3090 Dashboard

COMS 402 SENIOR DESIGN DOCUMENT

Team number: fs_4

Client: Srdan Kopunovic
Advisors: Simanta Mitra

TA: Fatema Siddika

Team Members
Lev Schaul
Lottie Peterson
Maria Sofia Sanchez
Rida Azam
Vinayak Trigunayat

Team Website:
https://seniord.cs.iastate.edu/2026-March-41/

EXECUTIVE SUMMARY

Development Standards & Practices Used

- Version Control: Git (project hosted in GitLab; branch-based workflows)
- CI/CD: GitLab CI / Docker-based pipelines (CI for linting, tests, builds)
- Backend Framework: Django (Django REST Framework for API design)
- Frontend / Client: Expo (React Native) with Tamagui for UI; web support via Expo/React Native Web
- Desktop Packaging (considered): Electron (listed as considered for desktop packaging)
- Containerization/Deployment: Docker, docker-compose for local dev and containerized deployment
- Coding Practices: PEP8 / ESLint style enforcement where applicable; code reviews and merge requests; unit and integration testing where implemented
- Security Practices: Role-based access control, token/session authentication, environment variable configuration for secrets, use of HTTPS in production recommended
- Data Practices: Use of relational DB with migrations managed by Django; periodic backups and consistent migrations workflow

Summary of Requirements

- Authentication & Roles: Role-based users (professor, TA, head TA, student) with different UI/permissions.
- FERPA compliant: Comply with FERPA regulations regarding data storage. Ensuring that sensitive data, such as personal email addresses and images, are not stored on ISU servers and instead are either not used at all or stored locally.
- Responsive: Users can view and use the application on mobile and web.
- Tasks: Create, assign, update tasks; tasks have statuses (e.g., todo, in-progress, completed).
- Groups: Create, edit, delete student project groups; view group progress and members.
- Students: Create, edit, delete student records; view student progress and flags.
- Comments: Add and delete comments tied to student progress or group demos.
- Sections: Organize students into course sections; manage section-level settings and assignments.
- Notifications (optional): Activity notifications for task assignments and comments.
- Admin: Admin UI for user/section management (Django admin available for backend admins).
- Persistence: All entities persisted in a relational database with migrations.
- Online-only: The app requires an internet connection for full functionality.

Applicable Courses from Iowa State University Curriculum

List all Iowa State University courses whose contents were applicable to your project.

- COMS 3090 - Software Development Practices
- COMS 3190 - Construction of User Interfaces
- COMS 3630 - Introduction to Database Management Systems

New Skills/Knowledge acquired that was not taught in courses

- Setting up a backend using Django and Django REST Framework.
- Integrating Expo (React Native) with a Django REST backend.
- Containerizing a full-stack app with Docker and docker-compose.
- Using Tamagui for cross-platform UI (React Native + web).

Table of Contents

Development Standards & Practices Used 2
Summary of Requirements 2
Applicable Courses from Iowa State University Curriculum 3
New Skills/Knowledge acquired that was not taught in courses 3
Table of Contents 4
1 Introduction 6
1.1 Acknowledgement 6
1.2 Problem and Project Statement 6
1.3 Operational Environment 6
1.4 Requirements 7
1.5 Design Assumptions and Limitations 7
2 Architectural Diagram 8
3 Components 9
3.1 Component 1 - Client Component 9
3.1.1 Homepage Module 9
3.1.2 Sign-In Module 9
3.1.3 Individual Group View Module 10
3.1.4 Individual Student View Module 10
3.1.5 Tasks Module 10
3.1.6 Profile View Module 10
3.2 Component 2 - Backend Component 10
3.2.1 Authentication and Authorization Module 11
3.2.2 Groups API Module 11
3.2.3 Students API Module 11
3.2.4 Tasks API Module 11
3.2.5 Comments API Module 11
3.2.6 Demos and Status API Module 12
3.2.7 Calendar Events API Module 12
3.2.8 Sections and Semesters API Module 12
4 Data Decomposition 13
5 Design Rationale 14
5.1 Design Issues 14
5.2 Backend Setup 15
5.2.1 Description 15
5.2.2 Factors affecting Issue 15
5.2.3 Alternatives and their pros and cons 15

5.2.4 Resolution of Issue.....15
5.3 Migration Conflicts.....15
5.3. Description.....15
5.3.2 Factors affecting Issue.....16
5.3.3 Resolution of Issue.....16

1 Introduction

1.1 Acknowledgement

We would like to thank our professor Simanta Mitra and our client Srdan Kopunovic for guidance and feedback throughout the project.

1.2 Problem and Project Statement

The Department of Computer Science at Iowa State University offers COMS 3090, Software Development Practices, a required course for Computer Science, Software Engineering, and Computer Engineering students. Each semester, the course includes multiple full sections with many student teams, weekly meetings, demo evaluations, and course staff responsibilities managed by professors, teaching assistants (TAs), and head TAs. As the number of teams and sections grows, so does the amount of information that must be tracked, including project progress, student status, teaching staff tasks, attendance, and comments. Today, much of this information is scattered across Discord channels, spreadsheets, and announcements, which can create confusion, increase workload, and make it easy for important details to be overlooked.

To address this problem, the 3090 Dashboard was created as a centralized web and mobile application for COMS 3090 staff and students. The system condenses student, group, task, and communication data into one accessible location so instructors and TAs can efficiently manage their sections. It provides tools for viewing and editing group project statuses, individual student feedback, teaching staff tasks, class attendance, and comments, while also giving students access to their own group progress and demo feedback. The application is being built as a responsive Expo-based client for web and mobile, supported by a Django REST backend that exposes APIs for role-based functionality, group and task management, student tracking, comment threads, and section administration.

1.3 Operational Environment

- Client devices: modern browsers (desktop, mobile) and mobile devices running Expo (iOS/Android).
- Backend: hosted behind HTTPS in a cloud environment (Dockerized container images recommended).
- Database: managed relational DB (SQLite for development; Postgres recommended for production).
- The app is online-only (requires network connectivity).

1.4 Requirements

Functional requirements (high level):

- Authentication and role-based access.
- CRUD for Tasks, Groups, Students, Comments, Staff, Events, Semesters.
- Assignment workflows (assign tasks to groups or students).
- Dashboard views for progress and flags.
- Admin tools for sections and users.
- Adherence to FERPA regulations for proper data hiding.
- Images for students and groups stored locally.

Non-functional requirements:

- Responsive design, cross-platform UI.
- Reasonable performance for class-sized data sets (tens to a few hundred students).
- Secure storage of credentials and sensitive data via environment variables.
- Maintainable codebase with CI checks.

1.5 Design Assumptions and Limitations

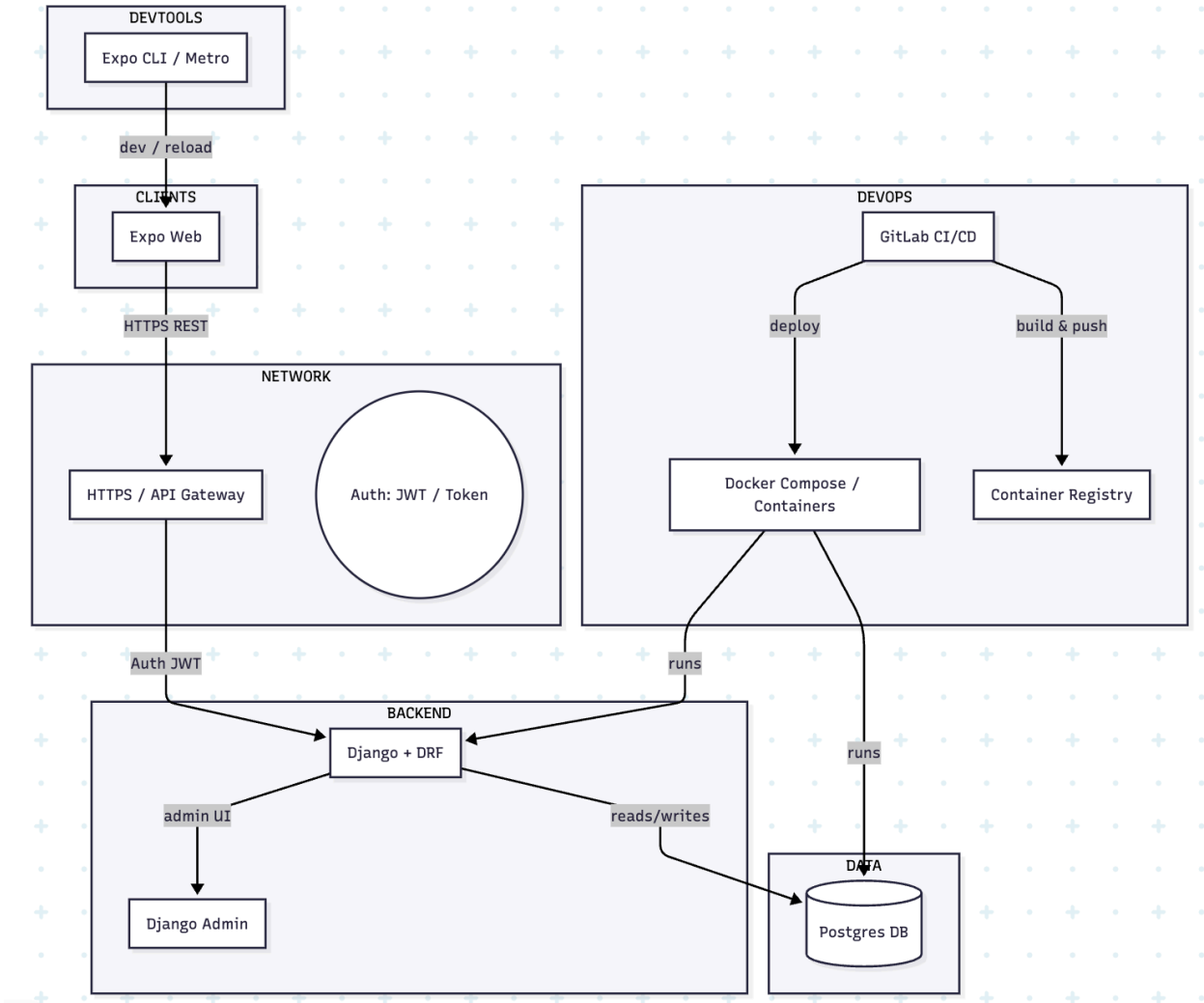
Assumptions:

- Typical simultaneous active users are small (class + TAs), not large-scale concurrent traffic.
- Client devices run modern browsers or supported Expo platforms.
- Instructors/TAs will have accounts and roles provisioned by admin.

Limitations:

- No offline-first functionality (online-only).
- Not designed for large-scale multi-course deployments without additional scaling.
- Some production concerns (load-balancing, high-availability DB, backups) require extra ops work not covered by the project.

2 Architectural Diagram



The 3090 Dashboard is organized as a client-server system. Users interact with the system through the Expo client, which provides the mobile and web interface for viewing sections, groups, students, tasks, and comments. The client sends requests over HTTPS to the backend API, where Django and Django REST Framework handle authentication, business logic, and permission checks. Role-based access control ensures that professors, TAs, and students each see and modify only the data relevant to their role.

The backend persists all application data in a relational database such as SQLite for development or PostgreSQL for deployment. Core entities include users, sections, students, groups, tasks, and comments, with relationships between them managed through Django models and migrations. The backend also exposes an admin interface for direct administrative management when needed.

For development and deployment, Docker and docker-compose can be used to run the backend and database consistently across environments, while GitLab CI/CD supports automated builds, testing, and container deployment. Electron is a considered desktop packaging option, but the primary production experience is the Expo-based mobile/web client.

3 Components

3.1 Component 1 - Client Component

The client component is the user-facing portion of the 3090 Dashboard application. It is responsible for presenting information to users, handling navigation between pages, and collecting user input for actions such as signing in, viewing groups, managing tasks, and editing profiles. This component is implemented as an Expo-based application so that it can run on both mobile and web platforms with a shared codebase. The client communicates with the backend through API requests and updates the interface dynamically based on the signed-in user's role and permissions.

3.1.1 Homepage Module

The Homepage module serves as the landing page after a user signs in. It displays a summary of the groups and tasks associated with the user's role and responsibilities. This module depends on the Groups API, Tasks API, and Users API. It provides access to group details through selectable group cards or entries, allows navigation through the header bar, and presents current tasks in a scrollable view. Groups may be filtered by status or TA if the user is a Professor or head TA.

3.1.2 Sign-In Module

The Sign-In module manages authentication for the application. It allows users to sign in using their email and password and, if enabled, supports sign-up or third-party authentication through Google OAuth. This module depends on Clerk, Google OAuth, and the Users API. Its primary purpose is to verify user identity and grant access to the appropriate parts of the application.

3.1.3 Individual Group View Module

The Individual Group View module displays detailed information for a specific student group. It includes the group name, project name, student members, demo progress, and comments made

by teaching staff. Authorized users can also update group status information and submit new comments on each demo. This module depends on the Groups API, Students API, and Comments API.

3.1.4 Individual Student View Module

The Individual Student View module presents detailed information about an individual student. It shows the student's name, associated group, demo progress, flags, and comments from staff. Authorized users can update student status information and add new comments. This module depends on the Students API and Comments API.

3.1.5 Tasks Module

The Tasks module is used to display and manage tasks assigned to a user, group, or section. Users can create new tasks, view task details, and move tasks between status categories such as to-do, in-progress, and completed. This module depends on the Tasks API and Users API, and may also reference the Groups API when tasks are assigned to groups.

3.1.6 Profile View Module

The Profile View module displays information about the signed-in user, including their name, role, and any groups or responsibilities assigned to them. It also allows the user to update permitted profile information. This module depends on the Users API.

3.2 Component 2 - Backend Component

The backend component provides the data management and business logic for the 3090 Dashboard application. It is responsible for user authentication, permission enforcement, data storage, and request handling. Implemented in Django, the backend exposes REST API endpoints consumed by the client component and maintains the persistent records for users, groups, students, tasks, and comments.

3.2.1 Authentication and Authorization Module

The authentication and authorization module is responsible for verifying the identity of every incoming request and enforcing access control across the API. Authentication is handled through Clerk-issued JSON Web Tokens (JWTs). On each request, the ClerkJWTAuthentication class

extracts the Bearer token from the Authorization header, fetches Clerk's public signing keys via the JWKS endpoint (cached with a one-hour TTL), and validates the token's signature, expiry, and issuer. As an additional domain restriction, tokens associated with non-@iastate.edu email addresses are rejected outright. Once a token is verified, the module resolves the corresponding application user. It first attempts a fast lookup by Clerk ID. If no match is found (i.e., the user's first sign-in), the module derives the user's netID from their email address and locates the pre-seeded Staff or Student record, then persists the Clerk ID for future fast-path resolution.

Authorization is enforced through DRF permission classes. The IsStaff class permits only authenticated Staff members to access protected endpoints. More restrictive classes (IsHeadTAOrProfessor, IsProfessor) gate actions reserved for senior roles. These classes are applied globally and individually to ViewSets as needed.

3.2.2 Groups API Module

The Groups API module manages the creation, retrieval, update, and deletion of student groups. Each group belongs to a section, may have TAs assigned, and holds a many-to-many relationship with its student members. The module exposes a move_student action for transferring a student between groups. Group responses include nested demo status summaries for both individual students and the group as a whole. Filtering by section and semester is supported.

3.2.3 Students API Module

The Students API module handles student records, including name, netID, and a staff-controlled flag used to mark students of interest. The module supports listing and filtering students by group or section and exposes endpoints for creating and updating student records.

3.2.4 Tasks API Module

The Tasks API module manages tasks associated with groups and sections. Each task includes a name, description, due date, status (Backlog, In Progress, or Resolved), and references to the assigning and assigned staff members. The module supports filtering by group and section.

3.2.5 Comments API Module

The Comments API module stores written feedback left by staff on individual students. Each comment records the title, body, authoring staff member, and timestamp. The module supports filtering comments by the associated student.

3.2.6 Demos and Status API Module

The Demos and Status API module tracks demo checkpoint grading. A Demo represents a named checkpoint (e.g., "Demo 1"), and each demo has a set of StatusCategory records representing graded dimensions (e.g., Teamwork, Coding). Status values (Blue, Green, Yellow, Red) are stored separately for individual students (StudentDemoStatus) and for groups (GroupDemoStatus), with the group-level record also supporting a comments field. This module exposes ViewSets for demos, status categories, student statuses, and group statuses.

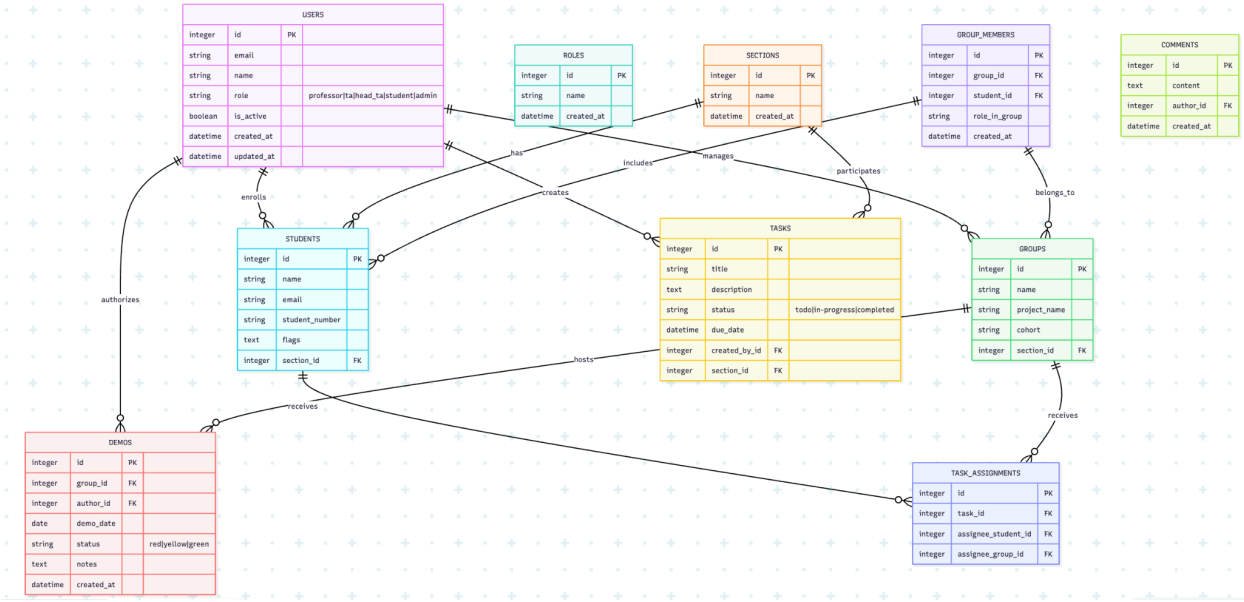
3.2.7 Calendar Events API Module

The Calendar Events API module manages scheduled events created by staff. Events include a name, date, start and end times, description, and optional associations with a group and a staff member. Records are returned ordered by date and start time.

3.2.8 Sections and Semesters API Module

The Sections and Semesters API module organizes the hierarchical structure of the course. A Semester represents an offering of the course (e.g., "SP26") and maintains a many-to-many association with staff members. A Section belongs to a semester and groups students into organizational units. All other entities in the system (groups, tasks, events) are scoped to a section, making semester and section selection the top-level filter for most data in the application.

4 Data Decomposition



The 3090 Dashboard stores persistent application data in a relational database. The data is organized around the main course-management entities used by professors, TAs, and students. Each entity represents a core piece of information needed to manage sections, groups, tasks, student progress, and comments. The database is structured so that related records can be linked together and queried efficiently by role and by section.

The main persistent data stored by the system includes:

- Users: Stores account information for all authenticated users, including their name, email, role, and account status.
- Roles: Represents the type of access a user has in the system, such as professor, TA, head TA, student, or admin.
- Sections: Stores course section information such as section name, semester, and course code.
- Students: Stores student records, including identifying information, flags, and the section they belong to.
- Groups: Stores project group information, including group name, project name, and section association.
- Group Members: Stores the relationship between students and groups, since a student may belong to a group and a group may contain multiple students.

- Demos: Stores demo or progress check information for each group, including status and notes from instructors or TAs.
- Tasks: Stores tasks created by staff, including title, description, status, due date, and the section or creator associated with the task.
- Task Assignments: Stores which student or group a task is assigned to.
- Comments: Stores comments written by staff about students, groups, demos, or tasks.
- Attachments: Stores uploaded files or related resources connected to a student, group, demo, task, or comment.
- Attendance: Stores attendance records for students in a section on a given date.

The relationships between these entities are as follows:

- A section contains many students, groups, and tasks.
- A group contains many students through the group membership relationship.
- A task may be assigned to either a student or a group.
- A user can create many tasks, comments, and demos.
- A comment may refer to a student, group, demo, or task.
- An attendance record belongs to one student and one section.

This structure allows the application to keep course information centralized while supporting role-based access and efficient updates. It also makes it possible to track progress over time for groups and students while preserving the historical record of tasks, comments, and attendance.

5 Design Rationale

5.1 Design Issues

While our frontend design and setup was fairly harmonious thanks to detailed screen sketches and the flexibility of our libraries, the backend was an ongoing challenge for a large portion of the semester. It took us a while to decide and get one system running, and later down the line we ran into problems with Django as we all worked as full-stack developers and lacked inter-module synchronicity.

5.2 Backend Setup

5.2.1 Description

When planning out the early stages of our project, we debated different tools for our backend api system. The two options were Springboot, which we were familiar with, or Django, which was unfamiliar but boasted many features.

5.2.2 Factors affecting Issue

Another factor complicating the decision was the remote server. There was confusion around our ability to execute commands and set up any sort of backend remotely. This made it even more difficult to try approaching a new method over SpringBoot.

5.2.3 Alternatives and their pros and cons

The pros of SpringBoot were mainly rooted in its familiarity. SpringBoot had clear documentation and would be easy to set up quickly and get running. It would also hasten the speed of development to work in a familiar environment, but we would be giving up the chance to work with new technologies. Django had a much steeper learning curve and the documentation was not particularly beginner friendly. However, it boasted a number of features that made it very convenient for development. Django comes with a lot of admin permissions so role based permissions would be easy to set up. Views allow developers to curate what information gets sent back through query sets, making for a more-lightweight backend than writing API endpoints for every type of information needed.

5.2.4 Resolution of Issue

In the end, our group wanted to prioritize learning new technologies, and Django's built in features made it a more convincing choice. Lottie took charge on setting up the backend on the server and locally, and we were able to get developing from there.

5.3 Migration Conflicts

5.3. Description

The initial weeks of development on Django were fairly smooth. However, as we had all taken on full-stack development roles, the team began running into migration conflicts as we created merge requests. Certain parts of the backend had to exist before others could use them as foreign keys, and a pattern of dependencies needed to be established.

5.3.2 Factors affecting Issue

Working as full-stack developers meant that we had a lack of dedicated developers for the backend. When a conflict arose, we would band-aid a solution in order to keep pushing forward, and the tech debt kept accumulating. Additionally, while we had an ER-Diagram plan for what apps were needed and how they would interact, there were still some decisions that needed to be made on design.

5.3.3 Resolution of Issue

After we finalized different backend decisions as a group, Lottie translated that into different Django apps that were interlinked accordingly. It was easiest to have one developer actually making the changes, so that they were all consistent and functional with each other. From there, we all fixed any front-end bugs that arose after the update. The result was a cohesive architecture that enabled smooth future development.