DESIGN SPECIFICATION

Smart Scheduler and Reminder System

Author: Changyu Jia

Student ID: C00292876

Programme: SETU — BSc Software Development (Year 4)

Supervisor: Dr Jamal Tauseef

Date: 1/10/2025

CONTENTS

1. System Overview 1
1.1 Project Background and Objectives 1
1.2 System Characteristics 1
2. Technical Architecture 3
2.1 Overall Architecture Design 3
2.2 Technology Stack Details 4
3. Core Module Design 5
3.1 Account Module 5
3.2 Campus Module 5
4. Database Design 6
4.1 User Module 6
4.2 Schedule Event Module 7
4.3 Reminder System 7
5. API Design 8
6. Security and Deployment Specifications 12
6.1 Security Specifications 12
6.2 Deployment Requirements 12
7. Scalability and Maintainability Design 12

1. System Overview

1.1 Project Background and Objectives

In a campus environment, students and faculty need to efficiently manage their personal schedules, course schedules, campus events, and task deadlines. Traditional calendar tools lack optimization for the campus environment and cannot intelligently identify schedule conflicts or provide personalized suggestions.

Therefore, we developed an intelligent campus schedule management system that integrates personal schedule management, campus event discovery, intelligent reminders, and conflict detection. This system aims to help users optimize time management, thereby improving the efficiency and quality of campus life.

1.2 System Characteristics

- **Conflict Detection and Automatic Resolution**: The system can automatically detect potential conflicts in user schedules regarding time and location and provide clear conflict alerts.

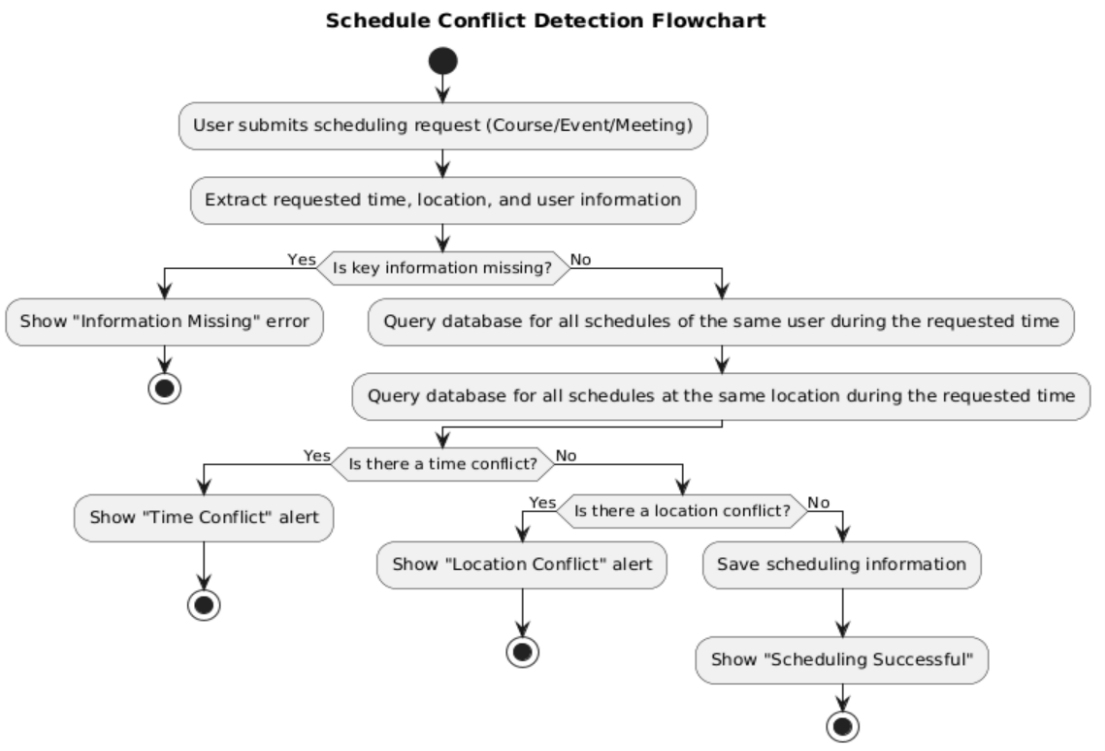

Figure 1.2.1 Schedule conflict detection flowchat

- **Flexible Notification System**: Supports multiple notification channels (e.g., email, in-app push) and allows users to customize reminder times and frequency based on event type and importance, enabling personalized message management.

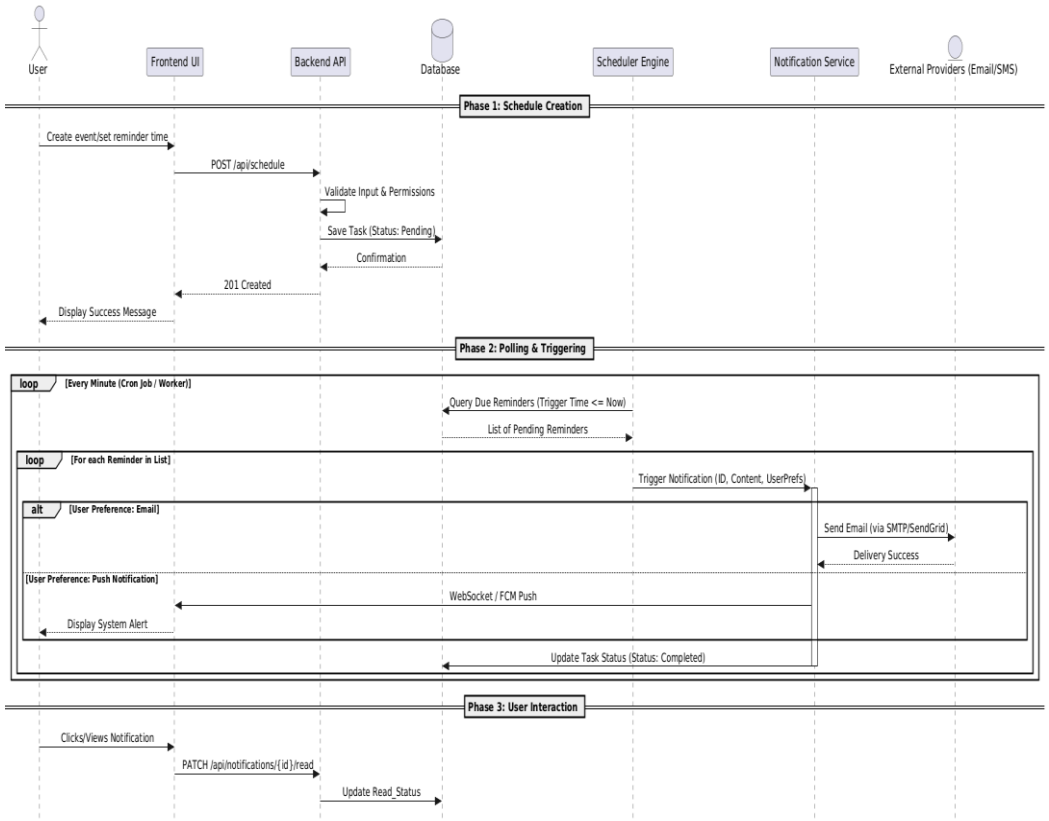

Figure 1.2.2 Notification system sequence diagram

- **Modular Design**: Adopts a modular design philosophy, dividing the system into highly cohesive and loosely coupled functional modules. This architecture ensures the system is easy for functional expansion, module maintenance, and subsequent iterative upgrades.

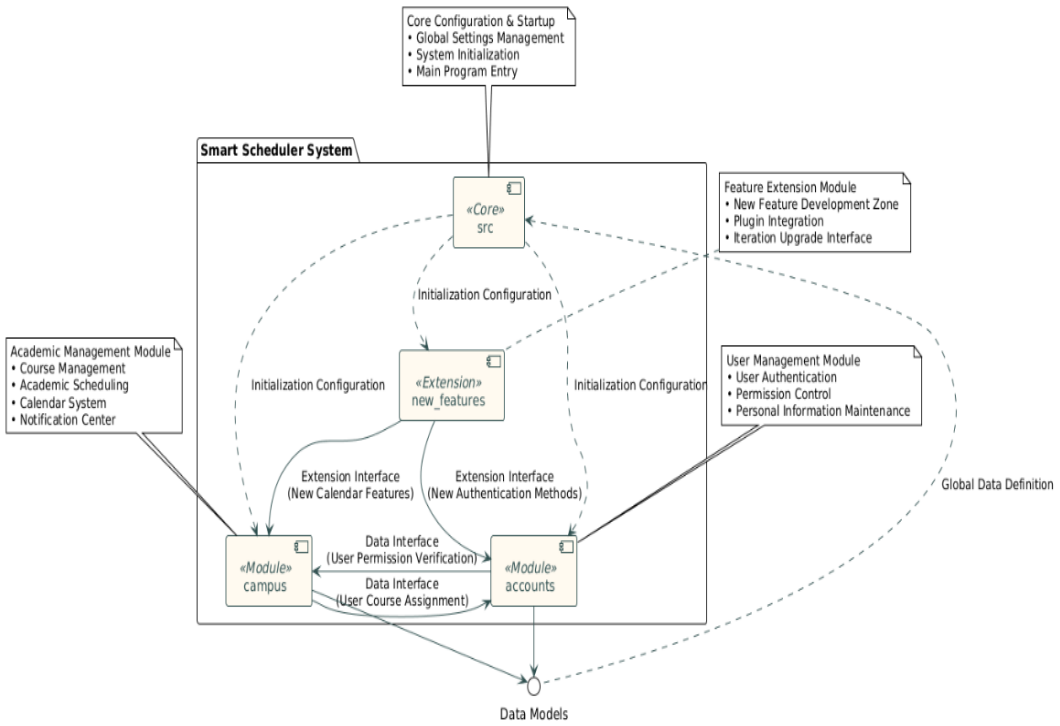

Figure 1.2.3 Module framework diagram

2. Technical Architecture

2.1 Overall Architecture Design

The system adopts a typical three-tier architecture pattern to achieve separation of concerns, ensuring good maintainability and scalability.

- **Presentation Layer**: Front-end presentation based on the Django template engine, combined with Bootstrap for responsive layout, ensuring a good user experience on both desktop and mobile devices.
- **Business Logic Layer**: The Django framework handles core business logic, and the Celery asynchronous task queue processes time-consuming operations (e.g., sending emails, generating complex reports), improving system response speed.
- **Data Access Layer**: Abstracts database operations through Django ORM, supporting SQLite for the development environment and PostgreSQL for production. Redis is used as a cache and message broker for Celery, enhancing system performance.

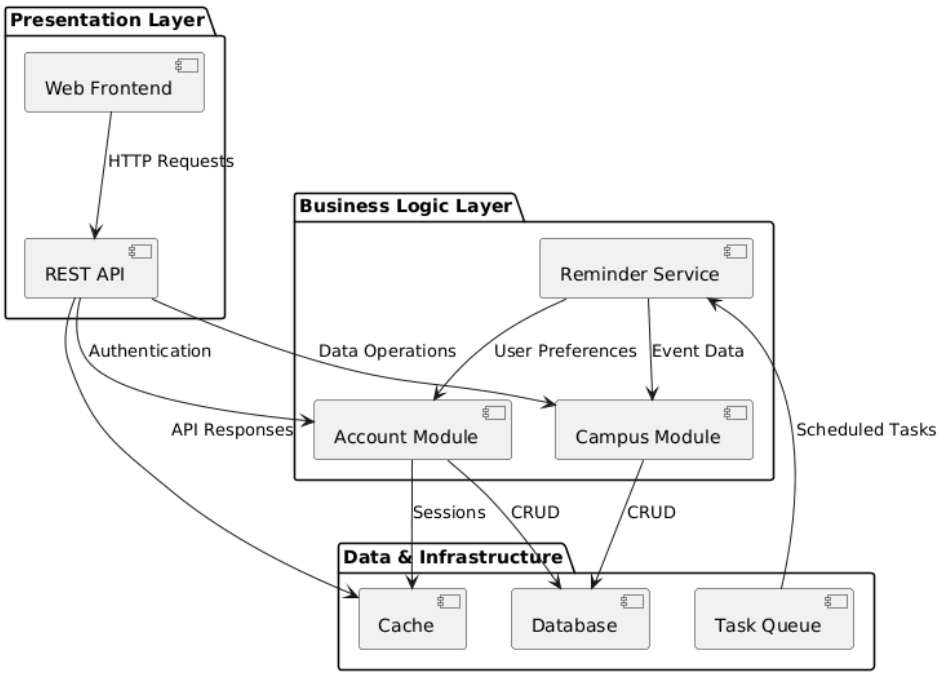

Figure 2.1. Architecture component diagram

2.2 Technology Stack Details

- **Frontend Technology**: Based on HTML5, CSS3, and JavaScript, uses Django template syntax to organize page structure, implements responsive layout via Bootstrap, and utilizes jQuery/Ajax for front-end interaction logic.
- **Backend Technology**: The core framework uses Django 4.0+ (based on Python 3.8+), employs Django REST Framework (DRF) to build RESTful APIs, handles asynchronous tasks via Celery + Redis, and utilizes Django's built-in powerful authentication system for user management.
- **Data Storage**: Uses the lightweight SQLite database during the development stage, with plans to migrate to the more stable and concurrency-capable PostgreSQL database for the production environment. Simultaneously, Redis uses the cache layer for session storage.

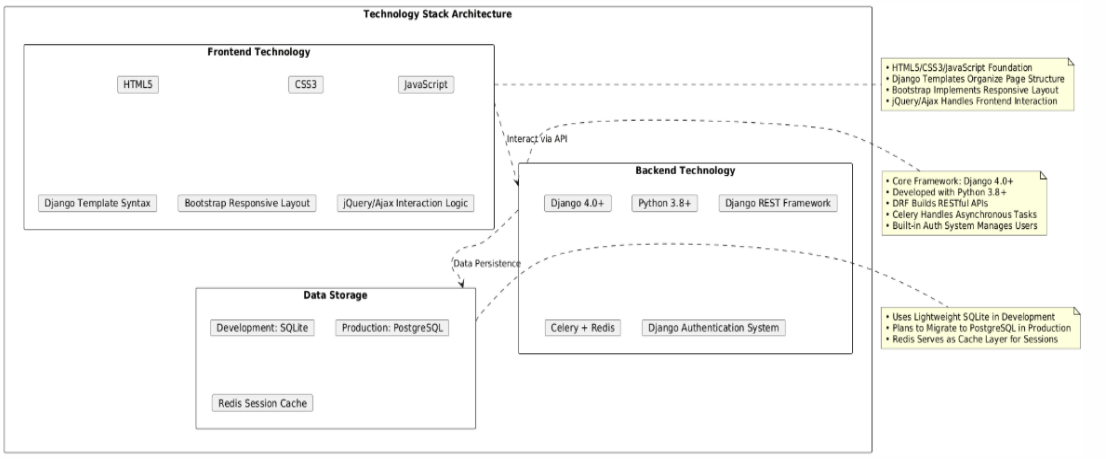

Figure 2.2 Technology stack architecture

3. Core Module Design

3.1 Account Module

This module handles all functions related to user identity.

- **User Authentication and Management:** Extends the Django built-in authentication system, fully supporting user registration, email/password login, session management, and secure password reset processes.
- **Reminder Preference Settings**: Provides granular reminder settings, allowing users to configure advance notice times (e.g., 15 minutes, 30 minutes, 1 hour) independently for different types of events (e.g., classes, meetings, activities), and set repeat reminder rules.
- **Notification Method Configuration**: Allows users to choose their preferred notification methods (e.g., email or in-app alerts) and customize notification content templates.

3.2 Campus Module

This module is the core embodiment of campus scenario functionalities. (Figure 2.)

- **Campus Activity Management**: Provides capabilities for creating, publishing, editing, deleting, and categorizing various activities on campus, and supports viewing real-time participant numbers.

- **Event Recommendation System**: Initially provides simple recommendations based on popularity and basic user information, laying the groundwork for future integration of more complex recommendation algorithms.
- **Schedule Conflict Detection**: Checks in real-time whether adding or modifying a schedule conflict with existing schedules in terms of time or location, providing users with clear conflict warnings.

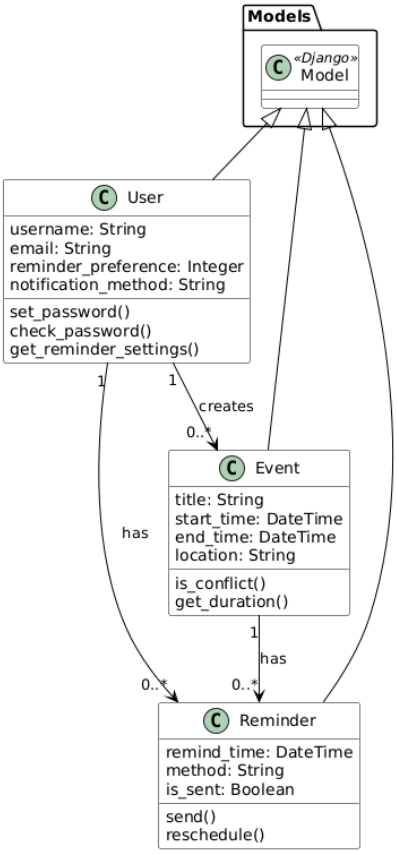

Figure 3.2. System structure class diagram

4. Database Design

4.1 User Module

The User module is designed by extending Django's built-in AbstractUser base model to incorporate additional user information. It includes essential fields such as a unique email address and phone number for basic user identification. The module also implements personalized notification settings, allowing users to configure their reminder preferences including advance notice timing and preferred notification

methods. Furthermore, role-based differentiation through a teacher identity flag (is_teacher) and department enable tailored functionality based on user roles within the campus ecosystem.

4.2 Schedule Event Module

The event module supports comprehensive scheduling needs by defining multiple event types—including lecture, meeting, exam, and activity—to accurately categorize campus occurrences. Each event records essential details such as the event title, description, type, start and end time, and location. It also establishes relational data by associating both the creator and participants with each event. Additionally, to accommodate academic use cases, the module includes teaching-related fields such as a course code and a recurrence flag (is_recurring) for managing repeated or series-based events.

4.3 Reminder System

The reminder system is designed to associate specific events with individual users, recording key details.

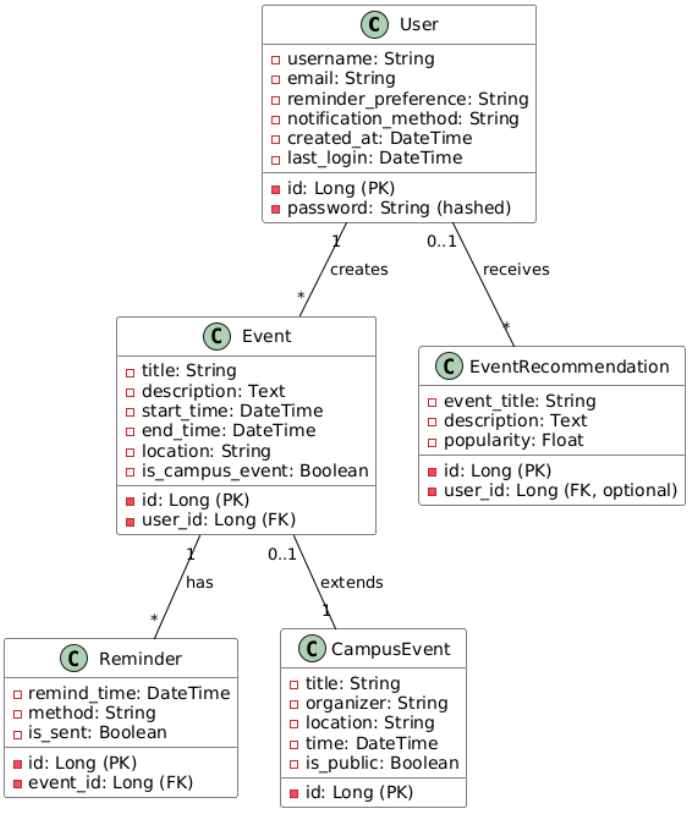

Figure 4.3. Database Entity Diagram

5. API Design

- **API Design Principles:** The system's backend interfaces follow the RESTful architectural style, operating on resources through standard HTTP methods (GET, POST, PUT, DELETE), ensuring the interface structure is clear, easy to understand, and use.
- **Authentication Mechanism:** Uses JWT (JSON Web Token) as a stateless authentication mechanism. After login, users obtain a token which is carried in subsequent requests, effectively ensuring the security and scalability of API access, particularly suitable for multi-endpoint scenarios (Web, Mobile App) .

URL Path	HTTP Method	View/Function	Description
/auth/login/	GET/POST	UserLoginView	User login
/auth/signup/	GET/POST	SignupView	User registration
/auth/profile/<staff_name>/faculty/	GET/POST	EditFacultyStaffProfileView	Edit faculty profile
/auth/profile/<_student_name>/student/	GET/POST	EditStudentProfileView	Edit student profile
/auth/logout/	GET	LogoutUserView	User logout

/campus/u/admin-dashboard/	GET	AdminDashboardView	Admin homepage
/campus/u/homepage/	GET	StudentHomepageView	Student homepage
/campus/u/lecturer/<lecturer_id>/calendar/	GET	LecturerCalendarView	Lecturer calendar view
/campus/u/lecturer/<lecturer_id>/find-slots/	GET	find_available_slots	Find available slots for lecturer
/campus/u/lecturer/<lecturer_id>/request-meeting/	GET/POST	request_meeting	Student requests meeting with lecturer
/campus/u/calendar/events/add/	POST	add_personal_event	Add personal event
/campus/u/calendar/events/get/	GET	get_personal_events	Get personal events
/campus/u/calendar/events/update/<event_id>/	POST	update_personal_event	Update personal event
/campus/u/calendar/events/delete/<event_id>/	POST	delete_personal_event	Delete personal

			event
/campus/u/units/<student_id>/register/	GET/POST	StudentsUnitsRegistrationView	Student course registration
/campus/u/lectures/<_student>/	GET	StudentsLecturesDetailView	Student course details
/campus/u/lecture/<lecture_id>/schedule/<_student>/confirm/	POST	LectureAttendanceConfirmationView	Student confirms course attendance
/campus/u/halls/<hall_id>/feedback/	POST	SubmitFeedbackView	Student submits feedback
/campus/u/dashboard/	GET	FacultyDashboardView	Faculty homepage
/campus/u/manage-requests/	GET	ManageMeetingRequestsView	Faculty manages meeting requests
/campus/u/approve-request/<request_id>/	POST	approve_meeting_request	Faculty approves meeting request
/campus/u/reject-request/<request_id>/	POST	reject_meeting_request	Faculty rejects

			meeting request
/campus/u/edit-request/<request_id>/	GET/POST	EditMeetingRequestView	Faculty edits meeting request
/campus/u/<staff_id>/lecture/<staff_name>/schedule/	GET/POST	ScheduleLectureView	Faculty course schedule list
/campus/u/<staff_id>/lecture/<staff_name>/schedule/<unit_id>/	GET/POST	ScheduleLectureView	Faculty submits course schedule
/campus/u/units/book/	GET/POST	AssignUnitsforLecturersView	Lecturer assigns courses page
/campus/u/records/faculty/<staff_id>/<staff_name>/	GET/POST	LecturesDetailView	Faculty course records
/campus/u/accept-meeting-request/<request_id>/	POST	accept_meeting_request	Faculty accepts meeting request
/campus/u/notifications/all/	GET	view_all_notifications	View all notifications

6. Security and Deployment Specifications

6.1 Security Specifications

Security is a primary consideration in the system design.

- **Password Security**: User passwords are never stored in plain text but are encrypted using industry-strength hashing algorithms (PBKDF2) to fundamentally prevent password leakage risks.
- **Input Validation and Protection**: All user inputs are strictly validated and filtered to effectively prevent common web attacks such as SQL injection and cross-site scripting (XSS), while CSRF protection mechanisms are enabled.

Figure 6.1 Password input verification and protection

6.2 Deployment Requirements

- **Programming Language**: Requires Python version 3.8 or above to utilize its modern language features and performance optimizations.
- **Service Dependencies**: Requires the deployment of Redis service, which acts as the message broker for Celery and the cache database, making it a key component for system asynchronous tasks and performance enhancement.

7. Scalability and Maintainability Design

Use Django's App mechanism to organize code. Each functional module (e.g., accounts, campus) is developed, tested, and maintained independently, significantly improving code readability and team collaboration efficiency, laying a solid foundation for the system's long-term evolution.