Campus Insider User Manual & System Documentation

Introduction

Campus Insider is a web application that helps members of learning institutions explore and review campus spaces such as classrooms, study rooms, buildings, and more. Students can browse campus locations, filter based on room type/size/facilities, read and submit ratings, and request new locations to be added.

Primary Features

- User registration & login (students, faculty, staff)
- University searching & selection
- University info pages (campuses, buildings, locations)
- Paginated & filterable location browser
- Rating-based location searching
- Ability for users to request missing locations
- Global authentication state with persistent login
- Responsive UI with TailwindCSS & React

Technology Stack

Frontend

- React + Vite
- React Router
- TailwindCSS
- Context API for global authentication
- Fetch API for all backend calls

Backend

- Flask (Python)
- Flash-Bcrypt for password hashing
- Flask-CORS for cross-origin requests
- MySQL + Stored Procedures
- REST-style API with JSON responses

Installation & Setup

Easy Install & Run

If you are using a device on MacOS, you can run two simple bash scripts to install all needed dependencies and run the web application locally.

Before running either script, make sure you have a .env file in the Phase 3/ directory. This file should contain the following:

```
# .env file
DB_HOST=localhost
DB_USER=root
DB_PASSWORD=[Your root password]
DB_NAME=campus_insider
DB_USER_WEBAPP=app_rw
DB_PASSWORD_WEBAPP=3z@b0k6m # any password works here; just don't change after install
```

Once that is setup, you can run the following commands from the **Phase 3 directory**:
chmod +x ./install.sh
./install.sh

chmod +x ./run.sh
./run.sh

The Flask backend will be running at port 5000 in the background, while the frontend will run on port 5173. The web app should now be running.

Manual Install

Requirements

- Python 3.10+
- Node 18+
- MySQL Server
- Pip, npm, virtualenv

Backend Installation

[info about install and run scripts here]

Backend runs at:
http://127.0.0.1:5000

Frontend Installation

- Install dependencies
    - npm install

- Run the Vite dev server
    - npm run dev
- Frontend runs at:
    - http://localhost:5173

MySQL Setup

Required Tables

You must have the following tables:

- Users
- University
- Campus
- Location
- Buildings
- Rooms
- Nonbuildings
- Rating_equipment
- Rating_accessibillity

Required Stored Procedures

- CreateUser
- RequestRoom
- SearchWithRating

System Architecture

React (Frontend) ←HTTP/JSON→ Flask API (Backend) → MySQL Database

Data Flow Diagram (Example: Creating new user during registration)

User → React Form → POST /api/register → Flask → MySQL Stored Procedure → Result → Flask → JSON Response → UI Success message / navigate to login screen

Usage Guide (User Manual)

Creating an Account

1. From the home screen, click the ‘Sign Up’ button
2. Enter email, password, confirm password
3. Choose your university using the autocomplete search
4. Choose your role at that university
5. Submit

Password is securely hashed with BCrypt

Logging In

- Navigate to Login page

- Enter email and password
- On success, your login is stored in the global AuthContext
- All pages automatically recognize your logged-in state

A persistent session is stored in localStorage.

Navigation (Home, Back, Account Buttons)

Every page includes:

Back Button
- Top-left corner
- Returns to previous page

Home Button
- Top-right corner
- Returns to landing page

Account Button
- Top-right (next to Home)
- Shows user email
- Dropdown: “Logout”

Searching for a University

1. Start typing a university name
2. Autocomplete dropdown appears
3. Clicking a result takes you to the respective university page
4. Clicking the Filter button enables a text box where you can specify to only display results from a specific state

Viewing University Information

University pages include:
- Basic university metadata
- List of campuses
- List of buildings/rooms/non-buildings
- A “Search for a location” button that navigates to location search
- A “Request Location” panel
- The Admin user will see an additional button for location management at that university.

Requesting a New Location

- Users can submit a form to request that a new room be added to the university, given its name/number, the building it’s in, and its campus.

API Documentation

POST /api/login

- Authenticates a user with username + password

Request

{
“username”: test@example.com,
“password”: “secret123”, ← The backend will hash the user input before checking password
}

Successful Response
{
“Message”: “Login successful”,
“User”: {
“username”: test@example.com,
“role”: “student”, ← whatever role they have in the database
“university_id”: 12 ← ID of university account is affiliated with
}
}
Errors

- 401: Invalid username/password

POST /api/register

- Creates a new user via CreateUser stored procedure

Request
{
“username”: test@example.com,
“password”: “secret123”, ← The backend will hash the user input before calling the procedure
“role”: “student”,
“university”: “University of Southern Maine”,
“state”: “Maine”
}

Successful Response
{“message”: “User registered successfully”}

GET /api/search

- Search for universities

Request Params

- q: search text (query)
- state: optional (only display results from that state)

Response
{
“results”: [
{“university”: “University of Southern Maine”, “state”: “Maine”}
]
}

GET /api/university

- Returns university information, locations, and campuses

Params:

- name
- state

GET /api/locationSearch

Advanced location filtering for:

- Type
- Room size
- Room number
- Campus
- Building
- Rating filters

Response returns formatted location entries

POST /api/request-room

- Allows users to request a location addition

Request

```
{
        “room_name”: “102”,
        “university_name”: “University of Southern Maine”,
        “state”: “Maine”,
        “campus_name”: “Portland”,
        “building_name”: “Luther Bonney”,
        “requested_by_username”: “test@example.com”
}
```

Response

```
{“message”: “Room added successfully”}
```

Database Documentation

Entities (Simplified)

university (university_id, name, state, wiki_url)
campus (university_id, campus_name)
location (LID, name, university_id, campus_name)
buildings (LID)
nonbuildings (LID)
rooms (LID, room_number, room_type, room_size, building_LID)
users (username, password, role, university_id)

Stored Procedures

- CreateUser
    - Creates a user and links the appropriate university
- RequestRoom

    - Adds a room or building request and returns a message
- SearchWithRating
    - Filters locations based on rating types and ranges

Security Overview

- Passwords encrypted with BCrypt
- CORS restricted to frontend domain
- Database connections per-request using Flask g
- No token/JWT yet; authorization is handled client-side
- SQL injection prevented through param binding

Future Improvements

- JWT sessions
- Fully functional Admin dashboards
- Location editing (as admin)
- Email verification
- Full analytics panel