughout the application, toast notifications are used to confirm the outcome of key user actions like adding, editing, or deleting employees or departments. These appear in the top center of the screen and automatically dismiss after a few seconds. Success and error states are clearly styled and provide concise, relevant feedback without interrupting the user's workflow.



UNIT TEST PLAN

INTRODUCTION

PURPOSE

The unit tests in this project were written using the JUnit 5 testing framework. These tests were designed to verify the core logic of employee creation and validation within the Employee Management System (EMS). The goal of these tests was to ensure that the Employee entity behaves as expected, including correct data assignment and equality checks, and that login and employee validation functions perform correctly. All tests passed successfully, and no remediation was required.

OVERVIEW

The unit tests were created for two primary components of the system: the Employee entity and the validation logic for login and employee data. These tests did not require the Spring framework to be initialized, which allowed them to run quickly and independently. The `EmployeeEntityTest` focused on verifying that the builder pattern created valid Employee objects and that the equality method correctly distinguished between employees based on field comparisons. The `EmployeeValidationTest` validated login requests and employee data by checking for required fields such as email, first name, last name, and password. All validation logic was encapsulated in helper methods within the test file. Error handling was managed through assertions, which failed the test if expected conditions were not met.

TEST PLAN

ITEMS

To complete the tests, the Java Development Kit (JDK) and a properly configured development environment with JUnit 5 support were required. The tests were executed using IntelliJ IDEA with the JDK version Corretto 21.0.6.

FEATURES

The tests covered the following features:

- Employee creation using the builder pattern
- Employee equality comparison

- Validation of login requests for empty and valid credentials
- Validation of employee data for required fields

DELIVERABLES

The deliverables included the source code for the test classes (EmployeeEntityTest.java and EmployeeValidationTest.java) and documentation in the form of assertions and comments. Additionally, a screenshot of the successful test run was included as visual confirmation of the test results.

TASKS

The primary tasks included writing test cases for the employee builder, testing the equality method, and implementing and verifying custom validation logic for employee and login data. Each test was executed and observed for pass/fail outcomes. All tests returned successful results, as indicated by green checkmarks in the test runner.

NEEDS

The testing process required the following:

- JDK 21 (Amazon Corretto 21.0.6)
- JUnit Jupiter (JUnit 5)
- IntelliJ IDEA as the development environment No external libraries were used beyond JUnit, as the validation logic was manually implemented in helper methods within the test files.

PASS/FAIL CRITERIA

A test was considered successful if all assertions within the test method passed without throwing exceptions. The IntelliJ test runner provided immediate visual feedback. In the case of failure, the corresponding test would be marked in red and the exception stack trace would be used for debugging. Since all tests passed, no remediation was required, but any failure would have been addressed by reviewing and correcting the logic or expected values in the test.

SPECIFICATIONS

Below is the actual test code used in the application:

```
EmployeeEntityTest.java

package dev.bradleysummers.ems;

import dev.bradleysummers.ems.entity.Employee;
import dev.bradleysummers.ems.enums.Role;
import org.junit.jupiter.api.Test;

import static org.junit.jupiter.api.Assertions.*;

/**
 * Tests for the Employee entity
 */
public class EmployeeEntityTest {  ⚡ Bradley Summers

    @Test  ⚡ Bradley Summers
    void employeeBuilder_ShouldCreateValidEmployee() {
        // Create an employee using the builder pattern
        Employee employee = Employee.builder()
            .firstName("John")
            .lastName("Doe")
            .email("john.doe@example.com")
            .password("password123")
            .role(Role.EMPLOYEE)
            .active(true)
            .build();

        // Verify the employee properties
        assertEquals("John", employee.getFirstName());
        assertEquals("Doe", employee.getLastName());
        assertEquals("john.doe@example.com", employee.getEmail());
        assertEquals("password123", employee.getPassword());
        assertEquals(Role.EMPLOYEE, employee.getRole());
        assertTrue(employee.isActive());
    }

    @Test  ⚡ Bradley Summers
    void employeeEquality_ShouldCompareAllRelevantFields() {
        // Create two identical employees
        Employee employee1 = Employee.builder()
            .id(1L)
            .firstName("John")
            .lastName("Doe")
            .email("john@example.com")
            .build();

        Employee employee2 = Employee.builder()
            .id(1L)
            .firstName("John")
            .lastName("Doe")
            .email("john@example.com")
            .build();

        // Create a different employee with same ID but different fields
        Employee employee3 = Employee.builder()
            .id(1L)
            .firstName("Jane")
            .lastName("Smith")
            .email("jane@example.com")
            .build();

        // Test equality
        assertEquals(employee1, employee1, "message: \"An employee should be equal to itself\"");
        assertEquals(employee1, employee2, "message: \"Employees with identical fields should be equal\"");
        assertNotEquals(employee1, employee3, "message: \"Employees with different fields should not be equal\"");

        // Test with different ID
        Employee employee4 = Employee.builder()
            .id(2L)
            .firstName("John")
            .lastName("Doe")
            .email("john@example.com")
            .build();

        assertNotEquals(employee1, employee4, "message: \"Employees with different IDs should not be equal\"");
    }
}
```

```
EmployeeValidationTest.java
package dev.bradleysummers.ems;

import dev.bradleysummers.ems.dto.auth.AuthRequest;
import dev.bradleysummers.ems.entity.Employee;
import dev.bradleysummers.ems.enums.Role;
import org.junit.jupiter.api.Test;

import static org.junit.jupiter.api.Assertions.*;

/**
 * Simple unit tests for employee validation that don't require Spring context
 */
public class EmployeeValidationTest {

    @Test
    void validateLoginRequest_EmptyPassword() {
        // Create a login request with empty password
        AuthRequest request = new AuthRequest();
        request.setEmail("test@example.com");
        request.setPassword("");

        // Validate the request
        boolean isValid = isValidLoginRequest(request);

        // Assert that the request is invalid
        assertFalse(isValid, "Login request with empty password should be invalid");
    }

    @Test
    void validateLoginRequest_ValidCredentials() {
        // Create a login request with valid credentials
        AuthRequest request = new AuthRequest();
        request.setEmail("test@example.com");
        request.setPassword("password123");

        // Validate the request
        boolean isValid = isValidLoginRequest(request);

        // Assert that the request is valid
        assertTrue(isValid, "Login request with valid credentials should be valid");
    }

    /**
     * Helper method to validate login request
     */
    private boolean isValidLoginRequest(AuthRequest request) {
        return request != null
            && request.getEmail() != null && !request.getEmail().isEmpty()
            && request.getPassword() != null && !request.getPassword().isEmpty();
    }

    @Test
    void validateEmployee_RequiredFields() {
        // Create an employee with missing required fields
        Employee incompleteEmployee = Employee.builder()
            .email("test@example.com")
            // Missing firstName and lastName
            .build();

        // Create a complete employee
        Employee completeEmployee = Employee.builder()
            .email("test@example.com")
            .firstName("John")
            .lastName("Doe")
            .password("password123")
            .role(Role.EMPLOYEE)
            .build();

        // Validate both employees
        boolean incompleteIsValid = isValidEmployee(incompleteEmployee);
        boolean completeIsValid = isValidEmployee(completeEmployee);

        // Assert validation results
        assertFalse(incompleteIsValid, "Employee with missing required fields should be invalid");
        assertTrue(completeIsValid, "Employee with all required fields should be valid");
    }

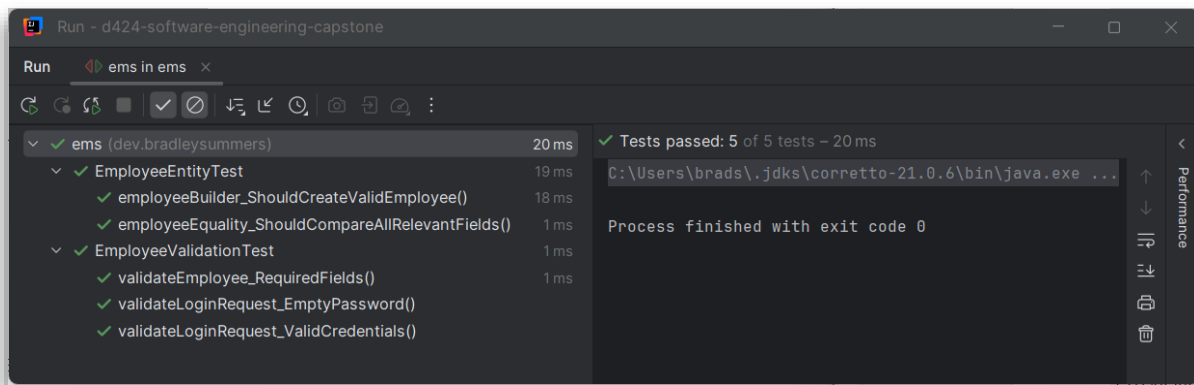
    /**
     * Helper method to validate employee
     */
    private boolean isValidEmployee(Employee employee) {
        return employee != null
            && employee.getEmail() != null && !employee.getEmail().isEmpty()
            && employee.getFirstName() != null && !employee.getFirstName().isEmpty()
            && employee.getLastName() != null && !employee.getLastName().isEmpty();
    }
}
```

PROCEDURES

The testing process began with the creation of two test classes, each targeting specific functionality. Each function was written with a clear goal, followed by one or more assertions to verify expected outcomes. After writing the tests, they were executed within IntelliJ IDEA using the built-in test runner. Results were reviewed immediately. If any test had failed, the logic would have been debugged and the test updated or corrected. However, no failures occurred during testing.

RESULTS

All five tests completed successfully, as indicated by the green checkmarks and the confirmation message in the test runner stating "Process finished with exit code 0". This confirmed that the employee entity behaved as expected and the validation logic correctly enforced input constraints. A screenshot of the test runner results is provided to visually support the test outcomes.



HOSTED WEB APPLICATION

HOSTED WEB APPLICATION LINK

<https://ems.bradleysummers.dev>

GITLAB REPOSITORY & BRANCH HISTORY

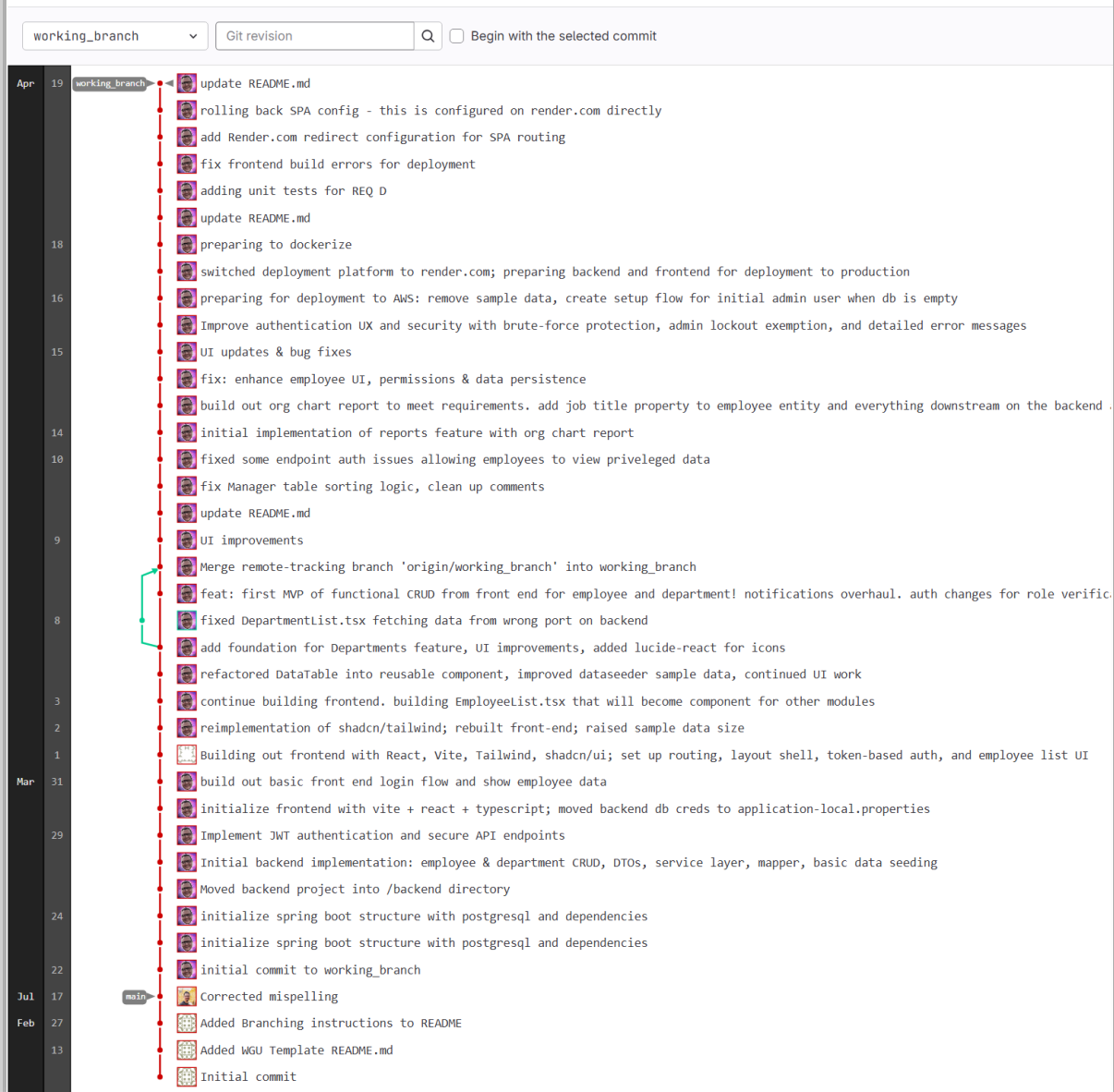
GITLAB REPOSITORY LINK

https://gitlab.com/wgu-gitlab-environment/student-repos/bsumm91/d424-software-engineering-capstone/-/tree/working_branch

GITLAB BRANCH HISTORY

Repository graph

You can move around the graph by using the arrow keys.



USER GUIDE – DEPLOYMENT & SETUP

INTRODUCTION

This guide provides comprehensive instructions for deploying the Employee Management System (EMS) application. I detail the step-by-step process for setting up the database on Neon.tech, containerizing and deploying the backend on Render.com as a web service, and deploying the frontend on Render.com as a static site.

The deployment approach was specifically chosen to be cost-effective for a Minimum Viable Product (MVP), utilizing free tiers of services wherever possible:

1. Database (Neon.tech): Free tier PostgreSQL hosting provides sufficient resources for an MVP.
2. Backend (Render.com): Starter tier selected instead of the free tier because the free tier spins down after periods of inactivity, which would affect the user experience for demonstration purposes.
3. Frontend (Render.com): Free tier static site hosting provides adequate performance.
4. Domain Management (Cloudflare): DNS management and CDN services available on the free tier, though the custom domain registration itself is a paid service.

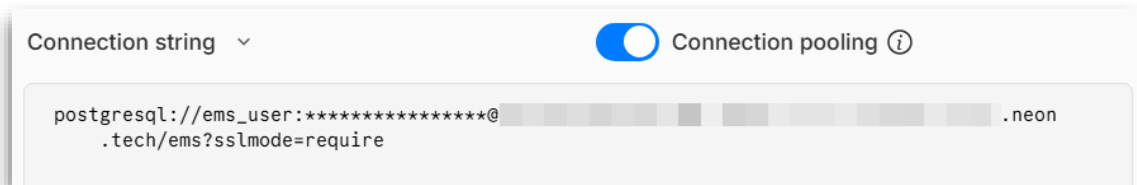
While this guide uses specific providers, the same general approach could be implemented with any number of cloud providers. These specific providers were chosen based on their free tier offerings and ease of use.

DATABASE DEPLOYMENT

1. Create a Neon.tech account and set up a new project with the following settings:
 - a. Region: Choose a region closest to your intended users and where you plan to deploy other services to minimize latency
 - b. Default compute size: 1 > 2 CU (Compute Units)
 - c. Postgres version: 16
2. Create a dedicated database user with the username “ems_user” and a secure password.
3. Execute the following SQL commands to grant the necessary permission to your database user:

```
1 GRANT ALL ON SCHEMA public TO ems_user;  
2 GRANT ALL PRIVILEGES ON ALL TABLES IN SCHEMA public TO ems_user;  
3 GRANT ALL PRIVILEGES ON ALL SEQUENCES IN SCHEMA public TO ems_user;  
4 ALTER DEFAULT PRIVILEGES IN SCHEMA public GRANT ALL PRIVILEGES ON TABLES TO ems_user;
```

4. Note your connection string, which will be in the following format:




BACKEND DEPLOYMENT

DOCKERIZING THE BACKEND APPLICATION

1. The project includes a Dockerfile in the root of the backend directory:

```
Dockerfile

# Build stage
FROM maven:3.9.5-amazoncorretto-21 AS  build

WORKDIR /app

COPY pom.xml .
RUN mvn dependency:go-offline

COPY src ./src
RUN mvn clean package -DskipTests

# Runtime stage
FROM amazoncorretto:21

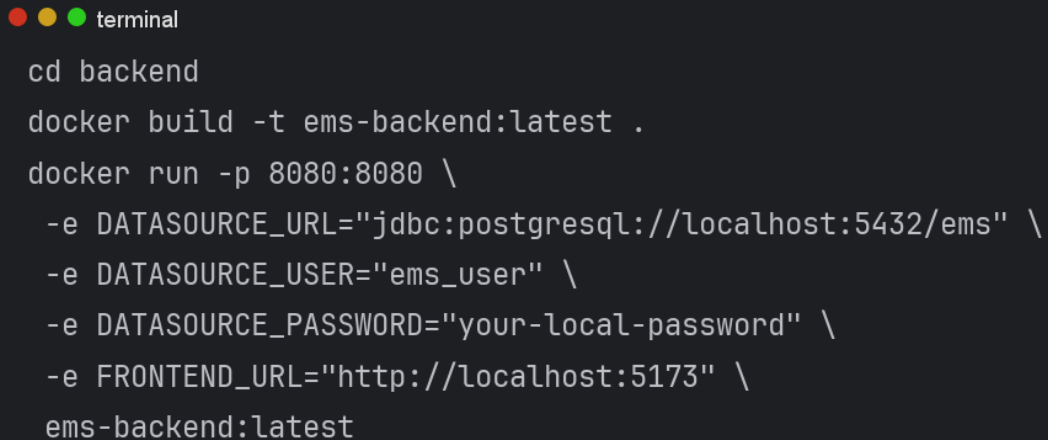
WORKDIR /app

COPY --from=build /app/target/ems-1.0.0-RC1.jar /app/ems.jar

EXPOSE 8080

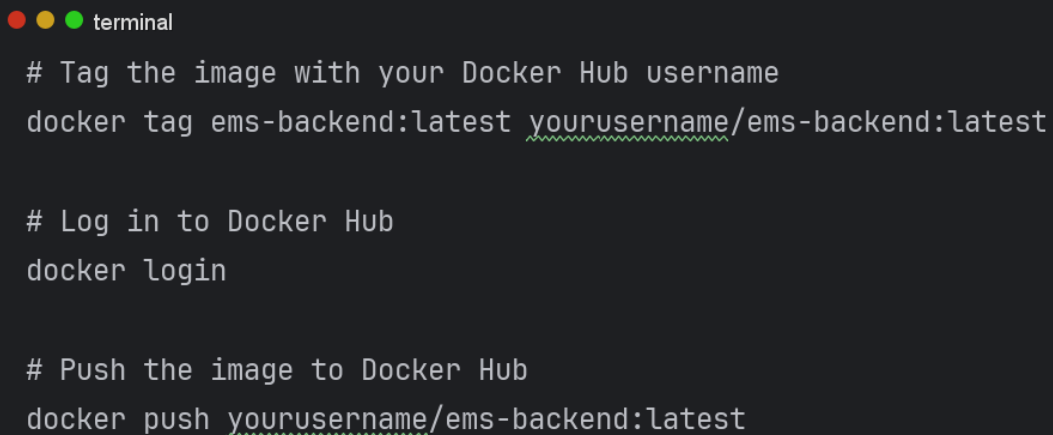
ENTRYPOINT ["java", "-jar", "/app/ems.jar"]
```


2. Build and test the Docker image locally:



```
terminal
cd backend
docker build -t ems-backend:latest .
docker run -p 8080:8080 \
  -e DATASOURCE_URL="jdbc:postgresql://localhost:5432/ems" \
  -e DATASOURCE_USER="ems_user" \
  -e DATASOURCE_PASSWORD="your-local-password" \
  -e FRONTEND_URL="http://localhost:5173" \
  ems-backend:latest
```

3. Once tested successfully, prepare the image for deployment by tagging and pushing it to Docker Hub:



```
terminal
# Tag the image with your Docker Hub username
docker tag ems-backend:latest yourusername/ems-backend:latest

# Log in to Docker Hub
docker login

# Push the image to Docker Hub
docker push yourusername/ems-backend:latest
```

DEPLOYING THE BACKEND TO RENDER.COM

1. Create a new Web Service on Render.com using the "Deploy an existing image from a registry" option with the following settings:
 - a. Name: ems-backend
 - b. Region: Select the same region as your database or one geographically close to it to minimize latency between services
 - c. Instance Type: Starter - Selected because the free tier spins down after periods of inactivity
 - d. Image URL: docker.io/yourusername/ems-backend:latest

2. Configure the following environment variables.

Environment Variables

Set environment-specific config and secrets (such as API keys), then read those values from your code. [Learn more](#).

Key	Value
DATASOURCE_PASSWORD
DATASOURCE_URL	jdbc:postgresql://... ...neon.tech/ems?sslmode=require
DATASOURCE_USER	ems_user
FRONTEND_URL	https://ems.bradleysummers.dev

3. Deploy the service and note the URL once deployment is complete.

WEB SERVICE

Image Starter

Connect

Manual Deploy

<https://...onrender.com>

April 19, 2025 at 2:29 PM Live

Deploy for aa2f6ac

All logs

live

Apr 19, 2:28 PM - 2:30 PM PDT

↑

↕

Apr 19 02:29:58 PM ==> Your service is live


FRONTEND DEPLOYMENT


1. Create a new Static Site on Render.com
 - a. Connect your Git repository and branch
 - b. Root Directory: frontend
 - c. Build Command: npm install; npm run build
 - d. Publish Directory: dist

2. Configure the environment variable with the backend URL you noted previously.

Environment Variables

Set environment-specific config and secrets (such as API keys), then read those values from your code. [Learn more](#).

 Edit

Key	Value
VITE_API_URL	https://[redacted].onrender.com 

3. Add a rewrite rule for proper SPA routing.

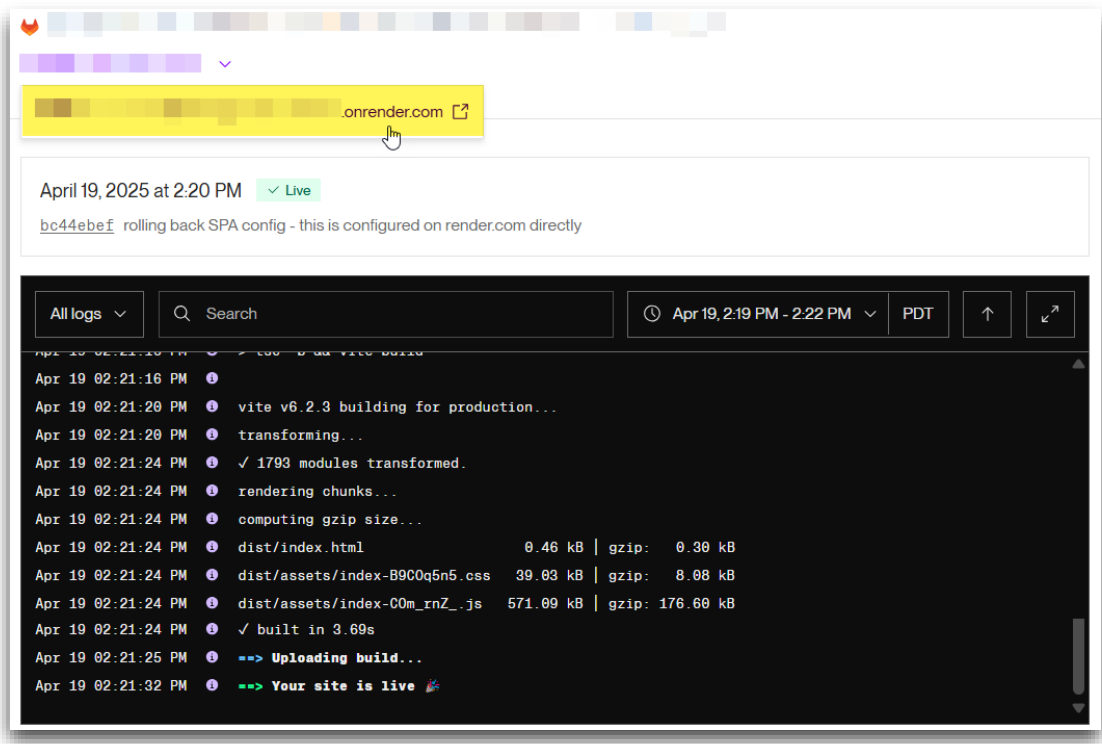
Redirect and Rewrite Rules

Add [Redirect](#) or [Rewrite Rules](#) to modify requests to your site. Use URL parameters to capture path segments and wildcards to redirect everything under a given path.

	Source	Destination	Action	
↑	↓	/*	/index.html	Rewrite
			+ Add Rule	Save Changes

4. Enable Auto-Deploy to automatically deploy your site whenever changes are pushed to your repository.

- 5. Deploy the site and note the URL once deployment is complete.



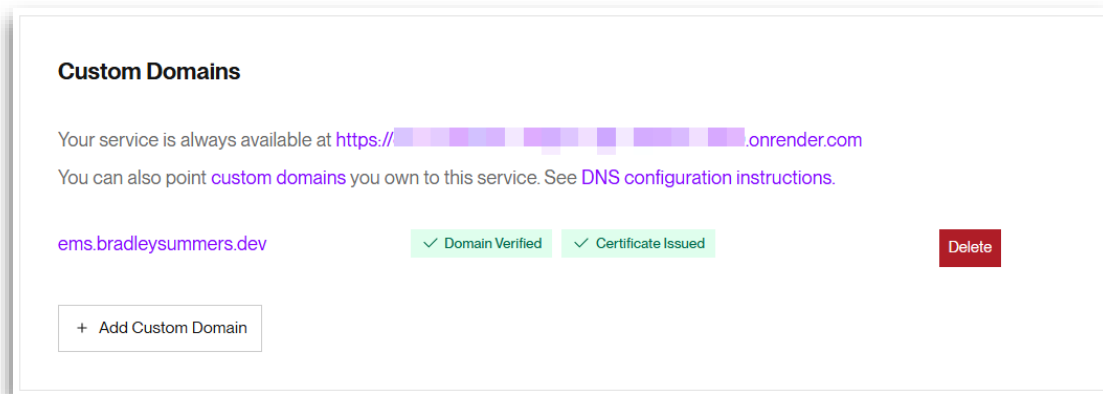
CUSTOM DOMAIN CONFIGURATION

- 1. Select your frontend service in the Render.com dashboard
- 2. **Settings > Custom Domains > + Add Custom Domain** and provide your custom domain name
- 3. Configure DNS with your provider (in our case, Cloudflare).
 - a. Create a new CNAME record that points to the frontend Static Site onrender.com subdomain URL
 - b. Your completed configuration should resemble this:

Type ▲	Name	Content	Proxy status	TTL
① CNAME	example.com	mysite-ne1j.onrender.com	☁️ DNS only	Auto
CNAME	www	mysite-ne1j.onrender.com	☁️ DNS only	Auto

- 4. Return to your service's Custom Domains settings in the Render Dashboard. Click the Verify button next to your custom domain.
 - a. If verification fails, your DNS settings might not have propagated yet. Wait a few minutes and try again.

5. If verification succeeds, Render issues a TLS certificate for your domain and updates the verification status:

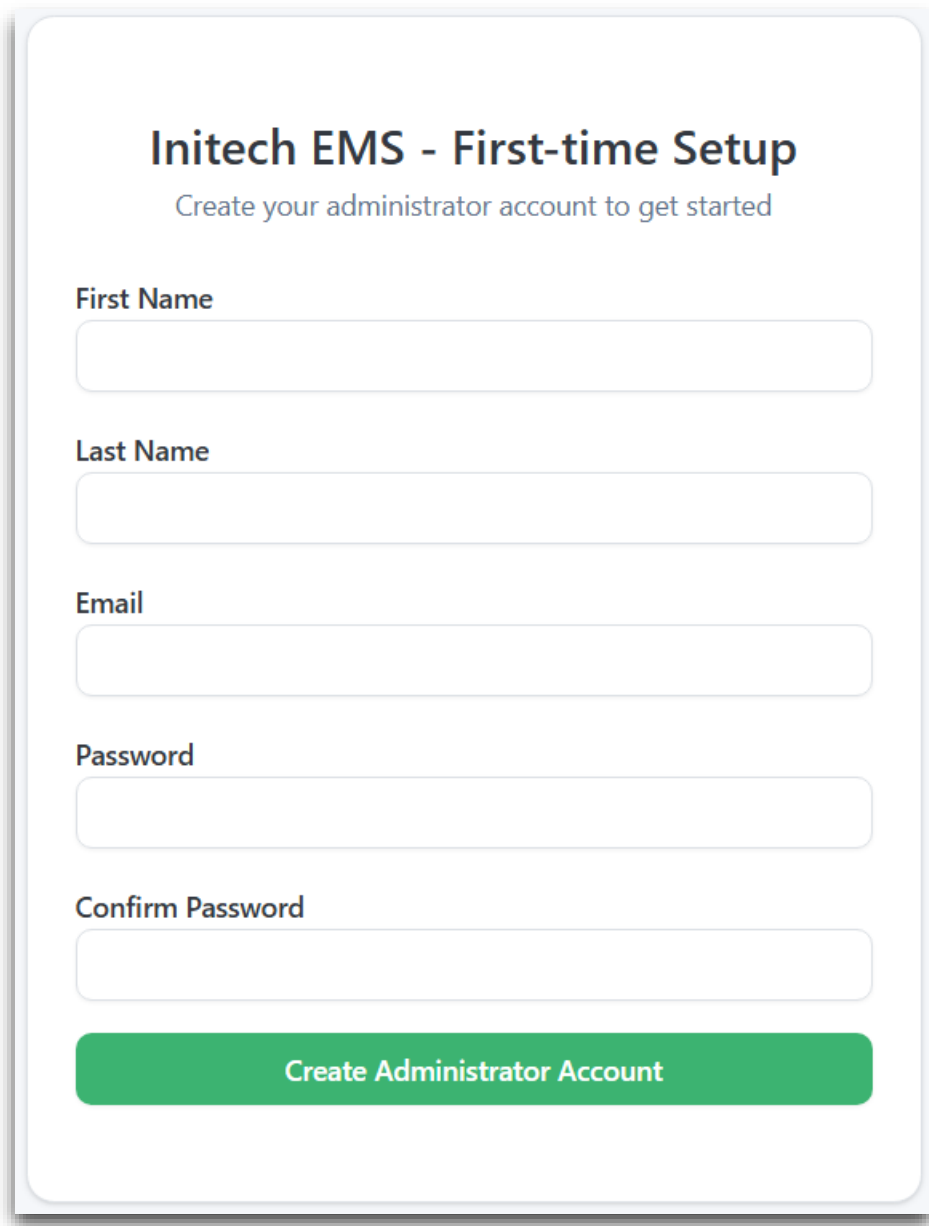


SETUP

Once verification is complete, your application will be accessible at your custom domain. When you first access the application, it automatically detects that no administrator account exists and presents you with the setup screen seen below.

The setup process requires you to create an administrator account by providing your first name, last name, email address, and password. After completing the form and clicking "Create Administrator Account" the system creates your account with the ADMIN role and automatically logs you in to the application dashboard.

This setup process only runs once when the application is first deployed. After the initial administrator account is created, subsequent visits to the application will display the standard login page.



The image shows a web form titled "Initech EMS - First-time Setup" with the subtitle "Create your administrator account to get started". The form contains five input fields: "First Name", "Last Name", "Email", "Password", and "Confirm Password". Each field is a simple white rectangle with a light gray border. Below the "Confirm Password" field is a green button with the text "Create Administrator Account" in white.

Initech EMS - First-time Setup
Create your administrator account to get started

First Name

Last Name

Email

Password

Confirm Password

Create Administrator Account

USER GUIDE – APPLICATION USAGE

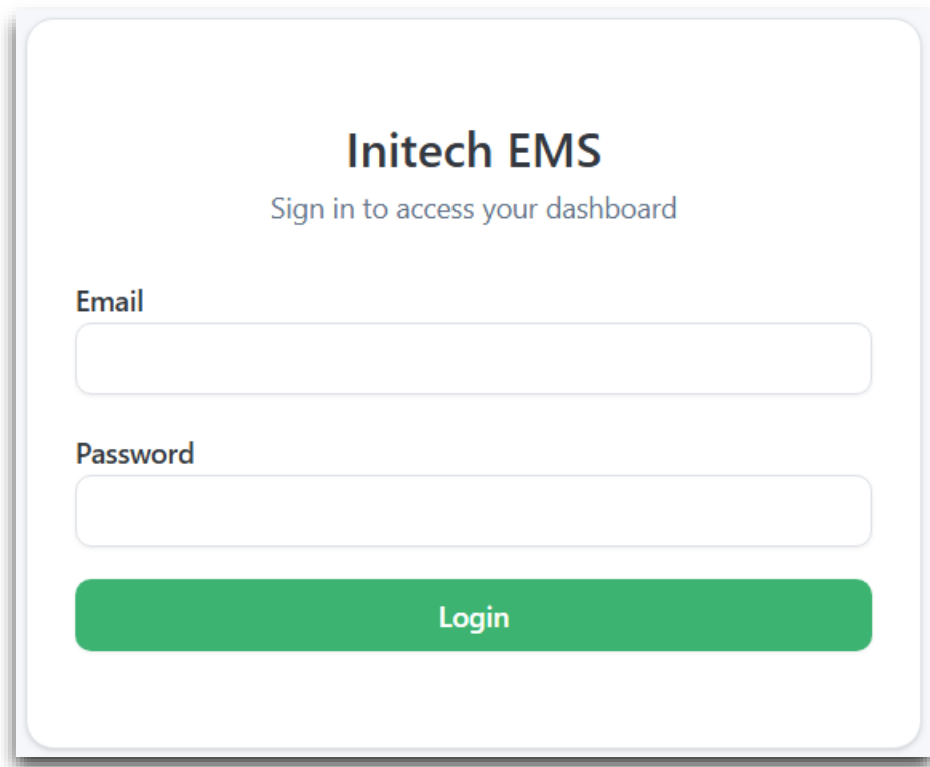
This section provides instructions for using the Employee Management System (EMS), organized by feature area with guidance for both administrator and standard employee users.

LOGIN

URL: <https://ems.bradleysummers.dev/login>

1. Navigate to <https://ems.bradleysummers.dev/login> in your web browser.
2. Enter your email address and password.
3. Click the "Login" button.
4. Upon successful login, you will be redirected to the Employees page.

Note: If you forget your password, contact an administrator. Only administrators can reset passwords.

A login form for Initech EMS. The form is white with rounded corners and a subtle shadow. At the top, it says "Initech EMS" in a bold, dark blue font, followed by "Sign in to access your dashboard" in a smaller, grey font. Below this are two input fields: "Email" and "Password", both with light grey borders. At the bottom is a green "Login" button with white text.

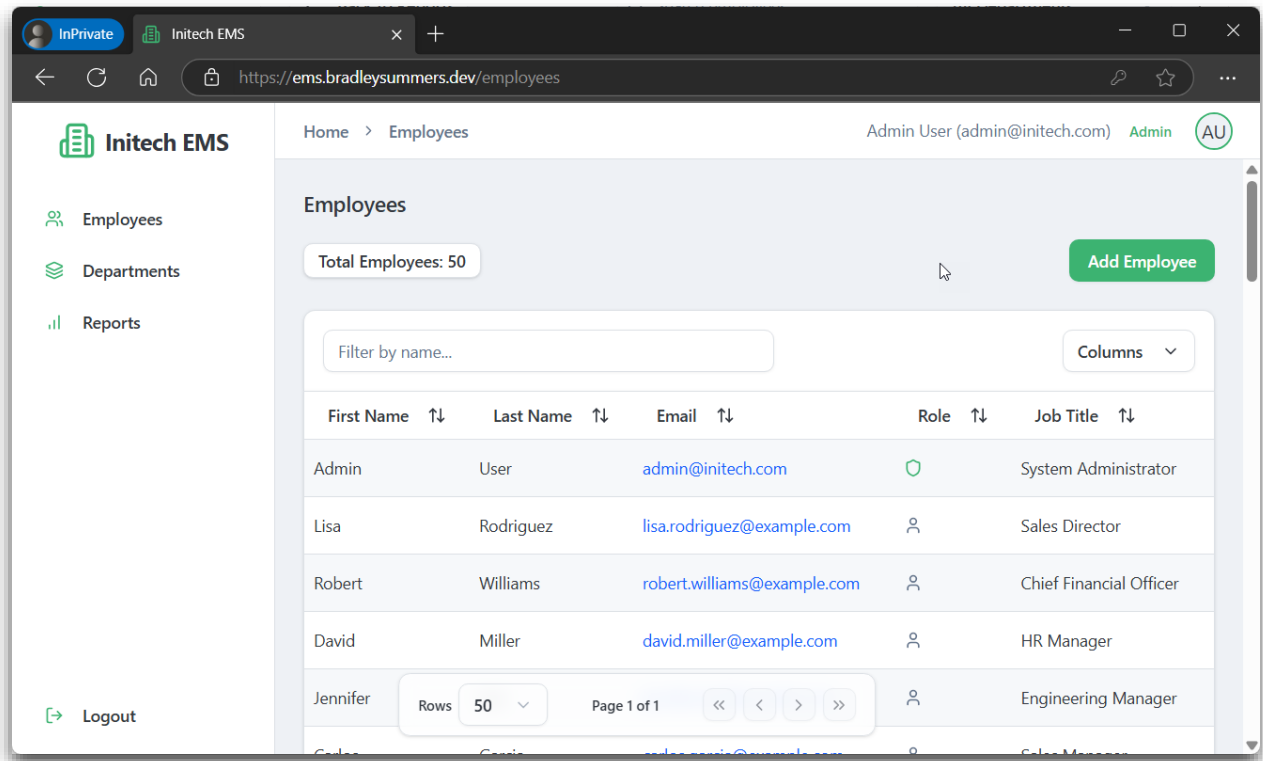
EMPLOYEES

VIEWING EMPLOYEE LIST

URL: <https://ems.bradleysummers.dev/employees>

1. After login, you are automatically directed to the Employees page.
2. View employees in the table with columns for name, email, role, job title, department, and manager.
3. Use the search box to filter by first name.
4. Click column headers to sort the table.

Note: Administrators see all employees. Standard employees only see themselves.



ADDING A NEW EMPLOYEE

URL: <https://ems.bradleysummers.dev/employees/new>

1. Go to the Employees page.
2. Click "Add Employee" in the top-right corner (administrators only).
3. Complete the form with required information:
 - a. First Name
 - b. Last Name
 - c. Email
 - d. Password
 - e. Role
 - f. Job Title
 - g. Department
 - h. Manager
4. Click "Create" to create the employee.
5. The system validates the information and shows any errors.
6. Upon success, you return to the employee list with a confirmation message.

Note: Only administrators can add employees.

The screenshot displays a web browser window with the Initech EMS application. The browser's address bar shows the URL `https://ems.bradleysummers.dev/employees/new`. The application's header includes the Initech EMS logo, a breadcrumb trail (Home > Employees > New), and the user's identity (Admin User (admin@initech.com) Admin AU). A left sidebar contains navigation links for Employees, Departments, and Reports. The main content area is titled 'New Employee' and includes a 'Back to Employees' link. The form itself contains several input fields and dropdown menus: First Name, Last Name, Email, Password, Job Title, Department (with a 'Select a department' dropdown), Manager (with a 'No Manager' dropdown), Role (with an 'Employee' dropdown), and Status (with an 'Active' dropdown). At the bottom of the form are 'Cancel' and 'Create' buttons. A 'Logout' link is located in the bottom left corner of the application interface.

Initech EMS

Home > Employees > New Admin User (admin@initech.com) Admin AU

New Employee

[← Back to Employees](#)

New Employee
Create a new employee

First Name Last Name

Email

Password

Job Title

Department
Select a department

Manager
No Manager

Role
Employee

Status
Active

[Cancel](#) [Create](#)

[Logout](#)

EDITING AN EMPLOYEE

URL: <https://ems.bradleysummers.dev/employees/{id}>

1. Go to the Employees page.
2. Find the employee using the search function if needed.
3. Click their email or "View details" from the actions menu.
4. Edit the fields you have permission to change.
5. Click "Update" to update the information.
6. The system validates changes and shows any errors.
7. Upon success, a confirmation message appears.

Note: Administrators can edit all records. Employees can only edit their password.

The screenshot shows a web browser window with the URL `https://ems.bradleysummers.dev/employees/56`. The application is titled "Initech EMS" and has a sidebar with navigation links: "Employees", "Departments", and "Reports". The main content area displays the profile for "Lisa Rodriguez" with a "Back to Employees" button. The profile form includes fields for "First Name" (Lisa), "Last Name" (Rodriguez), "Email" (lisa.rodriguez@example.com), "New Password" (Leave blank to keep current password), "Job Title" (Sales Director), "Department" (Sales), "Manager" (John Smith), "Role" (Employee), and "Status" (Active). There are "Cancel" and "Update" buttons at the bottom of the form. A "Logout" link is in the bottom left corner.

Initech EMS

Home > Employees > Lisa Rodriguez Admin User (admin@initech.com) Admin AU

Lisa Rodriguez

[← Back to Employees](#)

Lisa Rodriguez
lisa.rodriguez@example.com

First Name
Lisa

Last Name
Rodriguez

Email
lisa.rodriguez@example.com

New Password
Leave blank to keep current password

Job Title
Sales Director

Department
Sales

Manager
John Smith

Role
Employee

Status
Active

[Cancel](#) [Update](#)

[Logout](#)

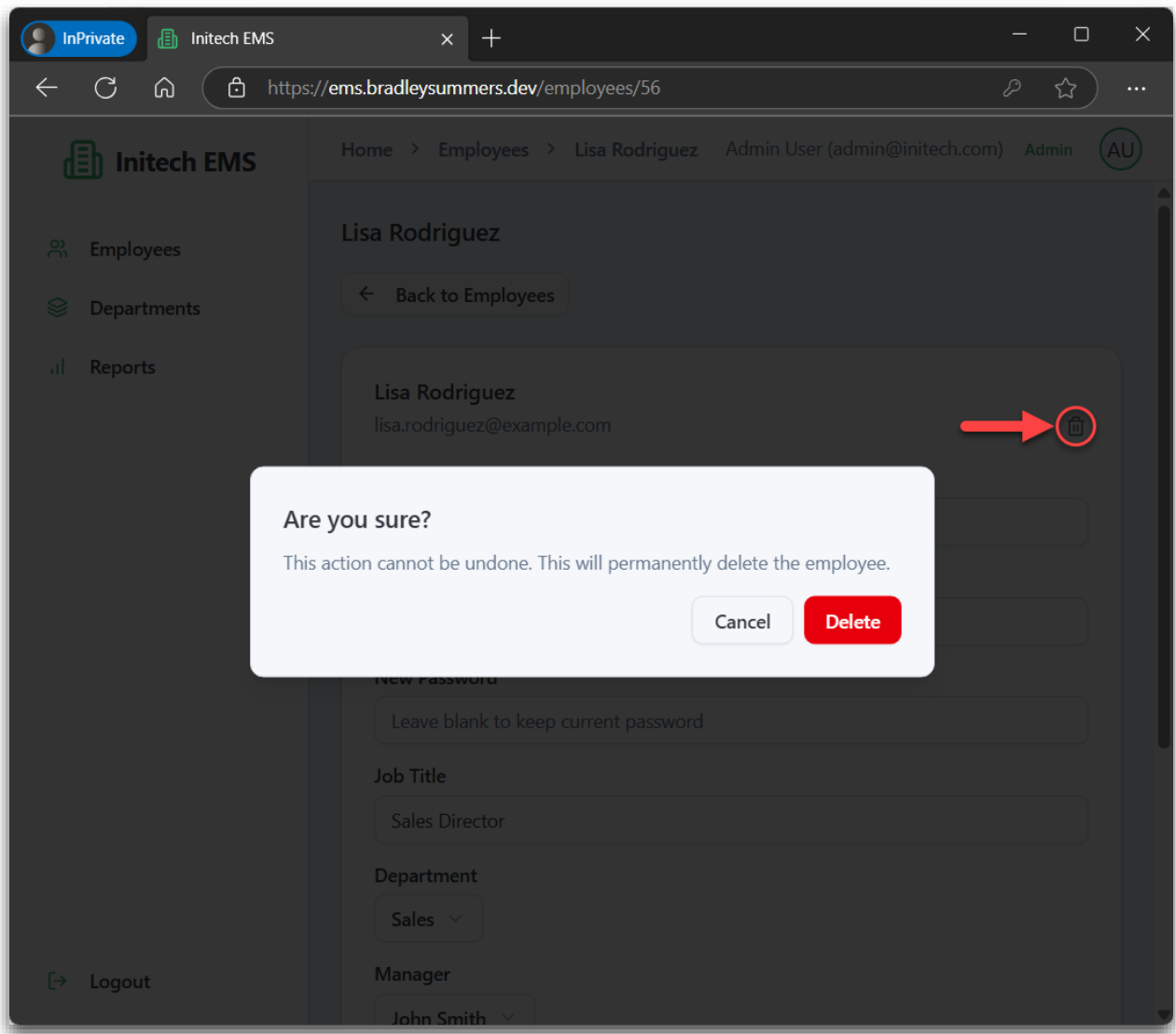
DELETING AN EMPLOYEE

URL: <https://ems.bradleysummers.dev/employees/{id}>

1. Go to the Employees page.

2. Find the employee to delete.
3. Click their email or "View details" from the actions menu.
4. Click the "Delete" button (trash icon) in the top-right corner.
5. Confirm deletion in the dialog.
6. Upon success, you return to the employee list with a confirmation message.

Note: Only administrators can delete employees and cannot delete their own account.



DEPARTMENTS

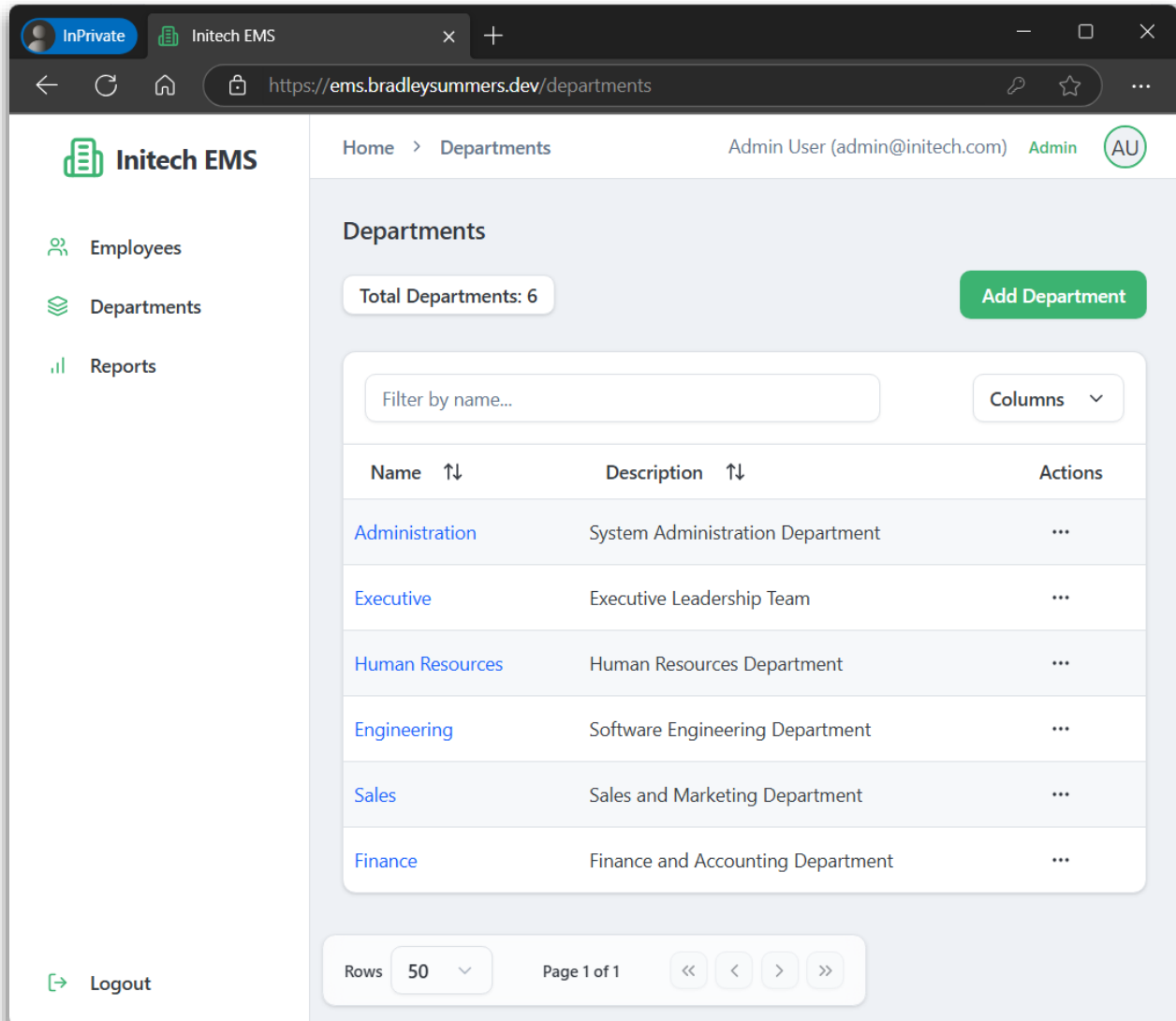
VIEWING DEPARTMENT LIST

URL: <https://ems.bradleysummers.dev/departments>

1. Click "Departments" in the left sidebar.
2. View departments in the table with name and description columns.

3. Use the search box to filter by name.
4. Click column headers to sort the table.

Note: All users can view the department list.



ADDING A NEW DEPARTMENT

URL: <https://ems.bradleysummers.dev/departments/new>

1. Go to the Departments page.
2. Click "Add Department" in the top-right corner (administrators only).
3. Complete the form with:
 - a. Name (must be unique)
 - b. Description
4. Click "Save" to create the department.
5. The system validates the information and shows any errors.
6. Upon success, you return to the department list with a confirmation message.

Note: Only administrators can add departments.

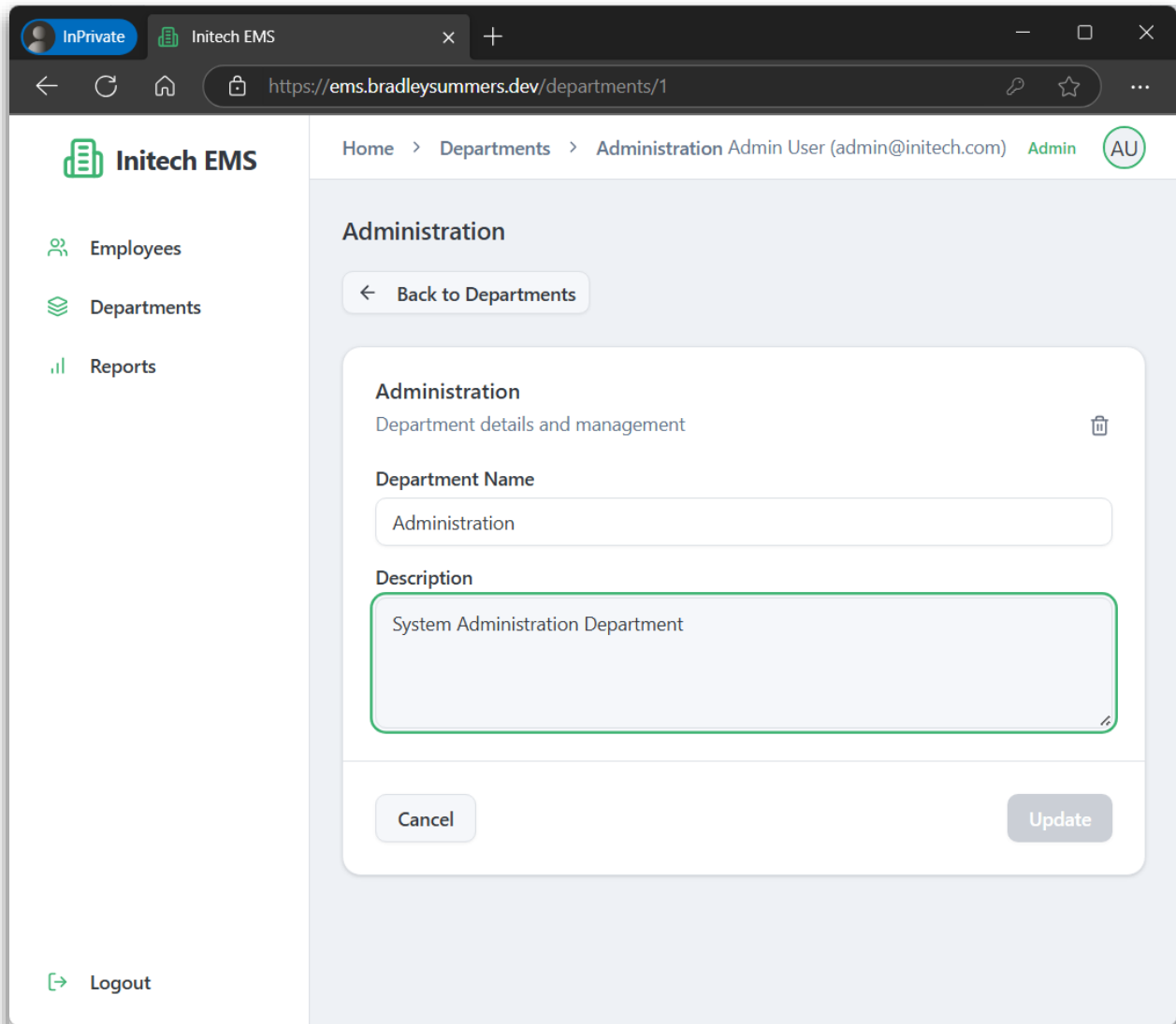
The screenshot shows a web browser window with the URL <https://ems.bradleysummers.dev/departments/new>. The page title is "New Department". The left sidebar contains links for "Employees", "Departments", and "Reports". The main content area has a "Back to Departments" button and a "New Department" form. The form includes a "Department Name" text input field and a "Description" text area. At the bottom of the form are "Cancel" and "Create" buttons. The user is logged in as "Admin User (admin@initech.com)" with the role "Admin".

EDITING A DEPARTMENT

URL: <https://ems.bradleysummers.dev/departments/{id}>

1. Go to the Departments page.
2. Find the department to edit.
3. Click its name or "View details" from the actions menu.
4. Edit the name and/or description.
5. Click "Save" to update the information.
6. The system validates changes and shows any errors.
7. Upon success, a confirmation message appears.

Note: Only administrators can edit departments.

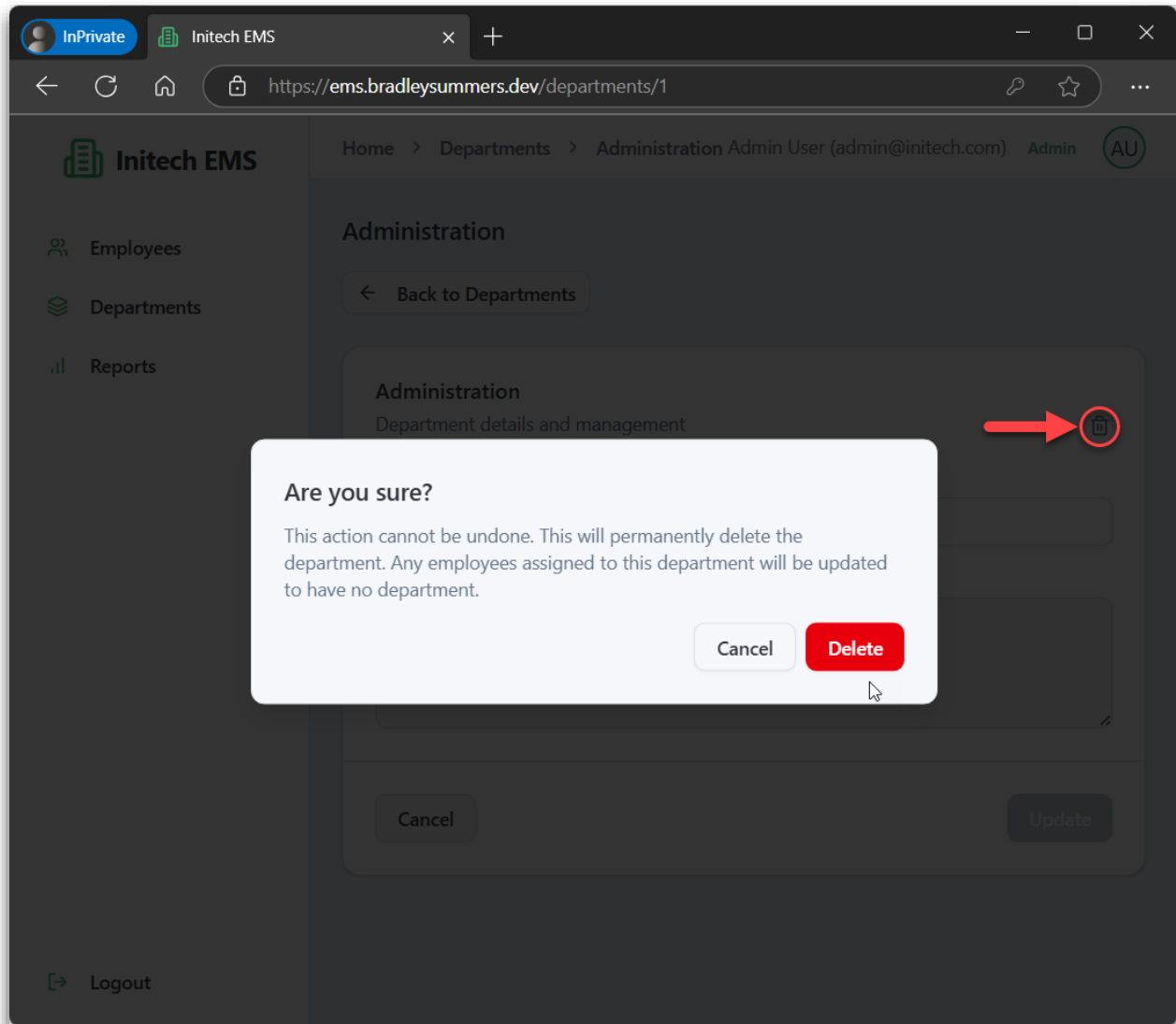


DELETING A DEPARTMENT

URL: <https://ems.bradleysummers.dev/departments/{id}>

1. Go to the Departments page.
2. Find the department to delete.
3. Click its name or "View details" from the actions menu.
4. Click the "Delete" button (trash icon) in the top-right corner.
5. Confirm deletion in the dialog.
6. Upon success, you return to the department list with a confirmation message.

Note: Only administrators can delete departments.



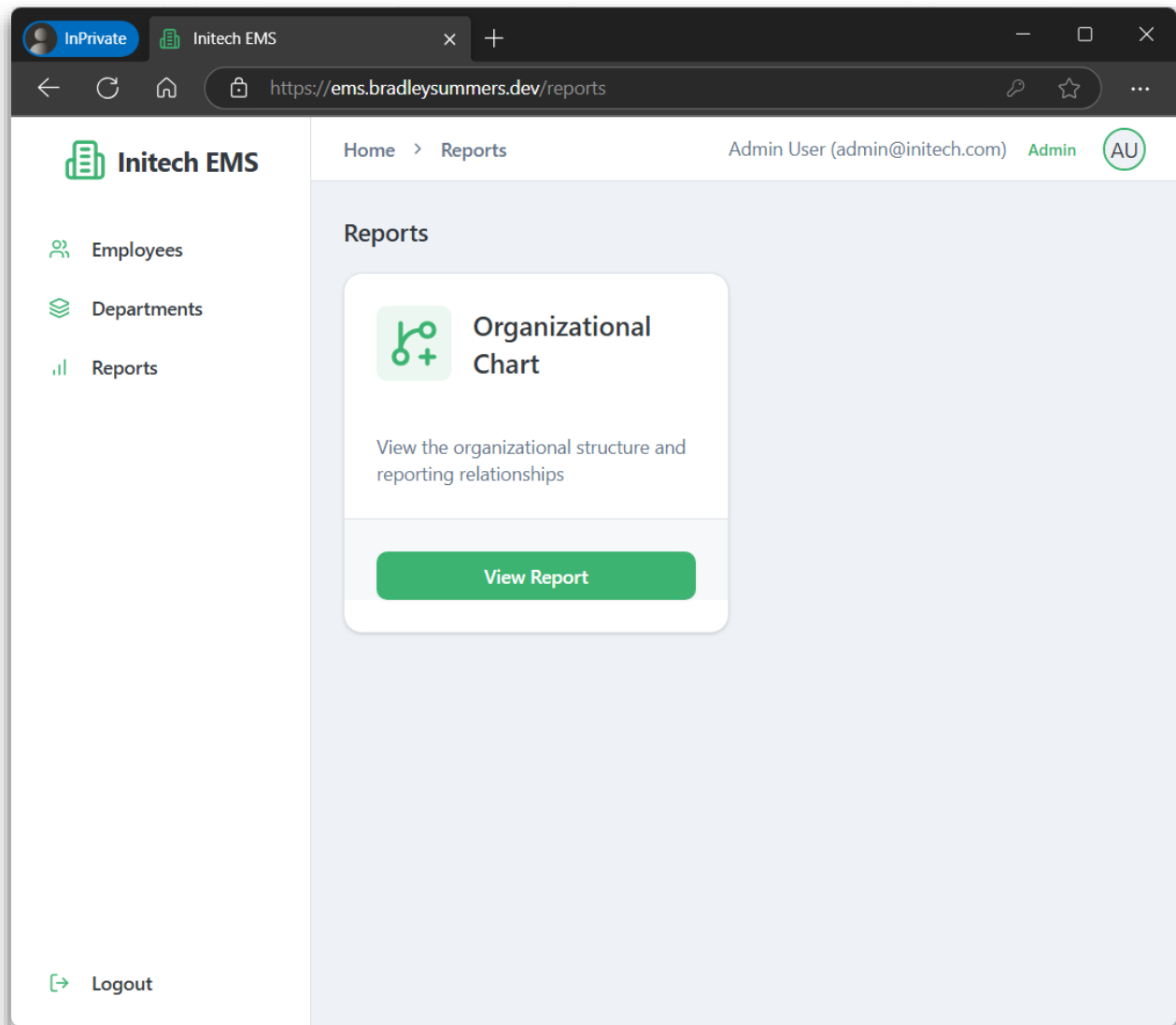
REPORTS

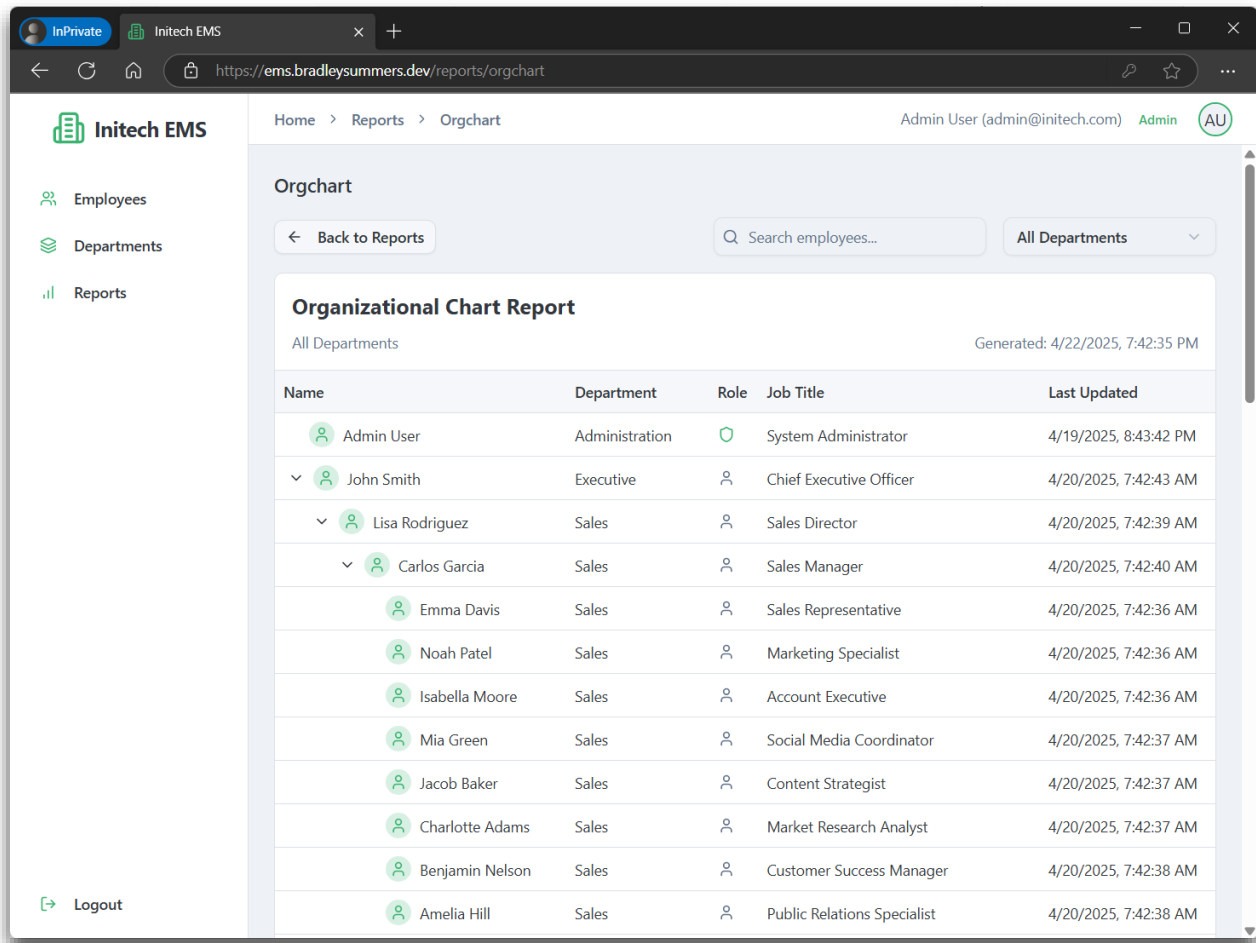
VIEWING THE ORGANIZATIONAL CHART

URL: <https://ems.bradleysummers.dev/reports/orgchart>

1. Click "Reports" in the left sidebar.
2. Select "Organizational Chart" from the dashboard.
3. View the hierarchical structure of employees.
4. Use the department filter to view specific departments or "All Departments."
5. Click chevron icons to expand/collapse direct reports.

Note: Administrators see the entire chart. Standard employees only see themselves and their direct reports.





APPLICATION NAVIGATION AND USER PROFILE

USING THE NAVIGATION MENU

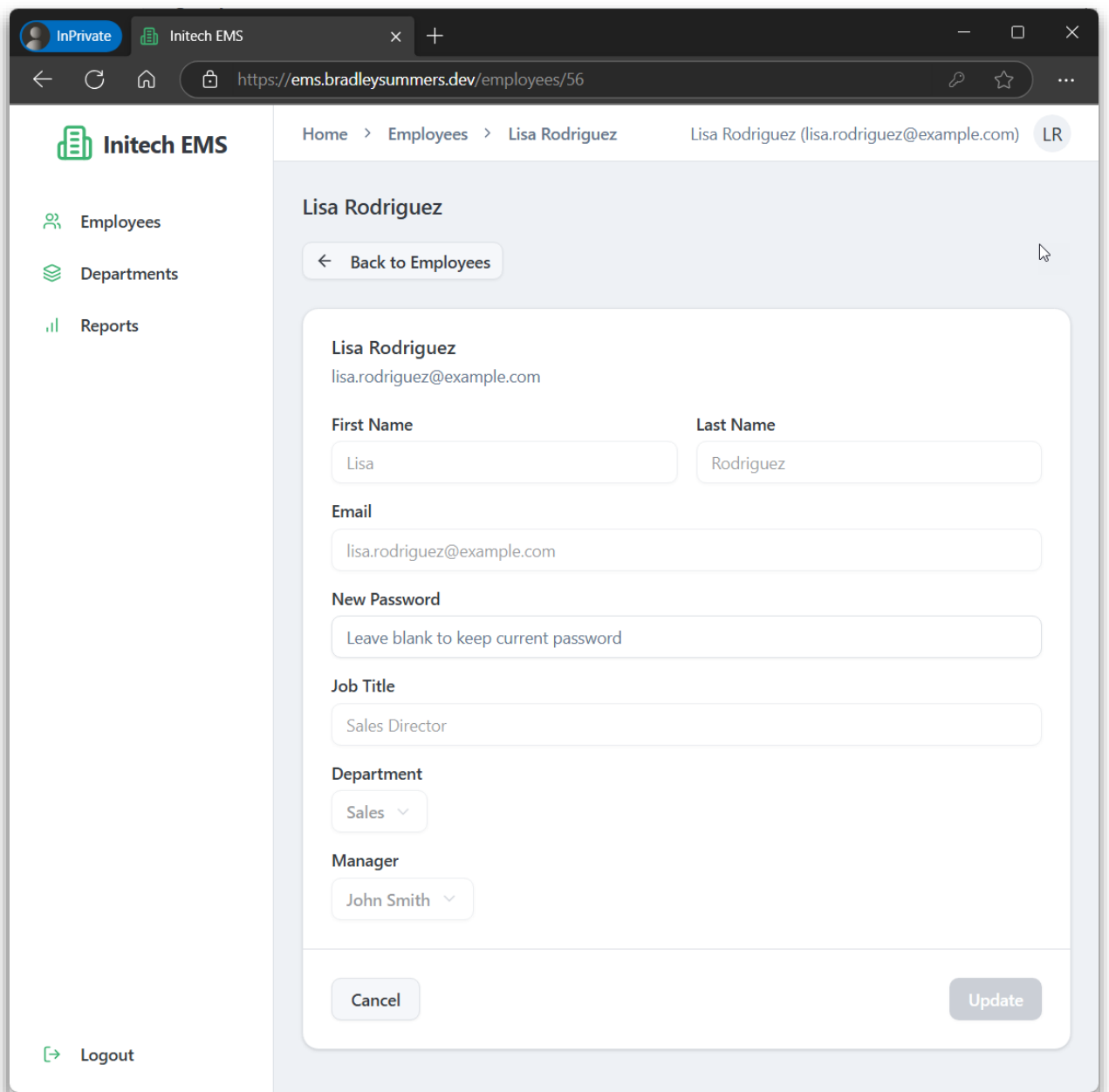
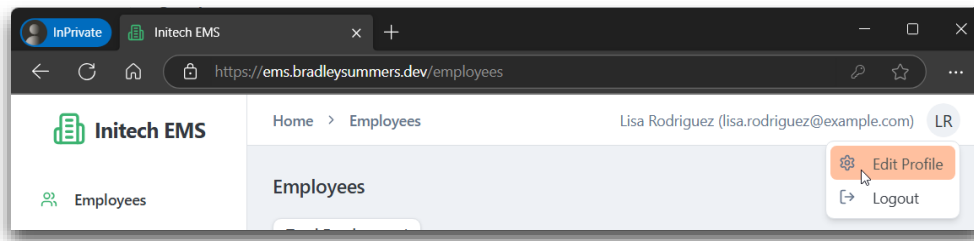
Main URLs:

- Employees: <https://ems.bradleysummers.dev/employees>
 - Departments: <https://ems.bradleysummers.dev/departments>
 - Reports: <https://ems.bradleysummers.dev/reports>
1. Use the left sidebar menu to navigate between sections.
 2. The current section is highlighted for reference.

VIEWING YOUR USER PROFILE

URL: <https://ems.bradleysummers.dev/employees/{id}>

1. Click your avatar (initials) in the top-right corner.
2. Select "View Profile" from the dropdown menu.
3. View and edit your information based on your permissions.



LOGGING OUT

1. Click either:
 - a. "Logout" at the bottom of the sidebar, or
 - b. Your avatar and select "Logout" from the dropdown

2. The system returns you to the login page.