SE/COM S 3190 – Construction of User Interfaces

Final Project Technical Documentation

Fall 2025

ISU Campus Explorer

Team Members

Mekhi San
Sanm20@iastate.edu

Ash Bhuiyan
mbhuiyan@iastate.edu

Iowa State University
Department of Computer Science

Contents

1 Introduction

1.1 Project Overview

ISU Campus Explorer is a comprehensive single-page application (SPA) designed to help students, faculty, and visitors navigate the Iowa State University campus. The problem it solves is the complexity of finding specific building details (like floor plans, hours, and internal departments) and planning efficient routes between classes. Our solution aggregates this data into an interactive visual map and directory, allowing users to create custom "Tours" to plan their day.

1.2 Target Users and Use Cases

Primary Users (Students/Visitors): Their main workflow is to search for buildings, view detailed floor plans, and create saved "Tours" (lists of stops) to visualize their route on the map.

Secondary Users (Administrators): Their workflow involves managing the building database, updating floor plan images, and repositioning buildings on the map using a drag-and-drop editor.

1.3 Core Features

Interactive Map: A visual map of campus where users can click buildings to see details or view active tour routes with connecting lines.

Tour Planner: A multi-step tool for users to create, order, and save custom lists of buildings to visit.

Building Directory (CMS): A rich detail view for every building showing hours, departments, and interactive floor plans.

Admin Dashboard: A secured area for admins to Create/Read/Update/Delete (CRUD) buildings and upload media files.

1.4 Originality vs Inspiration

This project is **Option 1 (Extended Midterm)**. It rebuilds our static HTML/CSS Midterm project into a dynamic React + Node.js application. While inspired by the concept of Google Maps, our version is unique because it offers "inside-the-building" data (floor plans) and custom tour creation specifically for ISU, which general map apps lack.

2 System Overview and Project Description

2.1 Major Functional Modules

Authentication Module: Handles User/Admin login and signup using persistent MongoDB storage.
Map Module: Renders the SVG map, handles drag-and-drop logic for admins, and draws polyline routes for tours.
Building CMS: Manages building data, including a file upload system for hero images and floor plans.
Tour Management: Allows users to create, edit, and delete their own saved tours.

2.2 UI and Navigation Flow

User Flow:

1. User lands on **Home** and navigates to **Campus Map** to explore.
2. User logs in via **Login** page.
3. User goes to **My Tours** and clicks **"Create Tour"**.
4. User selects buildings, orders them, and hits **Save**.
5. User is redirected to **Confirmation**, then back to **My Tours**.
6. User clicks **"View on Map"** to see their route visualized.

Admin Flow:

1. Admin logs in and is redirected to **Admin Dashboard**.
2. Admin clicks **"Manage Buildings"** to edit a building.
3. Admin uploads a new photo via the **"Edit Media"** modal.

2.3 CRUD Entities

We implemented full CRUD (Create, Read, Update, Delete) for two major entities:

1. **Buildings (Admin Only):**
    - **Fields:** Name, Code, Type, Description, Hours, Floors, Images, Coordinates.
    - **Actions:** Admins can add new buildings, update details (including positions), and delete old records.
2. **Tours (User Only):**
    - **Fields:** Name, Description, Ordered List of Building IDs, Creator ID.
    - **Actions:** Users can create custom tours, view their list, and delete tours they no longer need.

3 File and Folder Architecture

3.1 Folder Structure Overview

We used a standard MERN stack structure separating frontend (client) and backend (server). This ensures separation of concerns: the frontend handles the UI and state, while the backend manages data logic and database connections.

3.2 Backend Folder Structure

3.2.1 Backend Folder Tree:

Insert a picture of your backend folder structure here. Include all files and subdirectories relevant to your server, routes, controllers, configuration, and environment.

- **3.2.2 Backend Folder Description:**
  config/: Database connection logic (db.js).
- models/: Mongoose schemas (Building.js, Tour.js, User.js).
- routes/: Express API endpoints (auth.js, buildings.js, tours.js, upload.js).
- scripts/: Utility scripts for seeding data (seedBuildings.js).
- server.js: Entry point.

Description: We separated routes from models to keep the code clean. The upload.js route handles file storage using the Multer library, keeping heavy binary logic separate from standard CRUD data.

3.3 Frontend Folder Structure

3.3.1 Frontend Folder Tree:

Insert a picture representing your actual frontend directory. Mirror the style used in the backend diagram but modify according to your frontend setup.

3.3.2 Frontend Folder Description:

src/components/: Reusable UI parts (NavBar.jsx).

src/pages/: Main views (CampusMap.jsx, TourPlanner.jsx, AdminBuildings.jsx).

public/assets/: Static images and user-uploaded content.

App.jsx: Main routing definition.

Description: We organized the frontend by "Pages" to match the React Router structure. Each major route (like /campus or /admin) has a corresponding file in pages/.

4 Code Explanation and Logic Flow

This section explains how the application behaves internally, how data flows between components, and how requests are processed from the UI to the database.

4.1 Frontend–Backend Interaction

The frontend uses fetch calls to communicate with the backend. Data is sent as JSON.

- **Request:** The frontend sends a POST request to /api/tours with a JSON body containing the tour name and building IDs.
- **Processing:** The backend validates the IDs, creates a MongoDB document, and saves it.
- **Response:** The backend returns the saved object with a 201 Created status.

- **Update:** The frontend receives the response and redirects the user to the Confirmation page.

4.2 React Component Example

Component: BuildingDetail.jsx This component handles the logic for displaying interactive floor plans. It uses useState to switch between different floor images based on user clicks.

```
// frontend/src/pages/BuildingDetail.jsx
export default function BuildingDetail() {
  const [selectedFloor, setSelectedFloor] = useState('level1');

  // Switch floor based on button click
  const handleFloorSelect = (floorKey) => {
    setSelectedFloor(floorKey);
  };

  return (
    <div>
      {/* Floor Selection Buttons */}
      <div className="floor-buttons">
        <button onClick={() => handleFloorSelect('level1')}>Level 1</button>
        <button onClick={() => handleFloorSelect('level2')}>Level 2</button>
      </div>

      {/* Conditional Rendering of Floor Plan */}
      <div className="floor-display">
        <img src={floorPlans[selectedFloor].image} alt="Floor Plan" />
      </div>
    </div>
  );
}
```

4.3 API Route Examples

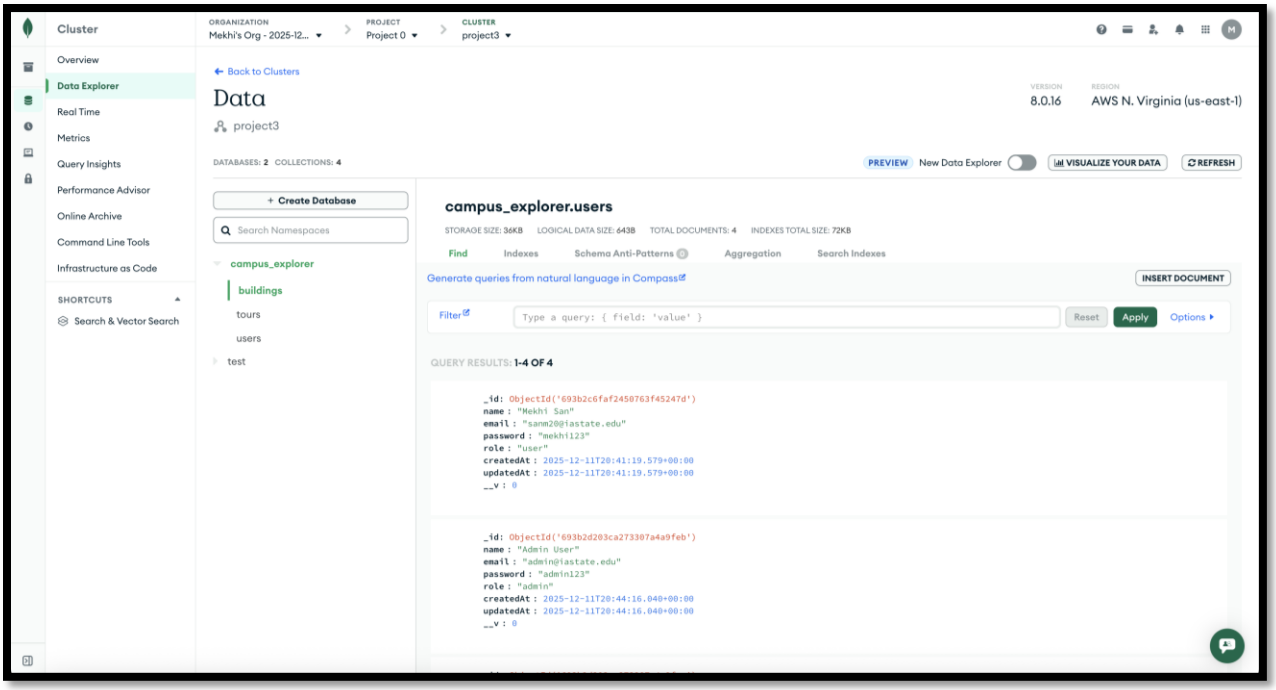
Route: POST /api/tours This route handles the creation of a new Tour in the database.

```
// backend/routes/tours.js
router.post('/', async (req, res) => {
  try {
    const { name, buildingIds, ownerEmail } = req.body;

    // Create new Tour document
    const tour = new Tour({
      name,
      buildingIds, // Array of building IDs
      owner: user._id
    });

    const savedTour = await tour.save();
    res.status(201).json({ success: true, tour: savedTour });
  } catch (error) {
    res.status(500).json({ success: false, message: error.message });
  }
});
```

4.4 Database Interaction

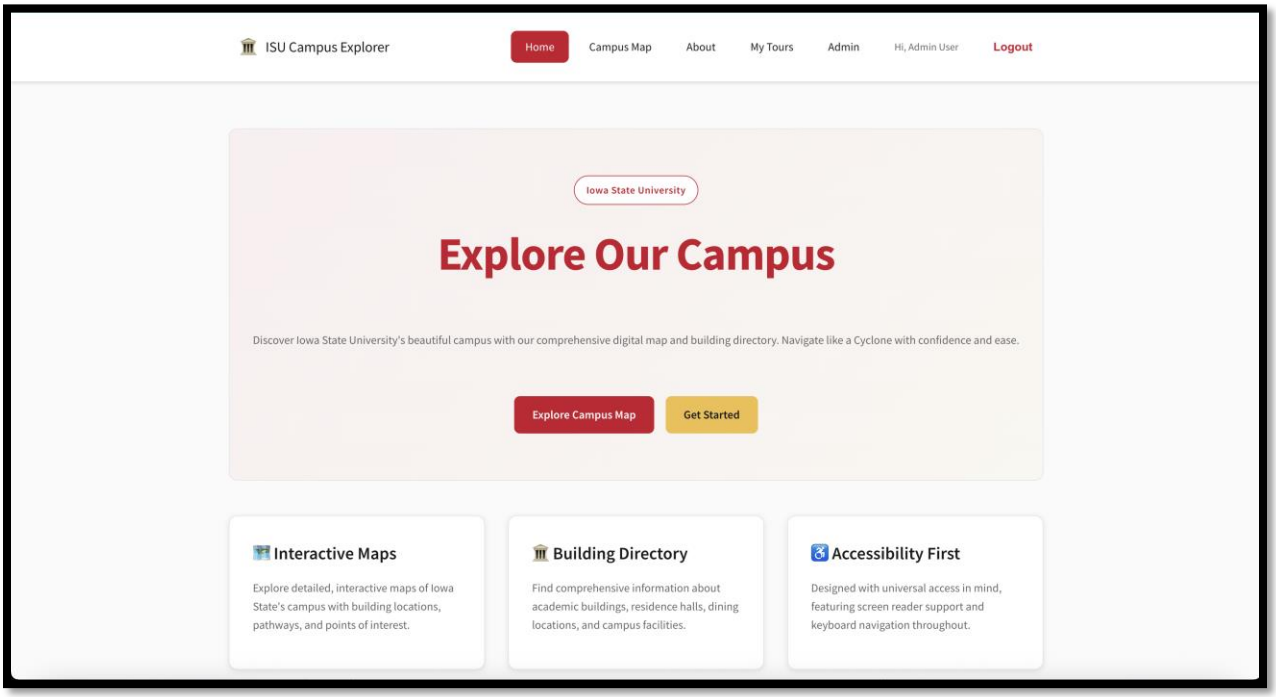
We use **Mongoose** to model our data. The Tour schema references the User schema, establishing a relationship between data.

```
const tourSchema = new mongoose.Schema({
  name: { type: String, required: true },
  // Reference to the User who owns this tour
  owner: { type: mongoose.Schema.Types.ObjectId, ref: 'User' },
  // List of buildings in the tour
  buildingIds: [String]
});
```



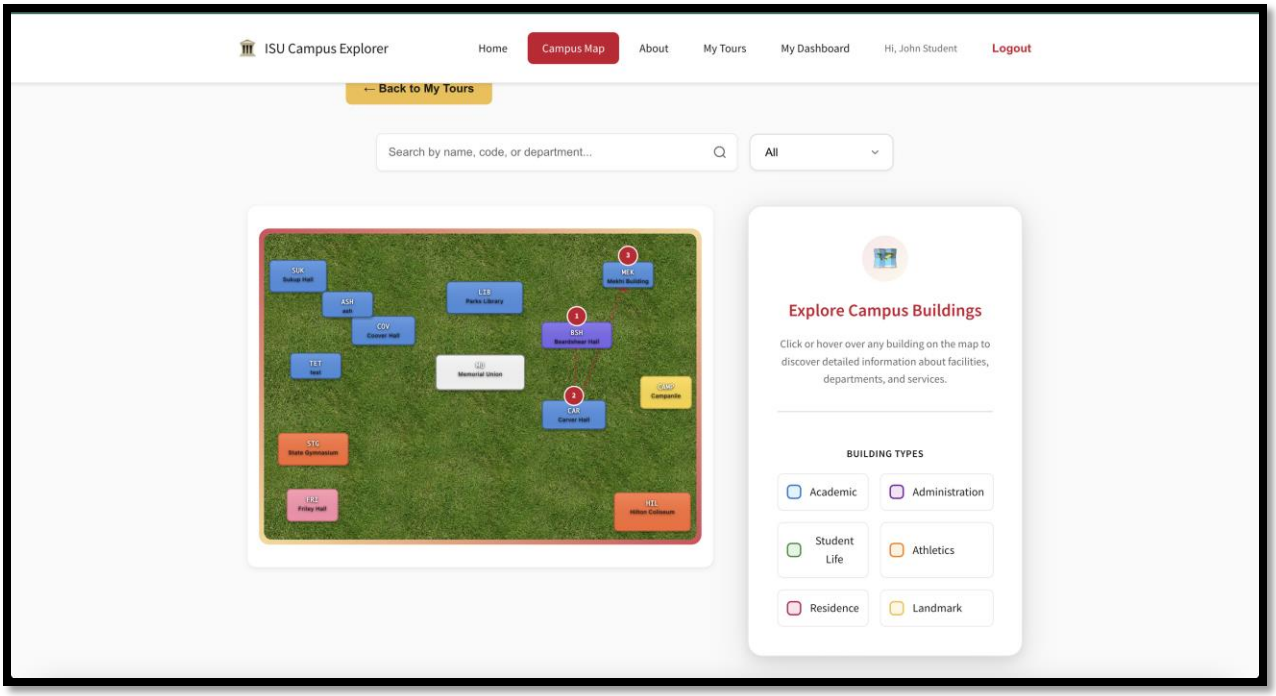
5 Screenshots and Web Views

Home Page:



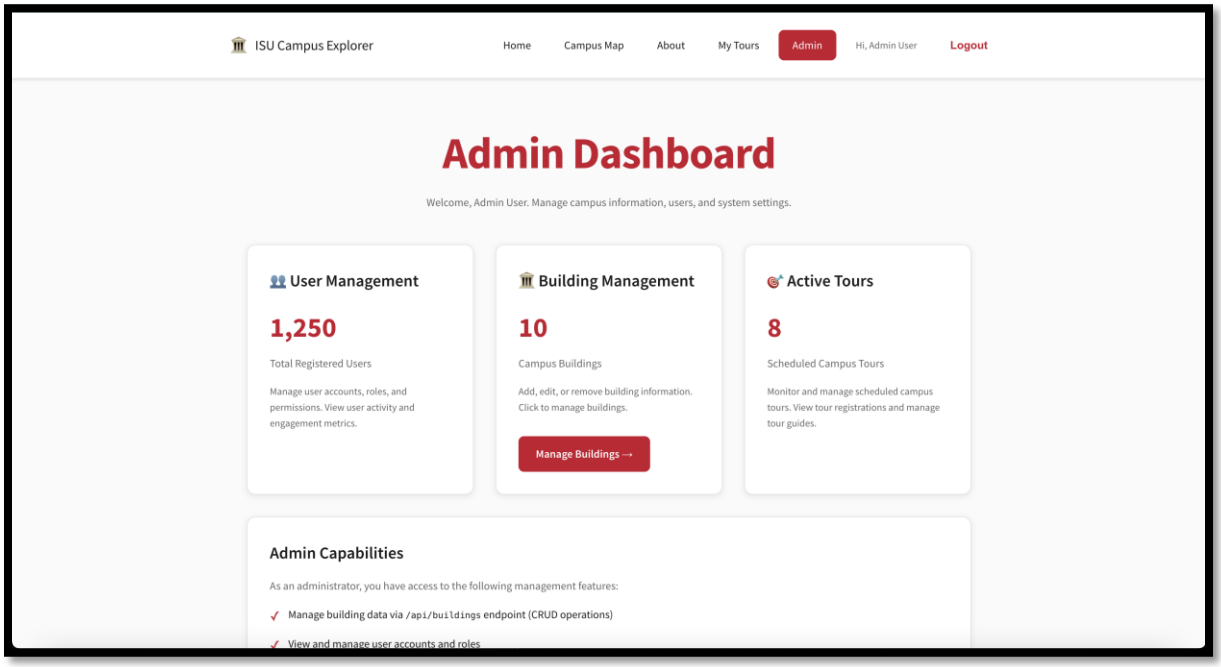
Description: Entry point featuring navigation and project overview.

I**nteractive Campus Map:**



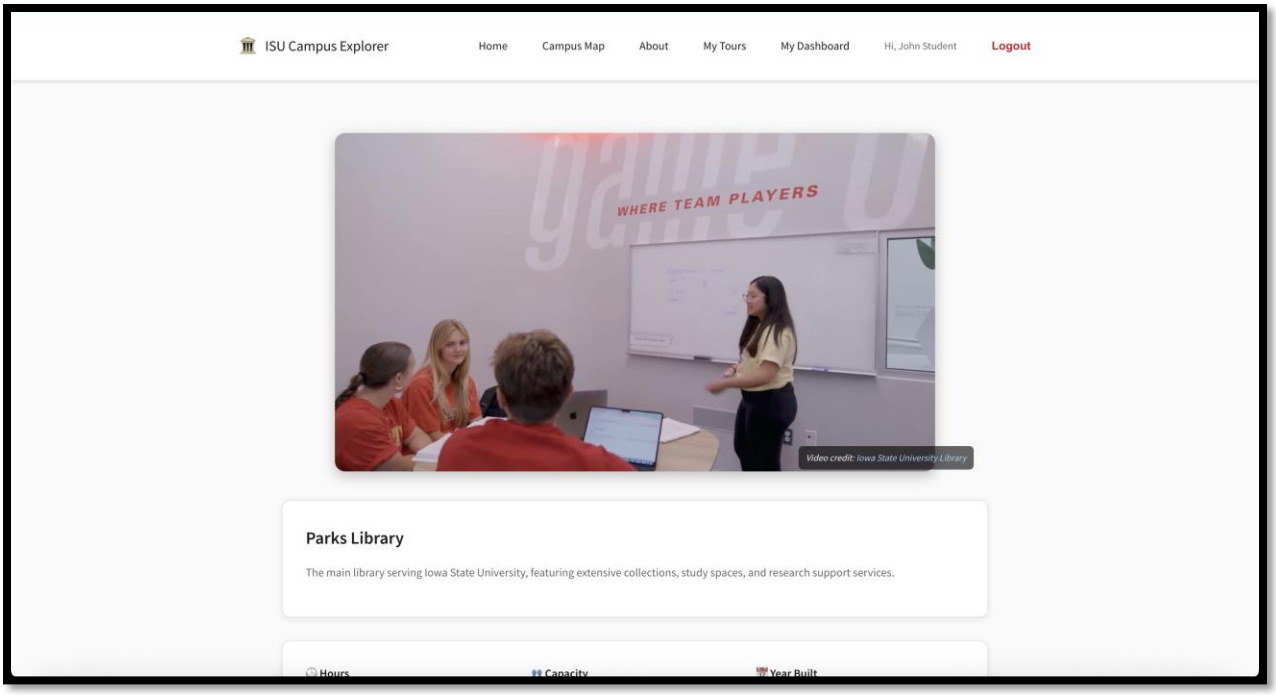
Description: Shows the advanced feature: Red dashed lines visualizing a user's tour route.

Admin Dashboard:



Description: Protected view where admins can manage users and buildings.

Building Detail:



Description: Shows the rich detail view, including the video player and floor plan selector.

6 Installation and Setup Instructions

This section ensures your project can be successfully run by another developer.

6.1 Frontend Setup

Navigate to the frontend folder: cd frontend
Install dependencies: npm install
Start the server: npm run dev
Access at: http://localhost:5173

6.2 Backend Setup

Navigate to the backend folder: cd backend

Install dependencies: npm install (installs express, mongoose, multer, dotenv)

Start the server: npm start

Runs on: http://localhost:3000

6.3 Database Setup

We use **MongoDB Atlas**. The connection string is stored in backend/.env.

1. Create a file backend/.env.

2. Add: MONGO_URI="mongodb+srv://..."
3. Run Seeder: node scripts/seedBuildings.js to populate initial data.

6.4 Environment Variables

MONGO_URI: Connection string for MongoDB Atlas.
PORT: (Optional) Port for the backend server (defaults to 3000).

7 Contribution Overview

Each member must describe their exact contributions. Avoid vague phrases like “helped with backend” be explicit and technical.

7.1 Member 1 Contributions

7.1 Member 1 (Mekhi San) Contributions

- Frontend: Built the Login/Signup pages and the entire Tour Planner flow (creation, listing, confirmation). Implemented the Building Detail page with the interactive floor plan state logic.
- Backend: Developed the auth.js routes for user management and the tours.js routes for the Tour CRUD operations. Created the upload.js route for handling file uploads.
- Features: Owned the "User" side of the application (Authentication + Tours) and the Media CMS.

7.2 Member 2 Contributions

7.2 Member 2 (Ash Bhuiyan) Contributions

- Frontend: Built the Campus Map component, including the SVG logic for rendering buildings and the Admin Drag-and-Drop logic. Developed the Admin Dashboard and Building Management forms.
- Backend: Set up the initial Server architecture and database connection. Developed the buildings.js CRUD routes and the seedBuildings.js script.
- Features: Owned the "Admin" side of the application (CMS + Map Editor) and the core infrastructure.

8 Challenges Faced

8.1 Member 1 Challenges

8.1 Member 1 Challenges (Mekhi)

- **Challenge:** Handling file uploads via the API was difficult because the browser sends data as multipart/form-data, but our Express server was expecting JSON.
- **Solution:** I implemented the multer middleware in the backend to intercept and process the file stream before the route handler, allowing us to save images to the public folder.

8.2 Member 2 Challenges

8.2 Member 2 Challenges (Ash)

- **Challenge:** Implementing the "Visual Route Lines" on the map was tricky because the buildings are SVG elements, but the tour data is just a list of IDs.
- **Solution:** I created a helper function that looks up the X/Y coordinates of each building in the tour list and dynamically generates an SVG <polyline> element to connect them in real-time.

9 Final Reflections

Each team member reflects individually.

9.1 Member 1 Reflection

This project significantly improved my confidence in **Full Stack Integration**. Connecting a React frontend to a real MongoDB database was rewarding. I learned how to manage complex state (like the multi-step Tour Planner) and how to handle user sessions securely. If I started over, I would implement TypeScript to catch data errors earlier.

9.2 Member 2 Reflection

I learned a lot about **System Architecture** and **Data Modeling**. Designing the schema for Buildings and Tours helped me understand relational data in a NoSQL environment. Building the drag-and-drop map editor was the most rewarding part, as it required deep knowledge of DOM events and SVG coordinate systems. Future enhancements would include GPS navigation support.