Is It Full?

Community-Based University Space Monitoring System

Dmytro Tupkalenko (89252101)

University of Primorska, FAMNIT
Data Engineering and Distributed Systems

January 2026

The Problem

Is the library full right now?

The Unnecessary Journey

- **Walk across university** to study area
- **Discover** it's completely full
- **Repeat** process elsewhere
- **Waste** time on a daily basis

Impact on Students

Lost study time $\Longrightarrow$ **Missed deadlines** $\Longrightarrow$ **Increased stress**

Existing Solutions

Sensor-Based (OpenRoom, Waitz)

Advantages:

+ Precise data

Disadvantages:

- Requires hardware installation
- Needs university approval
- Limited to equipped spaces
- Usage fees + hardware costs

Reservation-Based (MIDAS, Whatspot)

Advantage: Advance planning
Disadvantage: Only reservable spaces, not for open areas

Our Solution

Balanced Crowd-Sourced Approach

User-Generated Spaces

Spaces are controlled by community

Features

- **Traffic Light System**: Free/Busy/Full
- **Update Recency Timestamps**
- **Report System**
- **Verified Members Only**

Crowd-Sourced Updates

Users report status

Reputation System

Community incentives ensure accuracy

Closed Subcommunities

The spaces of concrete university can be accessed by students only

System Requirements Overview

Functional Requirements

- University Management
- Space CRUD Operations
- User Authentication
- Email + Domain Verification
- Occupancy Reporting

Technical Requirements

- Relational Database
- Data Integrity Constraints
- Space: Composite Design Pattern
- Password Hashing
- Query-based Occupancy

Technology Stack

Django Framework

Why Django?

- Rapid development
- Built-in security (CSRF, XSS, SQL injection)
- Powerful ORM
- "Simple" future migration to more complex architecture if needed

PostgreSQL Database

- **Recursive CTEs:** Native support for hierarchical data queries
- **MVCC:** High concurrency with minimal locking
- **Django Integration:** Native Django ORM support

Architecture: MVT Pattern

Model-View-Template (Django's MVC variant)

Model Layer

Data structure & business logic (University, Space, User)
Composite pattern for hierarchical spaces

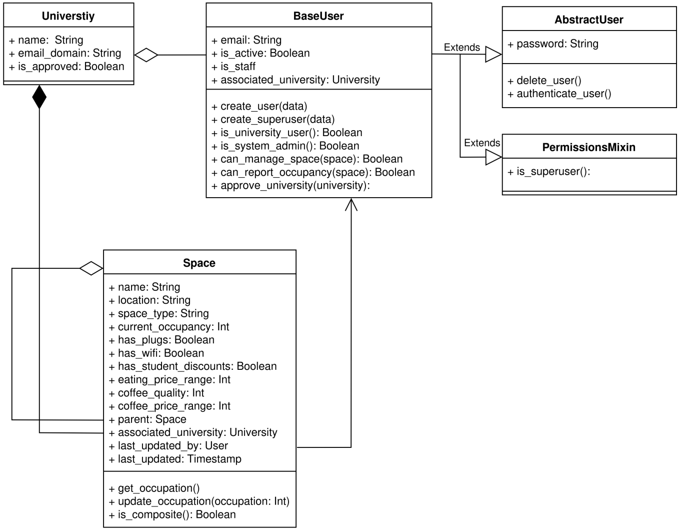
View Layer

Request handling & authorization
CRUD operations, form validation, access control

Template Layer

Server-side rendered HTML presentation
Responsive UI, conditional rendering, reusable components

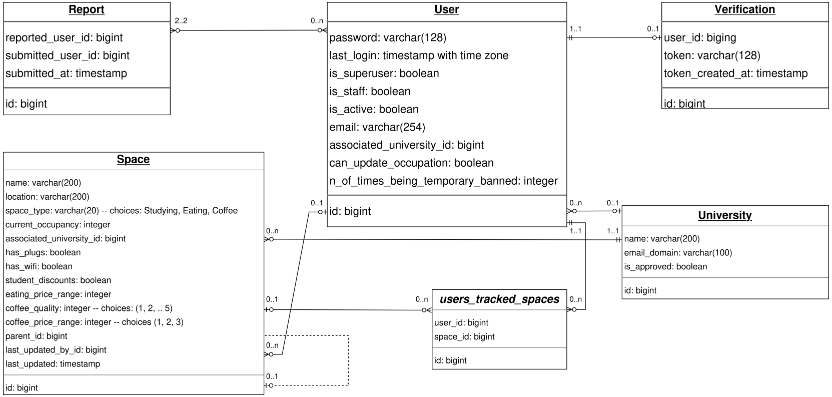
Class Diagram



Class Associations

- **BaseUser**
  - *Inheritance:* Django Auth functionality.
  - **Aggregation:** With `University`.
- **Space**
  - **Composition:** With `University`.
  - **Self-Aggregation:** Via `parent`.
  - **Association:** With `BaseUser`.
- **University**
  - Registered institutions.

Database Schema



Key Relationships

Uni ↔ **User** (1:m)
Uni ↔ **Space** (1:m)
User ↔ **Space** (m:m)
User ↔ **Report** (m:2)
User ↔ **Verify** (1:1)
Space ↔ **User** (updated by, m:1)

Composite Design Pattern

Hierarchical Space Structure

Self-referential foreign key in Space enables parent-child relationships

Example Hierarchy

- **Main Library** (parent)
  - Reading Room - Floor 1 (child)
  - Reading Room - Floor 2 (child)
  - Study Area - Basement (child)

Common Interface

- `get_occupancy()` - Recursively calculates from children
- `get_all_descendants()` - Traverses tree structure

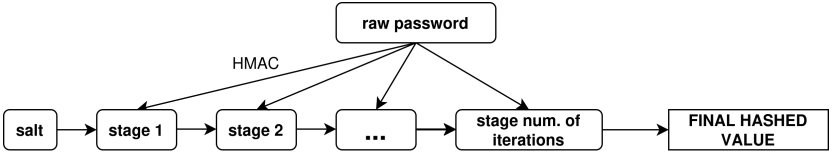
Authentication: Django Password Management

PBKDF2-HMAC-SHA256

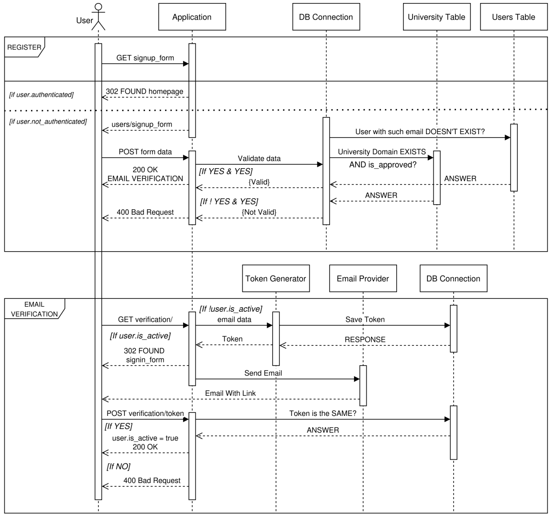
Django's built-in password hashing

Storage Format

`pbkdf2_hmac_sha256$720000$random_salt$hash_output`



Authentication Flow



Phase 1: User Registration

1. User accesses signup form
2. Submits email + password
3. Server validates: Email uniqueness, University domain exists & is approved (database query)
4. Outcomes: If valid $\rightarrow$ Email verification phase, If invalid $\rightarrow$ 400 Bad Request

Phase 2: Email Verification

1. System generates unique verification token
2. Token stored in database (linked to user)
3. Verification email sent with token link
4. User clicks link $\rightarrow$ submits token
5. System validates token against database
6. **If valid $\rightarrow$ Account activated**

Dashboard Interface

Public Dashboard

Displays all active spaces within the university
Community-wide overview of available spaces

Private Dashboard

Shows only user's tracked spaces
Personalized view via junction table (users_tracked_spaces)

Display Features

- ● Free ● Busy ● Full
- Last update timestamp
- Space details

Preventing Misuse

DDoS-Style Attacks

Problem: Repeated university or spaces submission requests

- Limit pending universities (Redis cache layer)
- Limit spaces per university ($\sim$20, form validation - the aggregated count of the tracked spaces)

Misreporting

- **Metadata edits:** Community approval (10% of trackers)
- **Occupancy misreports:** Community reporting within 5 min window after submission of update
- **Consequences:** 3 verified reports $\rightarrow$ temporary ban;
  3 temp bans $\rightarrow$ permanent ban from submitting updates

Thank You!

Questions?

Dmytro Tupkalenko
University of Primorska, FAMNIT
GitHub: github.com/tupkalenkodi/is_it_full