Project Summary

Our project goal is to build a web app for students to rate, review, and rank buildings across campus, specifically for water fountains, bathrooms, and vending machines. The application will be an interactive platform for ranking these amenities,helping people make decisions about where to go.

The problem we are trying to solve is to help students be satisfied with the basic amenities of campus buildings. For example with Grainger Library, the bathrooms are unclean, there aren't any vending machines on main floors, and the water fountains are poorly maintained. However, right next to Grainger is CIF with clean and new bathrooms and water fountains worth taking a quick commute for.. This kind of student data in theory could also provide UIUC cleaning staff with data to improve maintenance.

Detailed Description

We will develop a web application where users can rank campus amenities. The three categories of interest are:

Water Fountains – ranked by taste, water pressure, and temperature.

Bathrooms – ranked by cleanliness.

Vending Machines – ranked by variety of snacks, pricing, availability, and working conditions.

The problem we solve: Students often encounter bad quality amenities without warning. By aggregating ratings, we can improve student convenience and experience at various buildings throughout the campus.

Creative Component

To stand out, our project will include:

Interactive Maps: A searchable, interactive campus map where amenities are color-coded by average rating. This will use the Google Maps or OpenStreetMap API with custom overlays.

Gamification & Engagement System: Students earn points/badges for contributing reviews (“Top Bathroom Rater of Grainger Library”). This also includes a game like ranking system

These components are challenging and need multiple APIs, going beyond simple web forms or static data.

Usefulness:

The application is highly useful to the UIUC community. Amenities are things everyone uses on a regular basis no matter what, with high volume of usage. By having quality reporting on the state of amenities at a given building, students can make more accurate decisions about the buildings they frequent, and staff can use this information to determine which buildings need improvement in this regard.:

Basic Features: View ratings,gith

Advanced Features: Interactive map navigation

Comparison with Existing Tools

If you are familiar with the cs ranking website that was somewhat famous last semester, we are trying to do something like that but with uiuc buildings.

Realness (Data Sources)

We will use multiple real data sources to build and enrich the application:

Campus Map Data: UIUC building maps and amenity location data, available in CSV or shapefile formats from the UIUC Facilities & Services GIS datasets.

Student Crowd-Sourced Data: User-submitted ratings stored in a SQL database (PostgreSQL or MySQL). This will grow dynamically as users contribute reviews.

External Sources: Integration with Google Maps API or OpenStreetMap for geolocation and navigation.

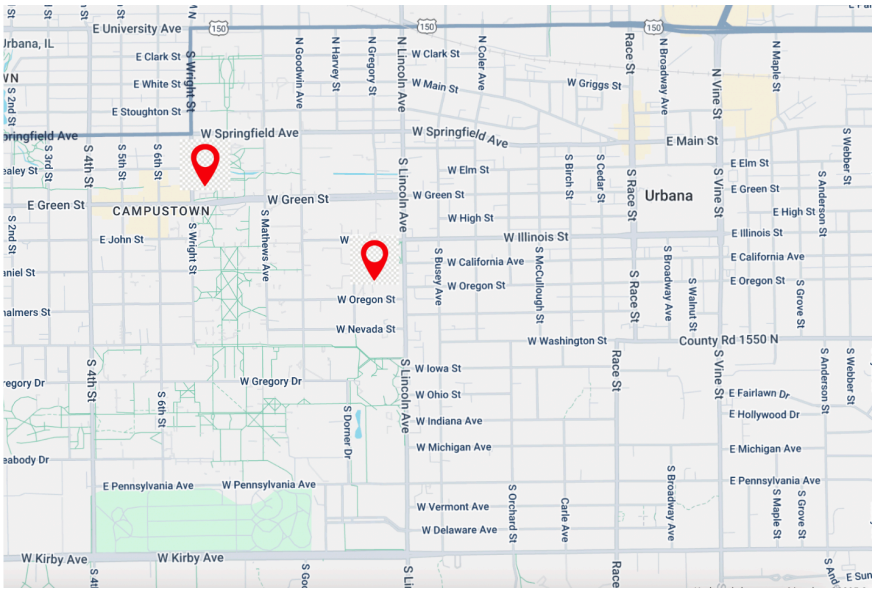
Size: UIUC has 200+ buildings; we anticipate ~100 amenities in the database initially, growing as users add more.

Functionality

Our web app will offer CRUD features, which allow users to add new reviews, update their past reviews, delete them, and view information about campus amenities. Each review will include qualities specific to each amenity type: bathrooms (cleanliness), water fountains (taste, pressure, and temperature), and vending machines (snack variety, pricing, availability, and working condition).

The app will also provide search and filter features so students can locate amenities based on building or type. The amenities will be ranked by scores that combine multiple reviews, and each location will display its overall rating. These results will be available in two formats, a list view that shows the ratings and reviews in a ranking list, and an interactive campus map view with ratings layered.

UI Overview:



Work Overview:

rhand7: Frontend Components and Map UI

samuel67: Database management and data collection

Sgiri6 & siyer35: Middleware and FastApi backend

Entities

User - UserId (PK), UserName, Email, JoinDate : User represents the database of unique users. Each user has its own reviews therefore it must be an entity.

Address - AddressId (PK), Address, Lat, Lon

Building - BuildingId (PK), Name, Address (FK) : Each building is unique and is the common thread for all amenities within the building and their reviews.

Amenity - AmenityId (PK), Type, Floor, Notes, BuildingId (FK): Each amenity is unique and is used to group different reviews. If there could be multiple of each entity reviews would be scattered and unorganized, therefore it must be an entity

Review - ReviewId (PK), OverallRating, RatingDetails, TimeStamp, UserId (FK), AmenityId (FK) : each review is displayed independently and has its own unique details, therefore it is an entity

Tag - TagId (PK), Label (Clean, bottlefiller, cold water,etc)

AmenityTag - AmenityId (PK,FK), TagId (PK,FK)

All IDs are the primary keys here

All attributes in our entities depend only on their primary key ID (UserId, AddressId, BuildingId, AmenityId, ReviewId, TagId,). There are no attributes that depend on other non-key attributes, so there are no partial or transitive dependencies. Therefore our database is properly normalized and satisfies 3NF.

Relationships

Building has amenities, 1 to many : Each building can have any number of amenities, however each amenity can only be in one building.

Amenity has reviews, 1 to many: Each review corresponds to a specific amenity, however an amenity can have any number of reviews

User wrote review, 1 to many optional: A user may or may not have written any number of reviews, however each review was written by a user

Amenities are labeled by Amenity Tags, many to many: Each amenity has one or more amenity tags, and each amenity tag is assigned to at least one amenity.

Relational Schema:
User(UserId: INT [PK], UserName: VARCHAR(50), Email: VARCHAR(255), JoinDate: DATE)

Building(BuildingId: INT [PK], Name: VARCHAR(120), Address: VARCHAR(255), Lat: DECIMAL(9,6), Lon: DECIMAL(9,6))

Amenity(AmenityID: INT [PK],
BuildingId: INT [FK to Building.BuildingId],
Type: VARCHAR(40),
Floor: VARCHAR(20),
Notes: VARCHAR(500))

Review(ReviewID: INT [PK],
AmenityID: INT [FK to Amenity.AmenityID],
UserId: INT [FK to User.UserId],
OverallRating: DECIMAL(2,1),
RatingDetails: VARCHAR(1000),
TimeStamp: TIMESTAMP)

Tag(TagID: INT [PK], Label: VARCHAR(40))

AmenityTag(AmenityID: INT [PK, FK to Amenity.AmenityID],
TagID: INT [PK, FK to Tag.TagID])

Assumptions (per entity)

User (UserId, UserName, Email, JoinDate)
Each account is unique; one person ↔ one user row. Users may have zero or more reviews.

Building (BuildingId, Name, Address, Lat, Lon)
A building is a physical location and the container for amenities. Location fields live on Building (not on Amenity) to avoid duplication.

Amenity (AmenityID, Type, Floor, Notes)
A specific fountain/bathroom/vending machine at a building/floor. We treat Type as an attribute (not a separate table) because the app constrains values

Review (ReviewID, OverallRating, RatingDetails, TimeStamp)
A user's evaluation of exactly one amenity at a point in time. RatingDetails is free-text/JSON for per-type specifics.

Tag (TagID, Label)
A controlled label applied to amenities (e.g., "bottlefiller", "cold water", "accessible"). We model Tag as an entity (not a comma-separated list on Amenity) so tags can be reused, filtered, and managed consistently across many amenities.

Assumptions (per relationship + cardinality)

Building has amenities - 1 to many
One building can contain many amenities, each amenity belongs to exactly one building.

Amenity has reviews - 1 to many
An amenity can receive many reviews, every review evaluates exactly one amenity.

User wrote review - 1 to many (optional on User)
A user may write zero or more reviews, each review is written by exactly one user.

Amenities are labeled by Tags - many to many
Each amenity can have multiple tags, each tag can be applied to multiple amenities.

3NF Normalized Schema:

```
CREATE TABLE User (
   UserId INT PRIMARY KEY,
   UserName VARCHAR(255),
   Email VARCHAR(255) UNIQUE,
   JoinDate DATE
);

CREATE TABLE Address (
   AddressId INT PRIMARY KEY,
   Address VARCHAR(255),
   Lat DECIMAL(9,6),
   Lon DECIMAL(9,6)
);

CREATE TABLE Building (
   BuildingId INT PRIMARY KEY,
   Name VARCHAR(255),
```

```
    AddressId INT,
    FOREIGN KEY (AddressId) REFERENCES Address(AddressId)
);

CREATE TABLE Amenity (
    AmenityId INT PRIMARY KEY,
    BuildingId INT,
    Type VARCHAR(255),
    Floor INT,
    Notes TEXT,
    FOREIGN KEY (BuildingId) REFERENCES Building(BuildingId)
);

CREATE TABLE Review (
    ReviewId INT PRIMARY KEY,
    UserId INT,
    AmenityId INT,
    OverallRating INT,
    RatingDetails TEXT,
    TimeStamp DATETIME,
    FOREIGN KEY (UserId) REFERENCES User(UserId),
    FOREIGN KEY (AmenityId) REFERENCES Amenity(AmenityId)
);

CREATE TABLE Tag (
    TagId INT PRIMARY KEY,
    Label VARCHAR(255)
);

CREATE TABLE AmenityTag (
    AmenityId INT,
    TagId INT,
    PRIMARY KEY (AmenityId, TagId),
    FOREIGN KEY (AmenityId) REFERENCES Amenity(AmenityId),
    FOREIGN KEY (TagId) REFERENCES Tag(TagId)
);
```