Software Engineering Project

Sep 25 - Team 38

Milestone 3

Team Members

Name	Roll Number
Treasa Janet	21F1002081
Palash Johri	21F1000512
Raghavendra Narayan Jha	21F1003534
Prabhu	21F2001015
Rishav Kumar	22F3001352
Sohini Sarkar	21F2001369
Omkar Shankar Pawar	23F1001898

Table of Contents

Project Schedule

Task Distribution

The entire project was divided into SDLC phases: Planning, Design, Development, Testing, and QA/Implementation. Each milestone was set for a 2–3 week duration, with the aim to complete and submit each milestone document within a single sprint.

The entire project is divided into 6 sprints, each aligned with the milestone deadlines:

- Milestone 1: 22 October, 2025
- Milestone 2: 22 October, 2025
- Milestone 3: 05 November, 2025
- Milestone 4: 16 November, 2025
- Milestone 5: 30 November, 2025
- Milestone 6: 07 December, 2025

Planning sprints

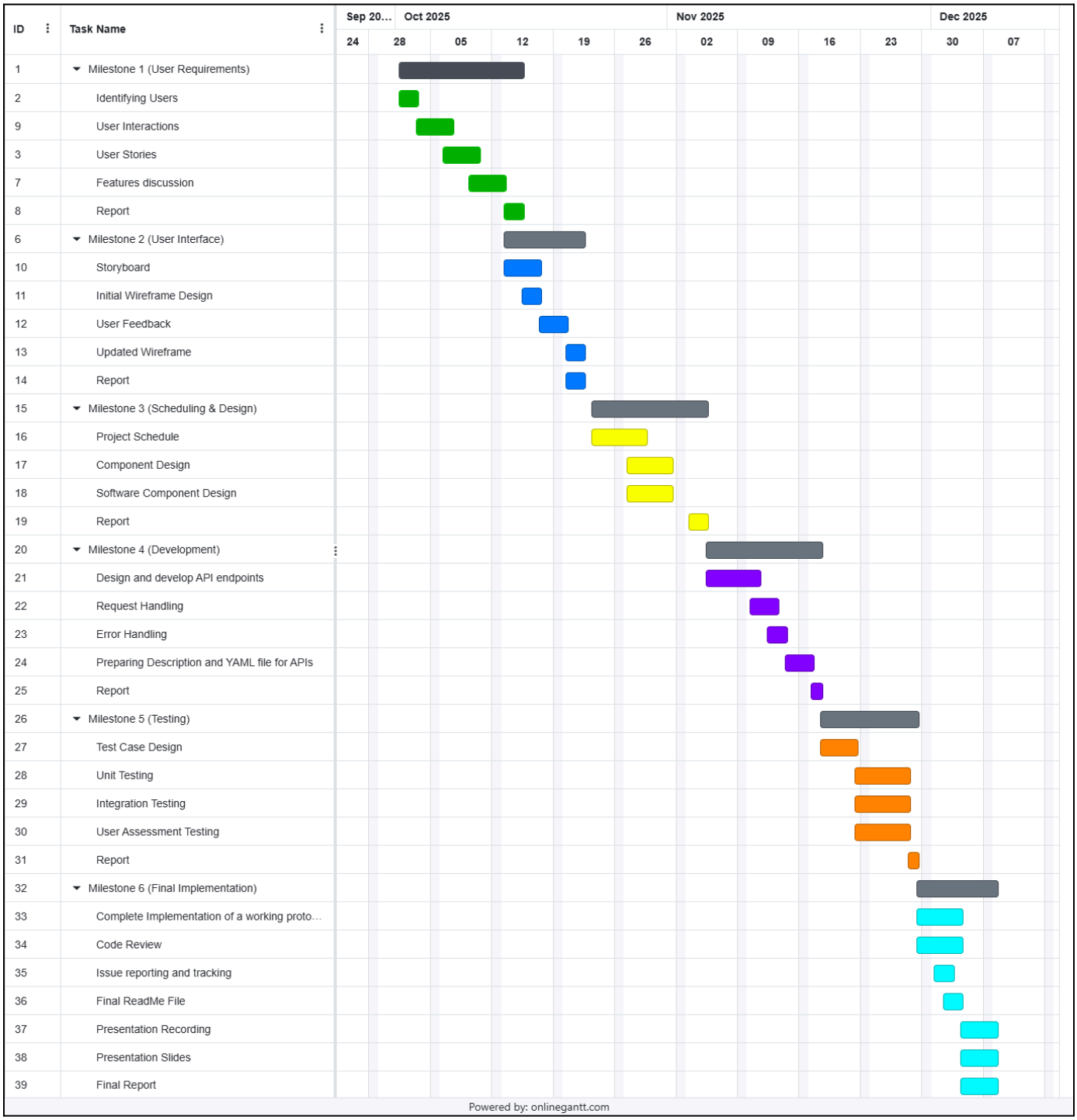
Milestone 1: User Identification & User Stories
Milestone 2: UI & Storyboarding
Milestone 3: Scheduling & Design

Implementation sprints

Milestone 4: API Development
Milestone 5: Testing & Integration
Milestone 6: Final Submission

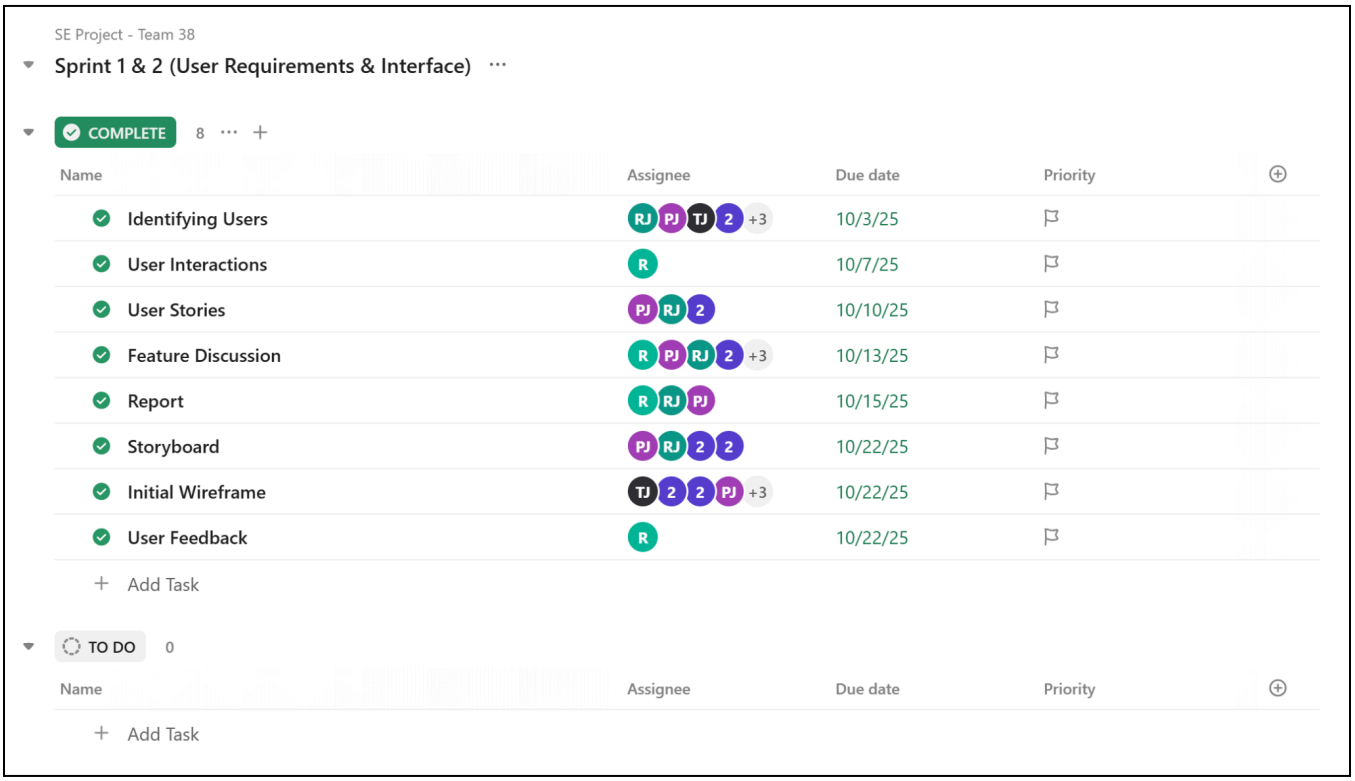
Gantt chart

Monthly Timeline view (click here for clear view)

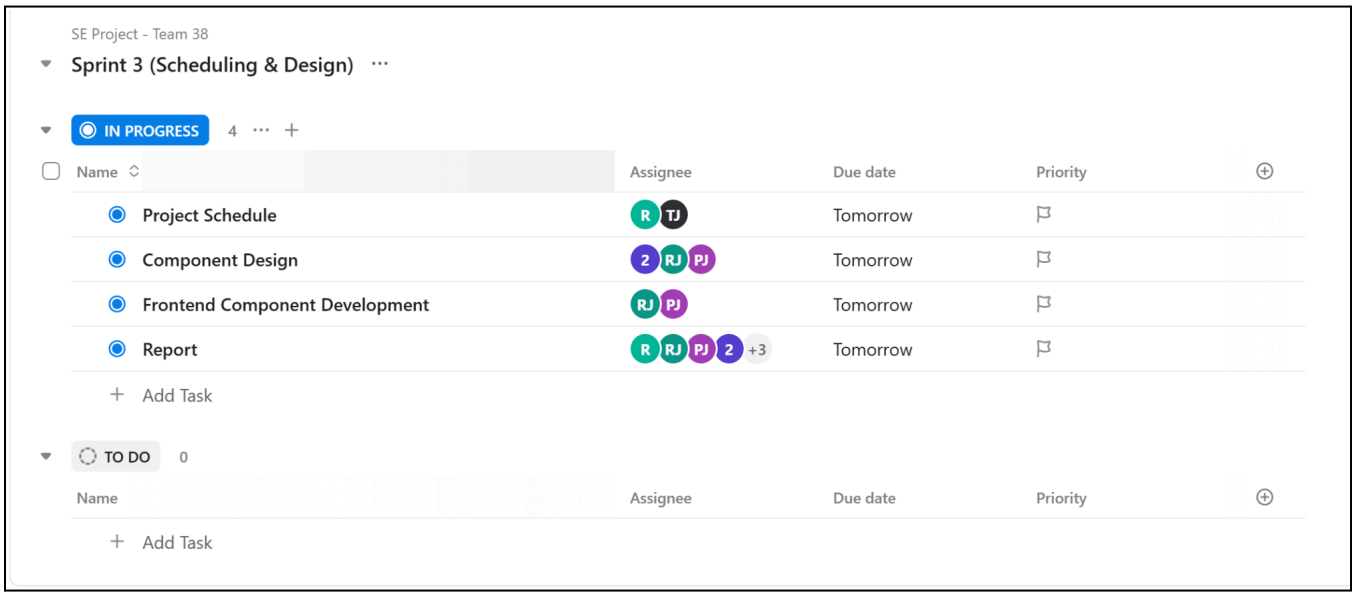


Sprint-wise Task Backlog

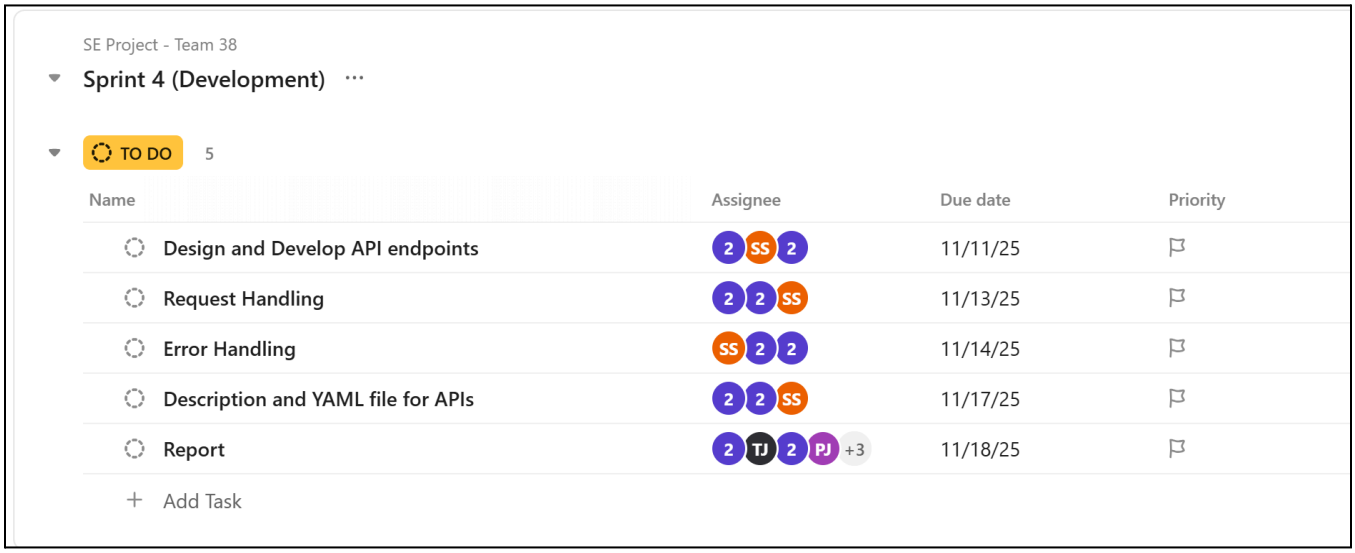
Sprint 1 & 2 (User Requirements & Interface):



Sprint 3 (Scheduling & Design):



Sprint 4 (Development):



SE Project - Team 38

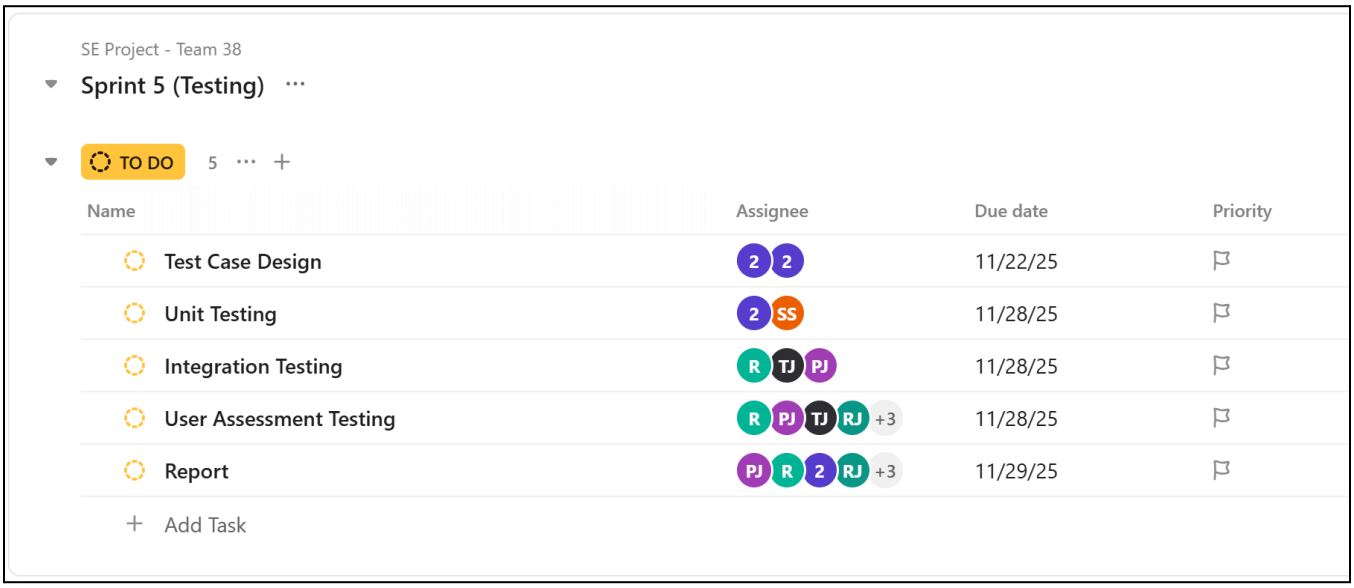
Sprint 4 (Development)

TO DO 5

Name	Assignee	Due date	Priority
Design and Develop API endpoints	2 SS 2	11/11/25	
Request Handling	2 2 SS	11/13/25	
Error Handling	SS 2 2	11/14/25	
Description and YAML file for APIs	2 2 SS	11/17/25	
Report	2 TJ 2 PJ +3	11/18/25	

Add Task

Sprint 5 (Testing):



SE Project - Team 38

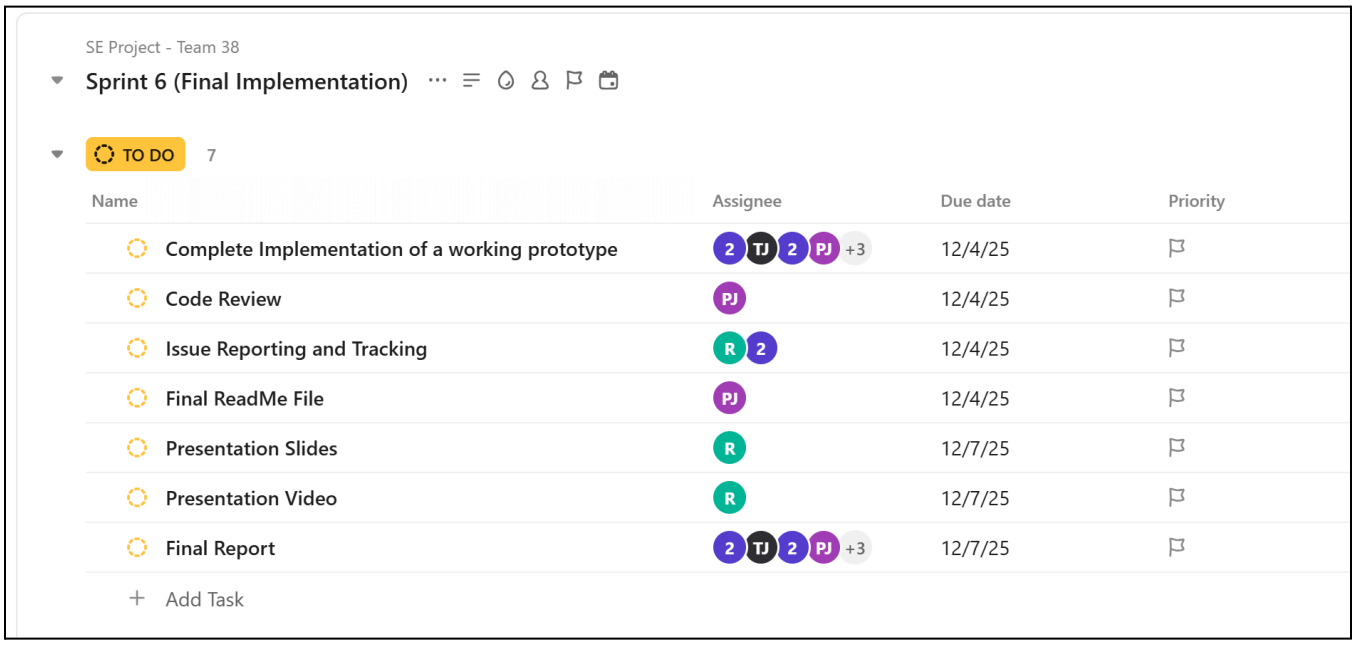
Sprint 5 (Testing)

TO DO 5

Name	Assignee	Due date	Priority
Test Case Design	2 2	11/22/25	
Unit Testing	2 SS	11/28/25	
Integration Testing	R TJ PJ	11/28/25	
User Assessment Testing	R PJ TJ RJ +3	11/28/25	
Report	PJ R 2 RJ +3	11/29/25	

Add Task

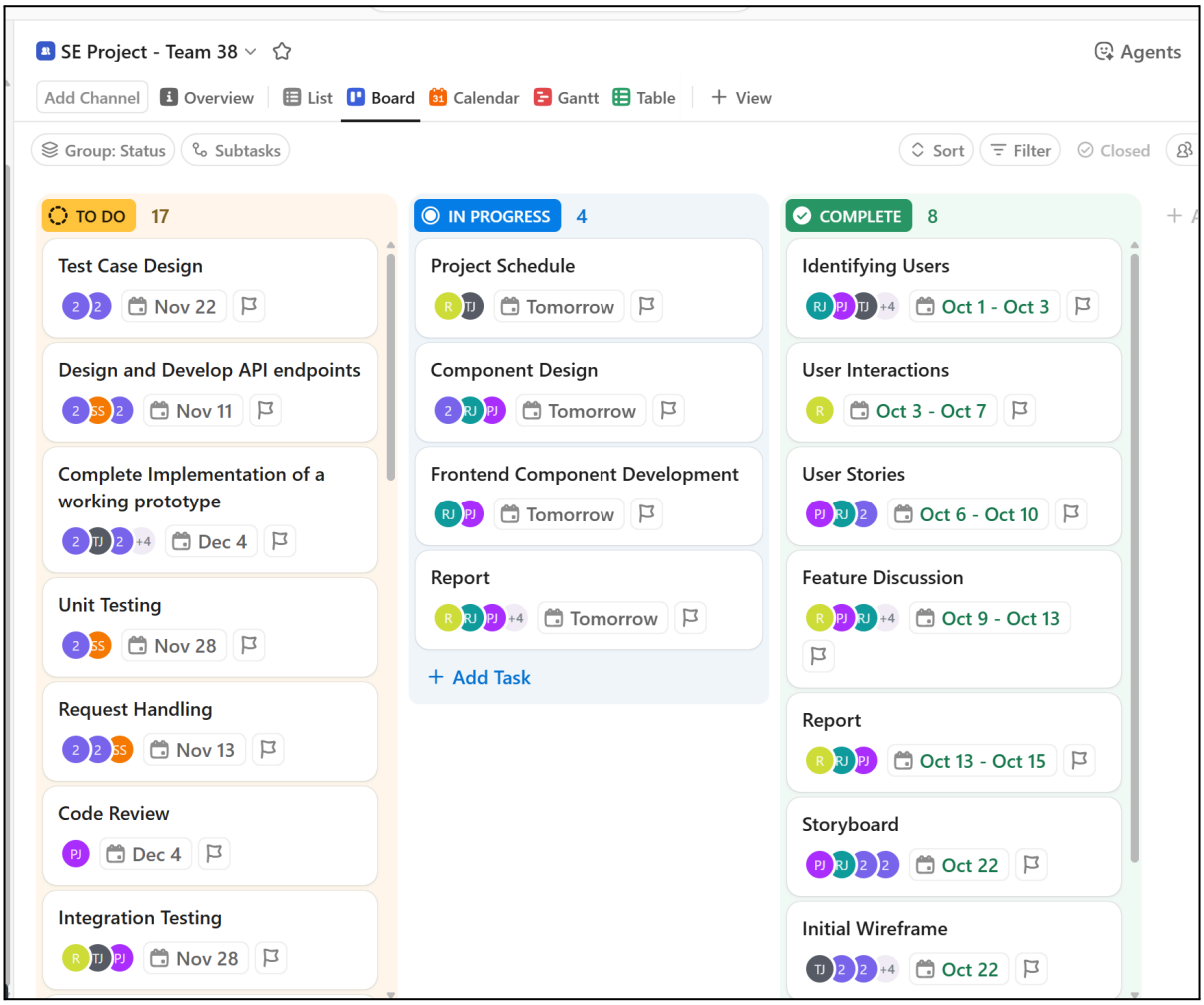
Sprint 6 (Final Implementation):



This structure ensures all tasks are visible, progress is tracked, and team members are accountable for their assignments throughout each sprint, in line with agile best practices.

Scrum Board

The Scrum board tracks each task in the backlog as it moves from ‘To Do’ status through ‘In Progress’ and finally to ‘Complete’ status.



Scrum Meetings

Scrum Meetings Minutes

Below is the meeting minutes of some of the scrum meetings conducted for the project, formatted for clarity and alignment with standard software engineering documentation practices:

Scrum Meeting Schedule:

- **Frequency:** Every **Monday**, **Wednesday** and **Friday**, **9:00 – 9:30 PM**

Minutes of Meeting:

✦ Scrum Meeting | Oct 8, 2025

Attendees: RISHAV, PALASH JOHRI, RAGHAVENDRA NARAYAN JHA, Prabhu, OMKAR SHANKAR PAWAR, TREASA JANET

Main Points Discussed:

1. **Interview Insights:**
    - Rishav shared findings from his interview with a Teaching Assistant (TA).
    - Students often rely on external sources for course-related information due to:
        - Language barriers
        - Lack of engagement with official communication channels
2. **Focus on Target Users:**
    - Palash emphasized the importance of prioritizing Teaching Assistants (TAs) as key users for further data collection.
    - Suggested using synthetic data to model and understand TA-specific challenges more effectively.
3. **Regional Language Support:**
    - The team discussed the need for regional language integration in communication tools to enhance accessibility and inclusiveness.
4. **User Research and Outreach:**
    - The team agreed to conduct additional client and user interviews to refine user stories and validate assumptions.
    - Plans were made to reach out to course instructors and Bharti (Operations Head) for deeper operational insights.

5. **Contact Strategy for Bharti ma'am:**
    - Palash proposed that two team members draft a professional outreach email.
    - Omkar volunteered to lead the email drafting effort.
6. **Project Management and Coordination:**
    - Rishav will set up project management tools to track progress and milestones.
    - The team will maintain a shared Excel sheet to log outreach activities and avoid duplication.
7. **External Connections:**
    - Raghavendra will reach out to his contacts in the SPG group for potential interviews and share updates with the team.

Action Items:

1. Draft outreach email for Bharti ma'am and course instructors — Omkar (lead), Palash (support)
2. Set up project management tools for progress tracking — Rishav
3. Create and maintain shared Excel sheet for documenting outreach activities — Treasa
4. Conduct additional client and TA interviews to refine user stories — All team members
5. Reach out for potential interviews — Raghavendra
6. Explore integration of regional language support in communication tools — Palash and team

✦ Scrum Meeting | Oct 13, 2025

Attendees: RISHAV, PALASH JOHRI, OMKAR SHANKAR PAWAR, Prabhu, RAGHAVENDRA NARAYAN JHA, SOHINI SARKAR, TREASA JANET

Main points discussed:

1. Updates on Project Deliverables and Instructor Outreach
    a. Raghavendra shared his ongoing research on Lang Graph and Lang Chain.
    b. Omkar reported Bharthi ma'am's suggestion to involve TAs in project insights.
    c. Treasa outlined project deliverables for Milestones 1 and 2, confirming ongoing interviews.
    d. Discussion included role allocation and progress tracking.
2. User Interviews and Project Milestones Discussion
    a. Rishav mentioned completion of three TA interviews; instructor interviews are pending.
    b. Omkar suggested involving students to capture diverse user perspectives.
    c. Treasa emphasized setting strict interview deadlines aligned with milestones.
    d. Palash encouraged collaboration for developing user stories and wireframes using interview findings.
3. Client Interview Preparation and Feature Proposals
    a. Palash discussed preparation for the upcoming client interview with Nidhish.
    b. Proposed integrating multimodal models as new project features.
    c. Raghavendra highlighted workflow enhancements for project clarity.
    d. Sohini volunteered to contribute via Figma prototypes.
    e. Palash coordinated responsibilities for storyboard and wireframe development.

Action items:

1. Reach out to instructors and request interviews - Omkar
2. Schedule a meeting for storyboard & wireframe planning - Palash Johri (with Raghavendra & Rishav)
3. Write user stories based on completed TA interviews. - Raghavendra Narayan Jha & Rishav
4. Summarize key pain points and reasoning for user stories - Rishav
5. Maintain Excel sheet with interview status and tracking - Treasa Janet

✦ Scrum Meeting | Oct 18, 2025

Attendees: NIDHISH , RISHAV , PALASH JOHRI , OMKAR SHANKAR PAWAR , Prabhu , RAGHAVENDRA NARAYAN JHA , SOHINI SARKAR , TREASA JANET

Main points discussed:

1. Palash reviewed the progress of the design and storyboard, inviting inputs from the team.
2. Nidhish advised categorizing features into top-level feature statements to simplify presentation and enhance clarity.
3. Emphasis on including login pages and user onboarding screens as part of the design.
4. Discussion on timeline adjustments, with a new deadline of October 22 for finalizing wireframes.
5. The team decided to share a comprehensive documentation package by Monday, integrating feedback and final deliverables.

Action items:

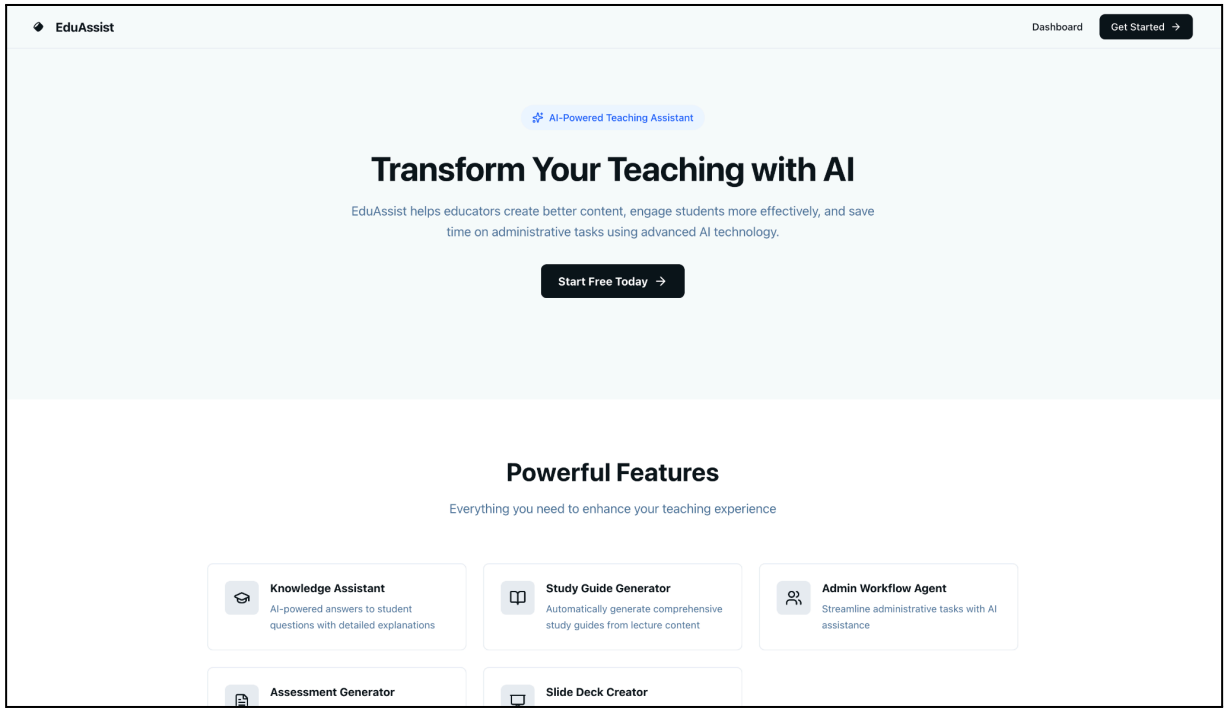
1. Share Figma link and storyboard PPT with the team for feedback.
2. Add login pages and necessary updates to the prototype.
3. Prepare the feedback video after wireframe finalization.
4. Create storyboard aligning with milestone one deliverables.
5. Compile and share the completed document with Nidhish.

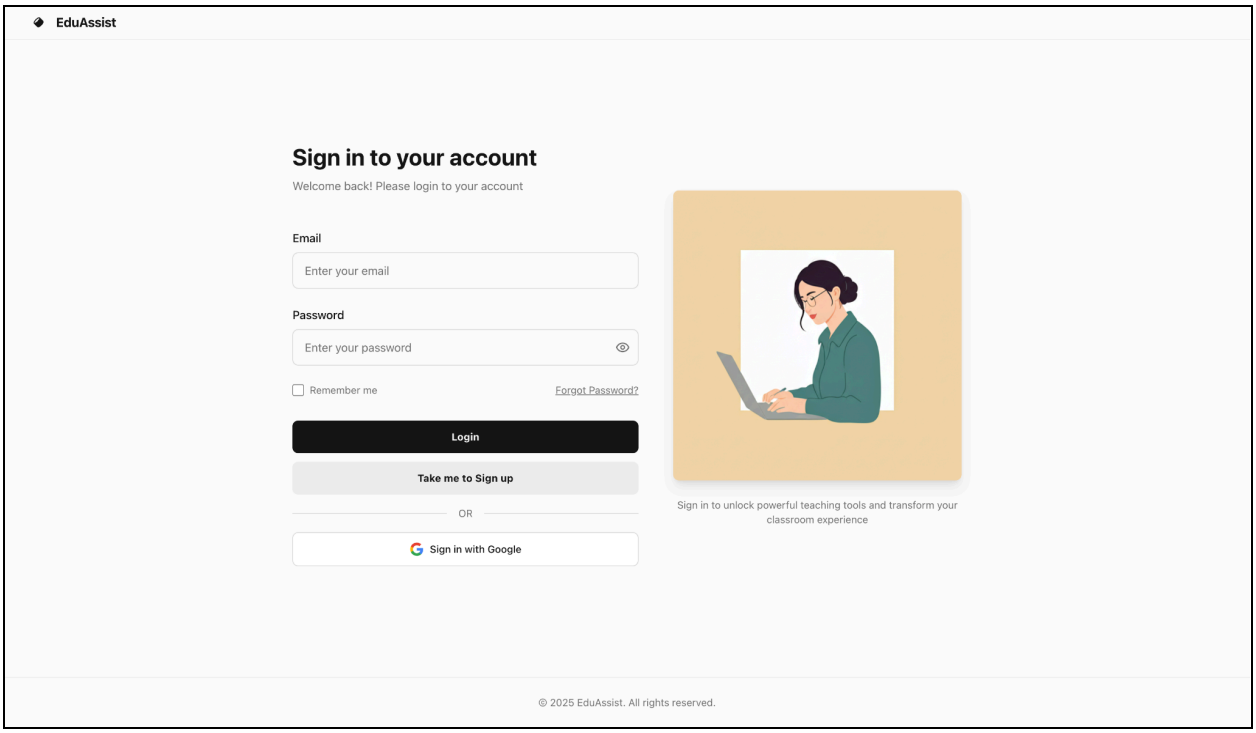
Proof of Interaction

Software Design

Frontend

Landing & Authentication





EduAssist

Reset your password

Enter your email address and we'll send you a verification code

Email Address

Enter your email

Send Verification Code →

Remember your password? Back to login

We'll help you recover your account with a simple verification process

EduAssist

Create your account

Name

Enter your name

Email

Enter your email

Password

Enter your password

Course Code

Enter course code

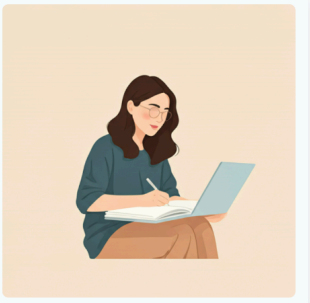
TA Role Verification

Select role

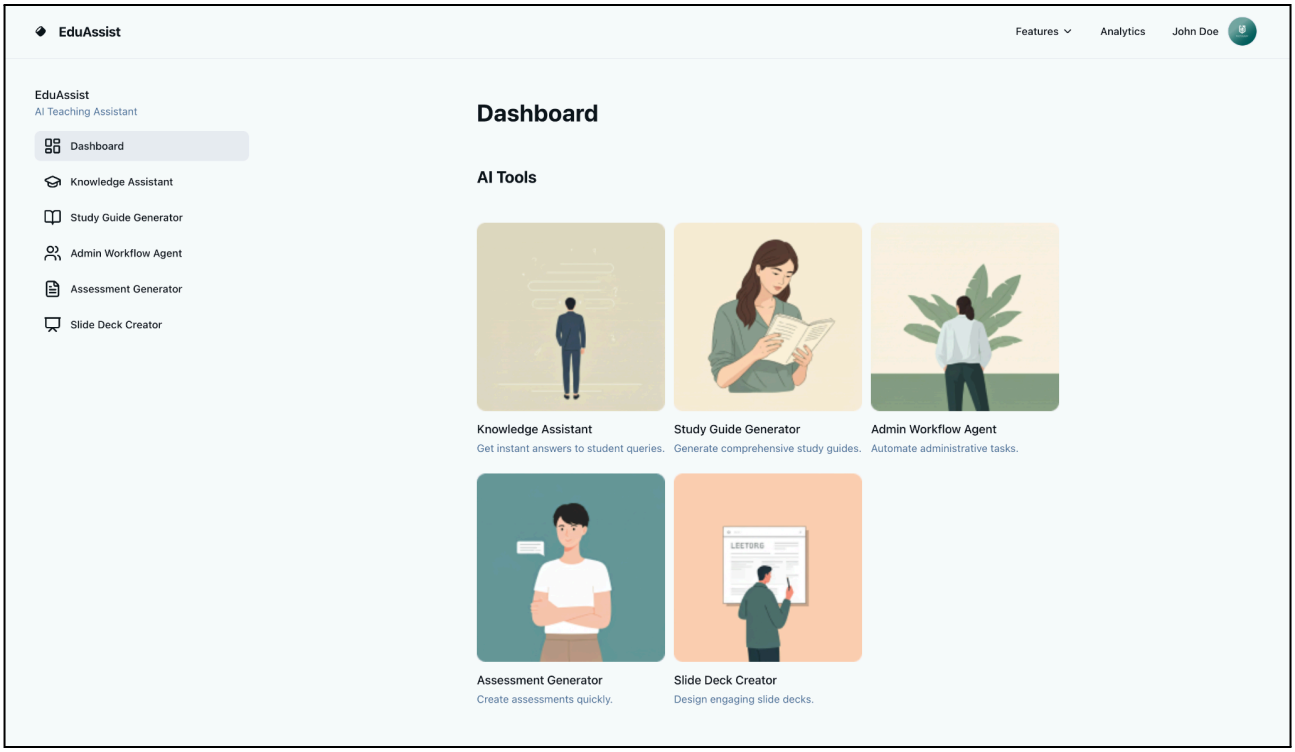
I agree to the Terms & Conditions and Privacy Policy

Create Account

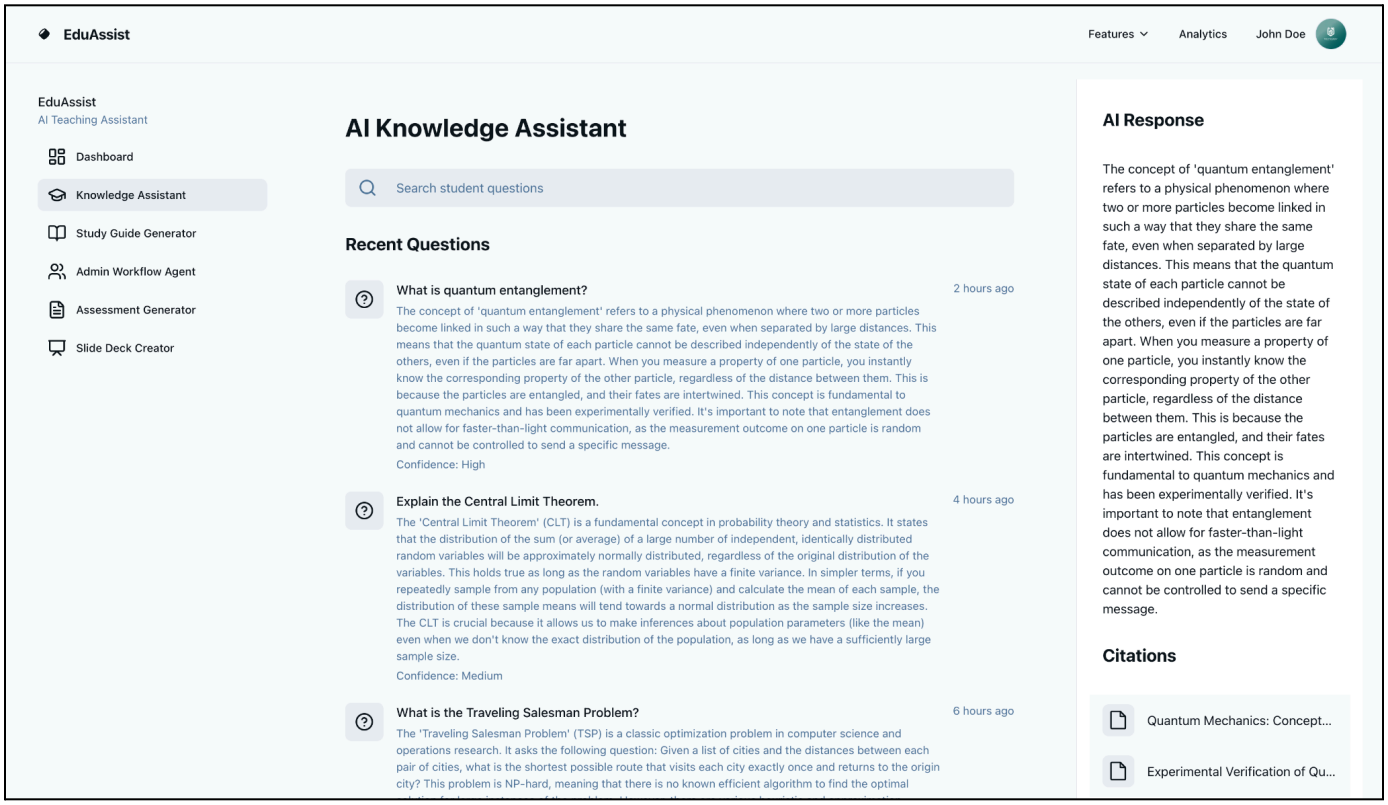
Already have an account? Login



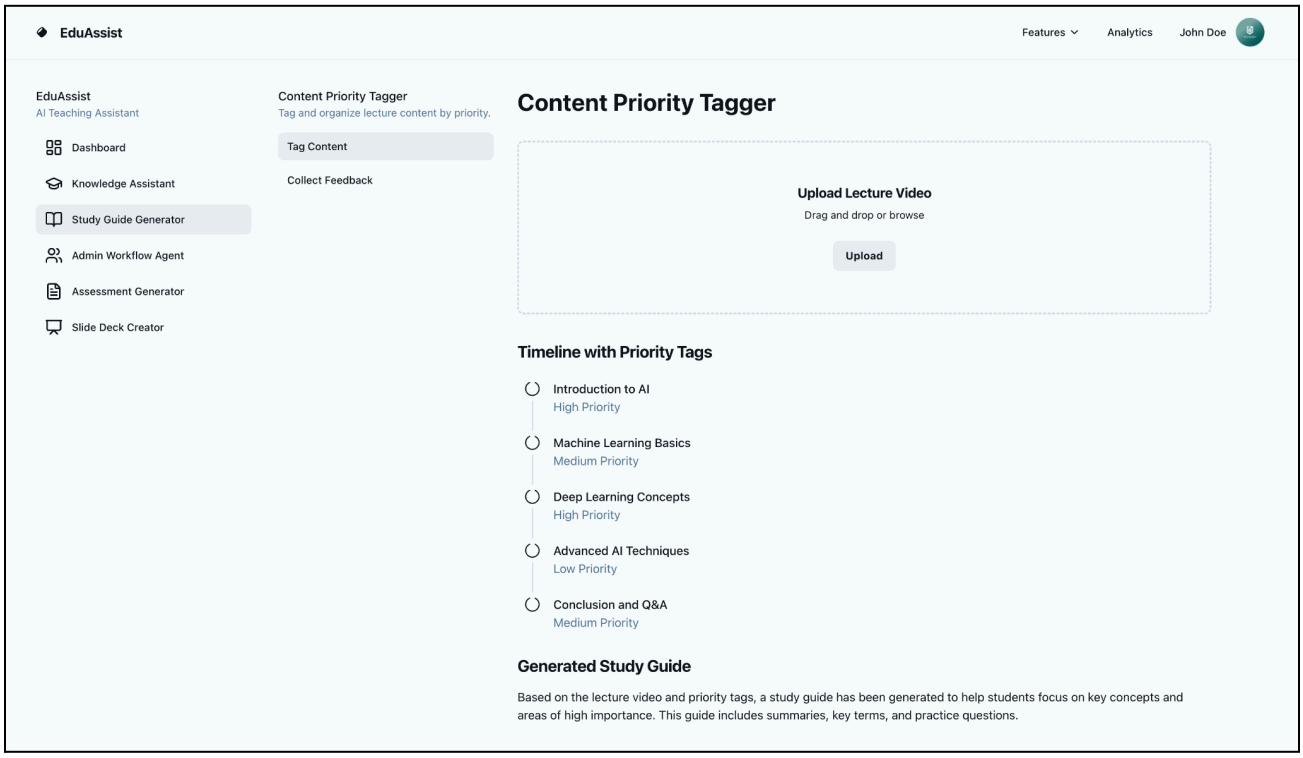
Join thousands of educators and TAs using EduAssist to enhance their teaching



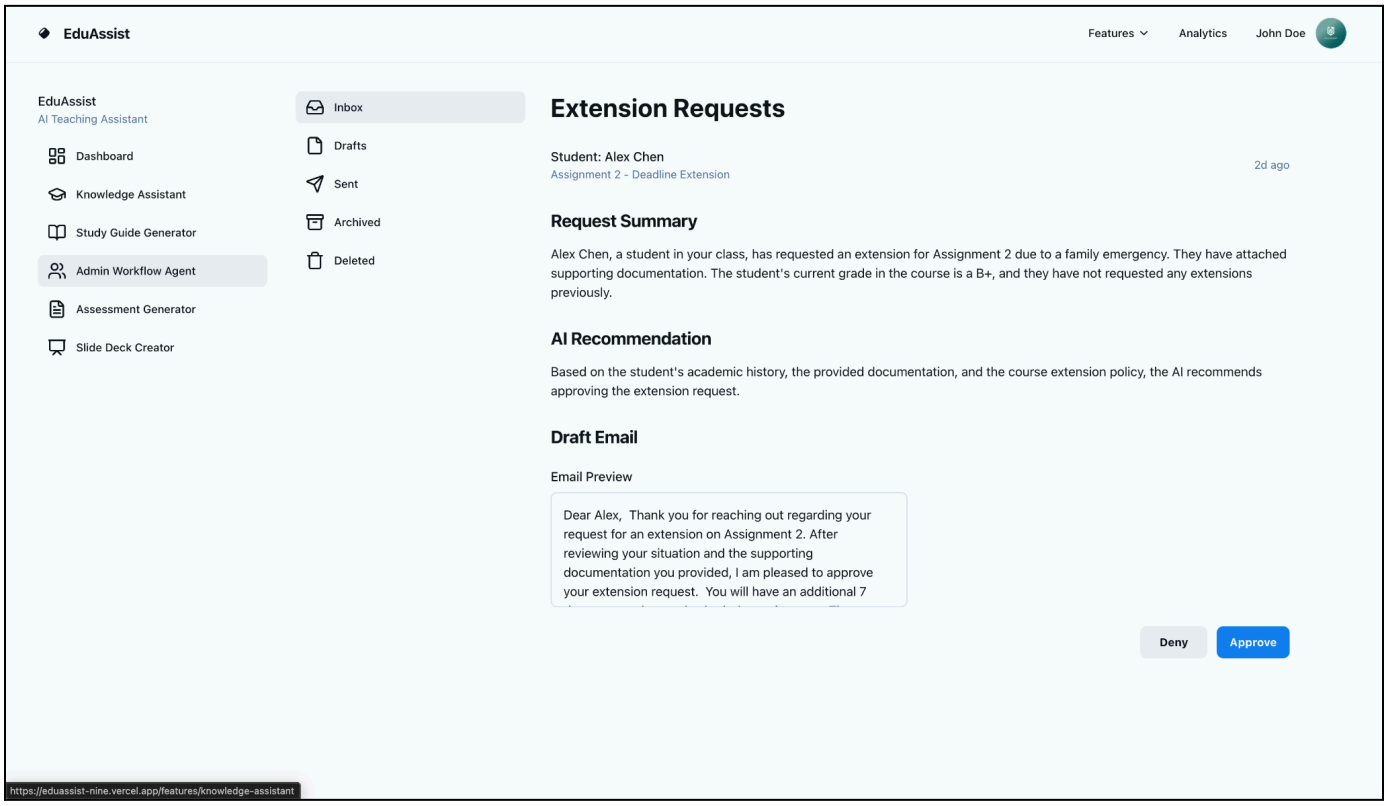
Knowledge Assistant



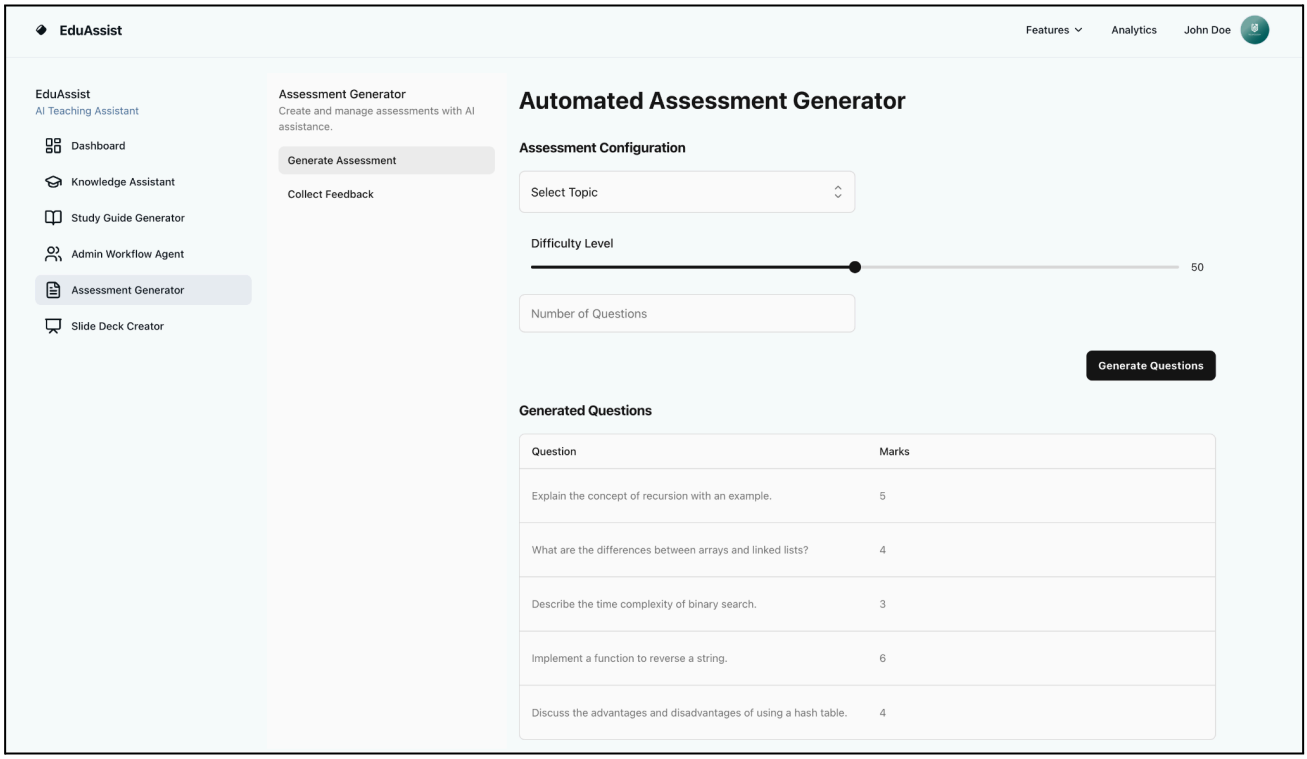
Study Guide Generator



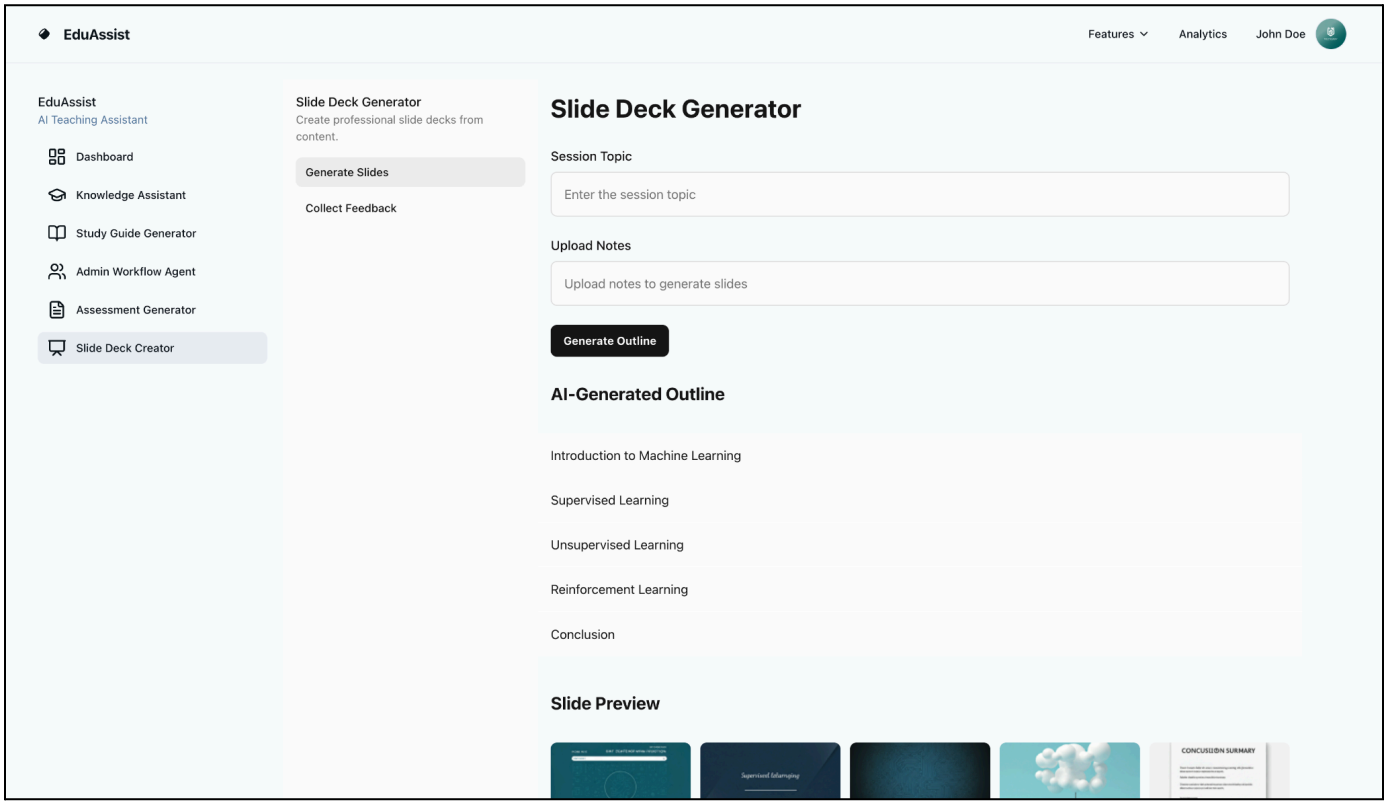
Admin Workflow Agent



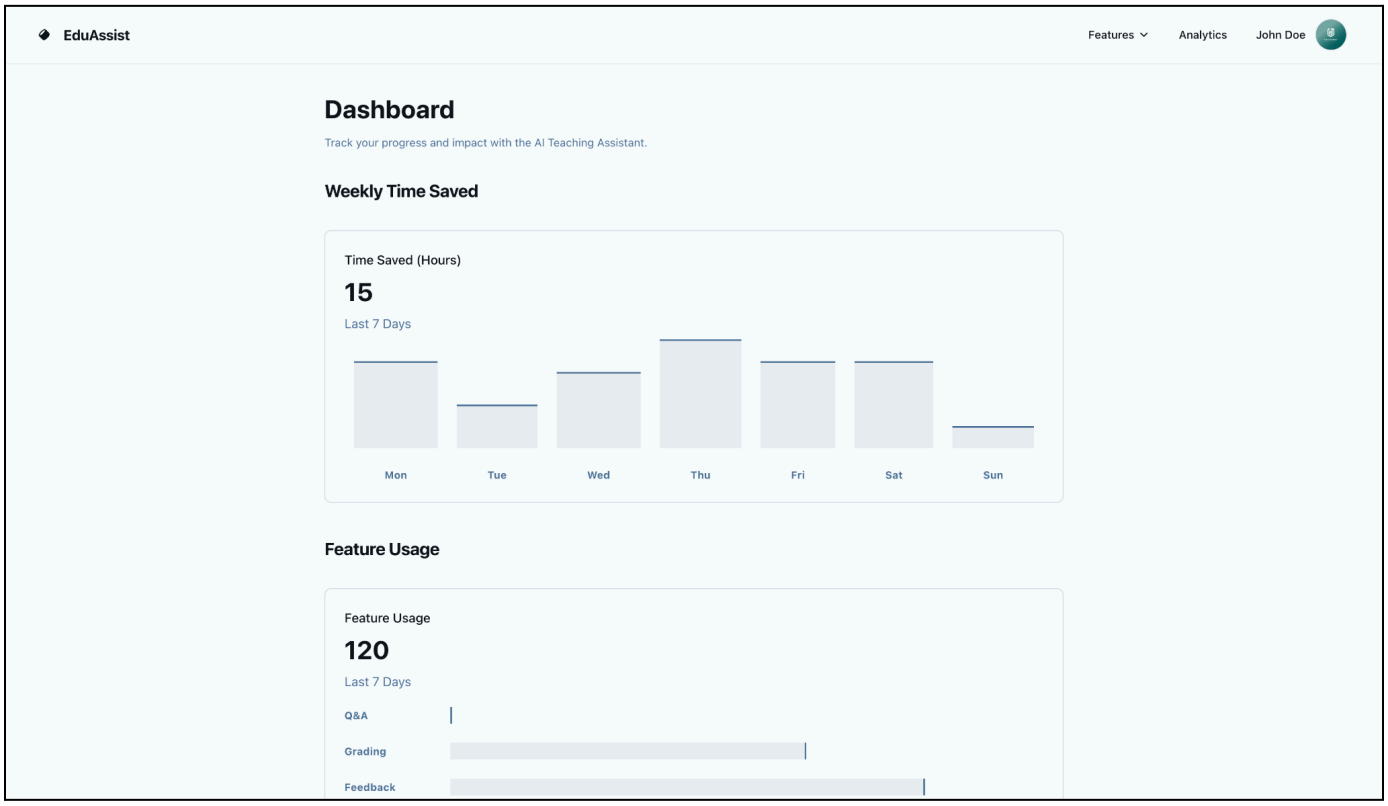
Assessment Generator



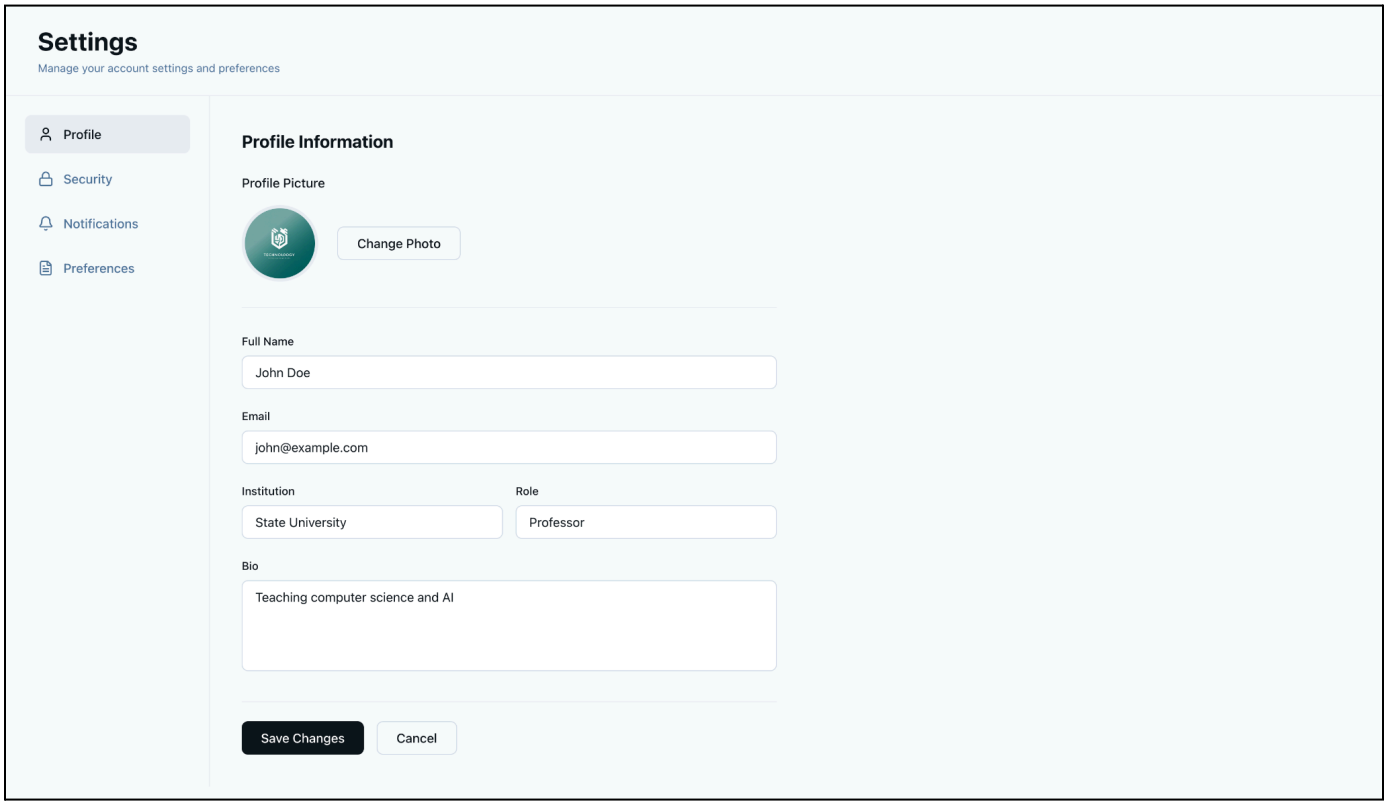
Slide Deck Creator



Analytics



User Settings



SYSTEM DESIGN DOCUMENTATION

FRONTEND DESIGN

1. Programming Language & Framework

TypeScript 5.3+ - Type-safe development with compile-time checking - Enhanced IDE support and code maintainability - Interface definitions for API contracts

Next.js 14 (App Router) - Modern React framework with server-side rendering - File-based routing and automatic code splitting - Vercel integration for seamless deployment

2. Architecture Overview

Component-based architecture with clear separation: - **Pages Layer:** Authentication, Dashboard, Feature pages - **Components Layer:** Reusable UI components (layout, forms, widgets) - **Services Layer:** API client, authentication utilities, type definitions - **Hooks Layer:** Custom React hooks for data fetching and state management

3. UI Components & Styling

shadcn/ui Component Library - Pre-built accessible components (Button, Card, Dialog, Form, Input, Select) - Built on Radix UI + Tailwind CSS - Dark mode support and WCAG 2.1 compliance

Tailwind CSS 3.3+ - Utility-first CSS framework for rapid development - Responsive breakpoints (mobile-first) - Custom theme configuration

4. Data Management

Axios HTTP Client - Request/response interceptors for auth tokens - Automatic error handling and retry logic - JSON transformation

State Management - React Hooks (useState, useContext, useReducer) - Custom hooks for data fetching (useAPI, useChat, useAssessment) - Context API for global state (authentication, user profile)

5. Key Frontend Features

- Responsive design (mobile, tablet, desktop)
- Real-time chat interface with typing indicators
- Document drag-and-drop upload
- Interactive charts and analytics
- Form validation with instant feedback
- Loading states and error boundaries

BACKEND DESIGN

1. Programming Language & Framework

Python 3.10+ - Rich ecosystem for GenAI integration - Strong typing with Pydantic - Async/await for concurrent operations

FastAPI 0.104+ - Modern async web framework - Automatic OpenAPI/Swagger documentation - Built-in request validation - High performance (async-first)

Uvicorn ASGI Server - Production-ready performance - WebSocket support for streaming - Auto-reload in development

2. Architecture Overview

Layered Modular Monolith: - **API Layer:** Route handlers and request validation - **Service Layer:** Business logic and GenAI orchestration - **Repository Layer:** Database CRUD operations - **Model Layer:** ORM entities and relationships

Clear separation ensures testability and maintainability.

3. Database & ORM

PostgreSQL 15+ - Relational database with ACID properties - Hosted on Neon (free cloud PostgreSQL) - Full-text search and JSON support

SQLAlchemy 2.0+ - Type-safe ORM with async queries - Relationship management and lazy loading - Alembic for database migrations

Core Data Models: - User hierarchy (User, TeachingAssistant, Student) - Course and content (Course, Document, Lecture) - Conversations (Conversation, Message) - Assessments (Assessment, Question) - Administrative (ExtensionRequest, Assignment) - Feedback (Feedback, FeedbackAnalytics)

4. Authentication & Authorization

JWT Token-Based Authentication - Access token (30 min expiry) - Refresh token (7 day expiry) - Token stored securely (localStorage + HTTP-only cookies)

Role-Based Access Control (RBAC) - Roles: TA, Student, Admin - Permission enforcement at endpoint level - Resource-level authorization

Security Measures - bcrypt password hashing (12 salt rounds) - Token signature verification - CORS configuration for allowed origins

5. GenAI Integration

Gemini API (Google) - Models: gemini-2.5-pro (primary), gemini-2.0-flash (fast) - Free tier: 15 RPM, 1.5M tokens/day - Multi-modal capabilities (text, images) - Streaming response support

LangChain Framework - LLM orchestration for complex workflows - Agentic reasoning with tool use - Prompt template management - Conversation memory

ChromaDB (Vector Database) - Semantic similarity search - Document embedding storage - Collection management per course - In-memory or persistent mode

6. Key Backend Features

- Async request handling for high concurrency
- Structured logging for debugging
- Rate limiting to prevent abuse
- Comprehensive error handling
- Database connection pooling
- Environment-based configuration

DESIGN COMPONENTS & INTERACTIONS

This section details the major system components and how they interact.

Component 1: Authentication System

Purpose: Secure user authentication and session management

Frontend Components: - LoginForm: Email/password input with validation - SignupForm: Registration with role selection - AuthContext: Global authentication state management

Backend Components: - AuthService: Credential validation and token generation - UserRepository: Database operations for user data - Security utilities: Password hashing and token verification

Interaction Flow:

```
User enters credentials in LoginForm
    ↓
Frontend validates input format
    ↓
Axios sends POST request to /auth/login
    ↓
Backend AuthService validates credentials
    ↓
UserRepository queries database for user
    ↓
Password hash verified with bcrypt
    ↓
JWT tokens generated (access + refresh)
    ↓
Tokens returned to frontend
    ↓
Frontend stores tokens securely
    ↓
Axios interceptor adds token to all future requests
```

```
    ↓
User redirected to dashboard
```

Key Interactions: - Frontend ↔ Backend: HTTP requests with JSON payloads - Backend ↔ Database: SQL queries via SQLAlchemy ORM - Token validation on every protected API call

---

Component 2: Knowledge Assistant (RAG Chatbot)

Purpose: AI-powered Q&A system with document retrieval

Frontend Components: - ChatInterface: Real-time message display and input - DocumentUpload: Drag-and-drop file upload with progress - ConversationList: History of past conversations - ConfidenceIndicator: Visual display of AI confidence

Backend Components: - KnowledgeAssistantService: Query processing orchestration - RAGService: Document processing and retrieval - GeminiClient: LLM integration for embeddings and completions - VectorStoreRepository: ChromaDB operations - ConversationRepository: Chat history persistence

Interaction Flow (Document Upload):

```
TA uploads course PDF via DocumentUpload
    ↓
Frontend sends file via multipart form
    ↓
Backend saves file to storage
    ↓
RAGService extracts text from PDF
    ↓
Text split into 512-token chunks with overlap
    ↓
GeminiClient generates embeddings for each chunk
    ↓
Embeddings stored in ChromaDB collection
    ↓
Document marked as processed in database
    ↓
Frontend shows "Processed" status
```

Interaction Flow (Student Query):

```
Student types question in ChatInterface
    ↓
Frontend sends message to backend
    ↓
KnowledgeAssistantService receives query
    ↓
Query converted to embedding via Gemini
    ↓
VectorStore performs similarity search
```

```
    ↓
Top 5 relevant document chunks retrieved
    ↓
Context + query sent to Gemini for completion
    ↓
LLM generates answer with citations
    ↓
Confidence score calculated based on context match
    ↓
Message saved to database with sources
    ↓
Response returned to frontend
    ↓
ChatInterface displays message with confidence badge
    ↓
If confidence < 60%: Show "Escalate to TA" option
```

Key Interactions: - Frontend ↔ Backend: REST API for messages and documents - Backend ↔ Gemini API: Embeddings and text generation - Backend ↔ ChromaDB: Vector similarity search - Backend ↔ PostgreSQL: Conversation and message persistence

---

Component 3: Assessment Generator

Purpose: Automated assessment creation with AI

Frontend Components: - ConfigurationPanel: Topic, difficulty, and question type selection - QuestionReview: Table display of generated questions with edit controls - DifficultySetter: Slider for difficulty calibration - ExportOptions: PDF and LMS export formats

Backend Components: - AssessmentGeneratorService: Question generation orchestration - GeminiClient: LLM for question creation - AssessmentRepository: Database persistence - QuestionBankRepository: Reusable question storage

Interaction Flow:

```
TA selects configuration (topics, difficulty, count)
    ↓
Frontend sends generation request
    ↓
Backend creates Assessment record
    ↓
For each question needed:
    Prompt sent to Gemini with topic and difficulty
    ↓
    Gemini generates question with options/answer
    ↓
    Question quality validated
    ↓
    Difficulty calibrated based on complexity
```

```
    ↓
    Marks allocated based on difficulty
    ↓
    Question saved to database
    ↓
    Question added to reusable bank
    ↓
All questions returned to frontend
    ↓
QuestionReview displays editable table
    ↓
TA can regenerate individual questions
    ↓
TA publishes or exports final assessment
```

Key Interactions: - Frontend ↔ Backend: RESTful assessment CRUD operations - Backend ↔ Gemini API: Dynamic question generation - Backend ↔ Database: Assessment and question persistence - Question reuse: Bank checks for similar questions before generating

---

Component 4: Workflow Automation Agent

Purpose: Intelligent processing of student extension requests

Frontend Components: - RequestInbox: Table of pending requests with filters - AIAnalysis: Display of agent reasoning and recommendation - EmailDraft: Auto-generated email with edit capability - DecisionPanel: Approve/Deny/Request Info buttons

Backend Components: - WorkflowAgentService: Agentic workflow orchestration - LangChain Agent: Multi-step reasoning with tools - GeminiClient: LLM for agent reasoning - ExtensionRequestRepository: Request data access

Agent Tools: - StudentHistoryTool: Retrieves past submission records - PolicyCheckerTool: Verifies compliance with course policies - SimilarCasesTool: Finds comparable past requests

Interaction Flow:

```
Student submits extension request
    ↓
Request stored in database with PENDING status
    ↓
TA opens request in WorkflowAgent interface
    ↓
TA clicks "Analyze" button
    ↓
Frontend requests AI analysis
    ↓
Backend WorkflowAgentService receives request
    ↓
LangChain Agent begins reasoning:
```

```
    ↓
    Step 1: Agent decides to check student history
    Calls StudentHistoryTool
    Tool queries database for submission records
    Returns: on-time count, late count, past extensions
    ↓
    Step 2: Agent decides to verify policy compliance
    Calls PolicyCheckerTool
    Tool performs RAG search on policy documents
    Returns: relevant policy sections and compliance status
    ↓
    Step 3: Agent decides to find similar cases
    Calls SimilarCasesTool
    Tool queries database for similar requests
    Returns: 2-3 comparable cases with decisions
    ↓
    Step 4: Agent synthesizes information
    Gemini generates recommendation with reasoning
    Outputs: APPROVE/DENY/REQUEST_INFO + confidence
    ↓
Analysis saved to database
    ↓
Frontend displays reasoning in AIAnalysis panel
    ↓
TA reviews and makes final decision
    ↓
Email draft auto-generated based on decision
    ↓
TA reviews/edits and sends email
```

Key Interactions: - Frontend ↔ Backend: Request management API - LangChain Agent ↔ Tools: Sequential tool calls with results - Tools ↔ Database/RAG: Data retrieval operations - Backend ↔ Gemini: Agent reasoning and email generation

---

Component 5: Feedback System

Purpose: Collect and analyze user feedback on AI features

Frontend Components: - FeedbackWidget: Thumbs up/down with comment box - TAFeedbackDashboard: Queue of feedback items - AnalyticsView: Charts showing feedback trends

Backend Components: - FeedbackService: Feedback processing and analytics - FeedbackRepository: Database operations - NotificationService: Alert TAs to critical feedback

Interaction Flow:

```
Student receives AI response in chat
    ↓
FeedbackWidget appears below response
```

```
    ↓
Student clicks thumbs down
    ↓
Comment box expands
    ↓
Student types issue and submits
    ↓
Frontend sends feedback to backend
    ↓
FeedbackService creates feedback record
    ↓
If rating < 3: Alert TA via NotificationService
    ↓
Feedback appears in TA dashboard
    ↓
TA reviews feedback and investigates issue
    ↓
TA takes corrective action (fix document, update response)
    ↓
TA marks feedback as reviewed
    ↓
Status updated in database
    ↓
Analytics aggregated for reporting
```

Key Interactions: - Frontend ↔ Backend: Feedback submission and retrieval - Backend ↔ Database: Feedback persistence and queries - Feedback triggers: Notifications sent to TAs for critical issues - Analytics: Periodic aggregation of feedback metrics

---

Component 6: Cross-Cutting Concerns

Purpose: Middleware and shared services

Components:

CORS Middleware - Validates origin of incoming requests - Allows specific frontend domains - Blocks unauthorized cross-origin requests

Authentication Middleware - Extracts JWT token from Authorization header - Verifies token signature and expiry - Injects current user into request context - Returns 401 if invalid token

Rate Limiting Middleware - Tracks requests per IP address - Enforces limits (100 requests/min general, 15 RPM for Gemini) - Returns 429 if limit exceeded

Request Logging - Logs all incoming requests with metadata - Records response status and duration - Stores errors with stack traces

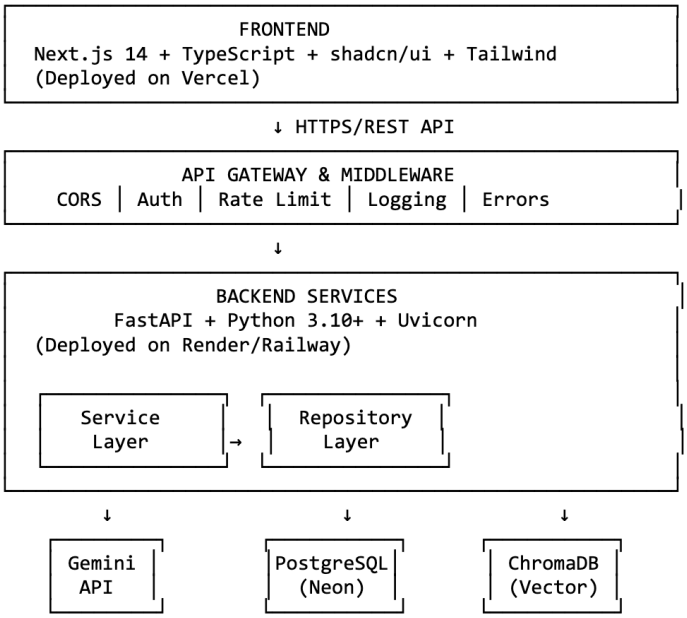
Error Handling - Catches all exceptions globally - Formats errors consistently - Maps exception types to HTTP status codes - Returns structured error responses

Interaction Flow (Protected Request):

```
Frontend makes authenticated request
    ↓
CORS Middleware validates origin
    ↓
Request Logging records incoming request
    ↓
Authentication Middleware extracts token
    ↓
Token verified (signature + expiry)
    ↓
User context injected into request
    ↓
Rate Limiting checks request count
    ↓
Route handler processes request
    ↓
Response returned
    ↓
Response Logging records outcome
    ↓
Error Handler (if error occurred)
```

SYSTEM ARCHITECTURE

High-Level Architecture



Component Communication Patterns

Frontend → Backend: - REST API calls via Axios - JSON request/response format - Bearer token authentication - WebSocket for streaming (future)

Backend → Gemini API: - HTTP POST requests - JSON payload with prompts - API key authentication - Rate limit: 15 requests/minute

Backend → PostgreSQL: - SQL queries via SQLAlchemy ORM - Connection pooling (5-10 connections) - Async queries for performance - Transaction management

Backend → ChromaDB: - Python client library - Collection-based organization - Vector similarity queries - In-memory or persistent storage

Data Flow Examples

User Authentication: Frontend → Backend → Database → Backend → Frontend

RAG Query: Frontend → Backend → ChromaDB (similarity) → Gemini (completion) → Database (save) → Backend → Frontend

Assessment Generation: Frontend → Backend → Gemini (generate questions) → Database (save) → Backend → Frontend

Workflow Analysis: Frontend → Backend → LangChain Agent → Multiple Tools (DB/RAG queries) → Gemini (reasoning) → Database (save) → Backend → Frontend

Security Layers

Layer 1: Network - HTTPS/TLS encryption - CORS protection - Rate limiting

Layer 2: Authentication - JWT token verification - Token expiry enforcement - Refresh token rotation

Layer 3: Authorization - Role-based access control - Resource-level permissions - Endpoint protection

Layer 4: Data - Input validation (Pydantic) - SQL injection prevention (ORM) - XSS protection (output sanitization) - Prompt injection prevention

Scalability Considerations

Horizontal Scaling: - Stateless backend (JWT tokens) - Database connection pooling - Async operations for concurrency

Performance Optimization: - Response caching (future: Redis) - Database query optimization - Vector search indexing - Lazy loading of relationships

Monitoring & Observability: - Structured logging (JSON format) - Error tracking and alerts - Performance metrics - API usage analytics

---

SUMMARY

EduAssist is a modern full-stack application that seamlessly integrates frontend and backend components to deliver AI-powered teaching assistant automation.

Frontend provides an intuitive interface built with Next.js and TypeScript, offering responsive design, real-time interactions, and comprehensive form handling. Components communicate with the backend via a centralized Axios client with built-in authentication and error handling.

Backend implements a clean layered architecture using FastAPI, separating concerns across API, service, repository, and model layers. This enables easy testing, maintenance, and future enhancements.

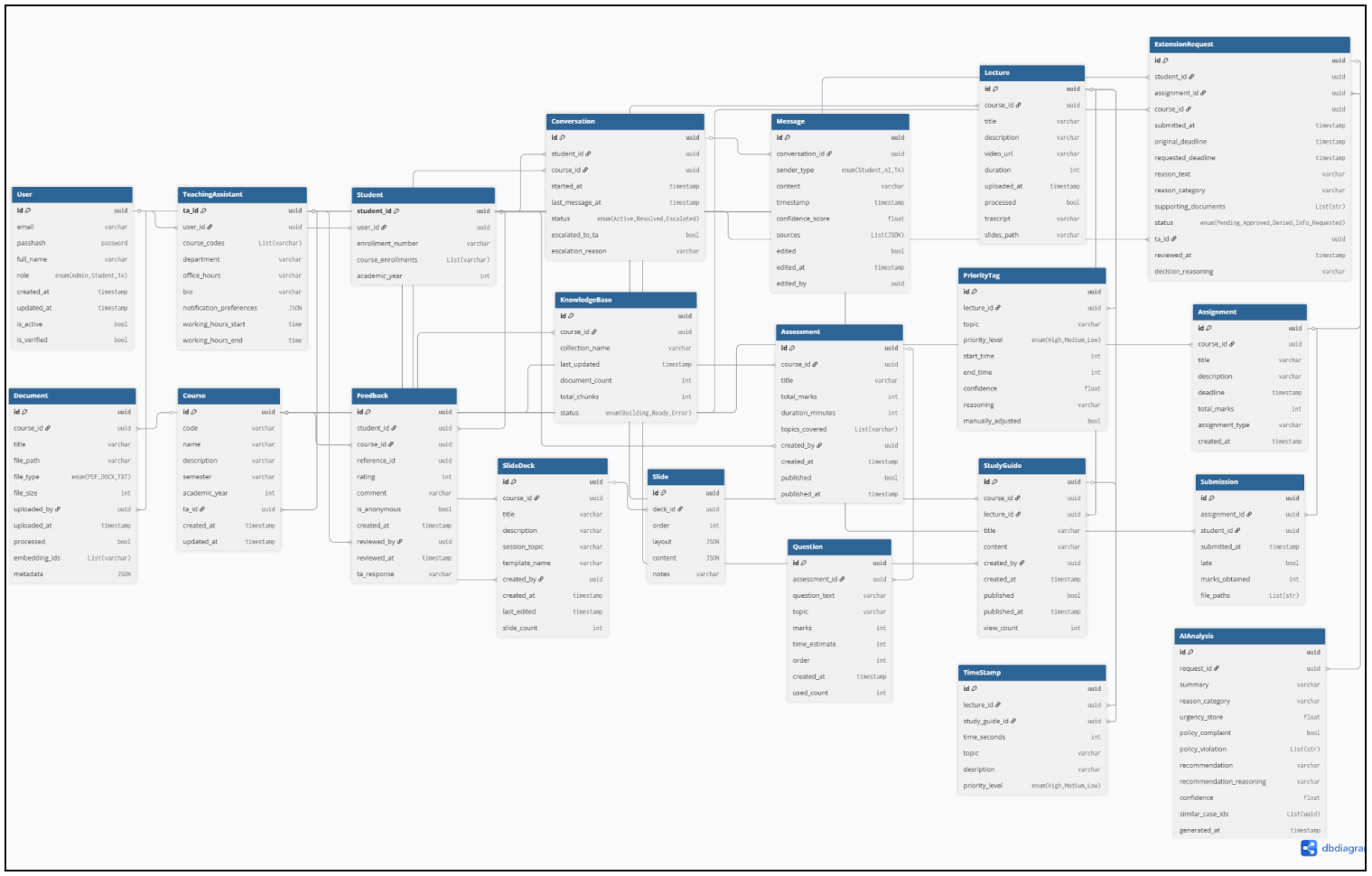
Component interactions follow clear patterns: frontend components trigger API calls, backend services orchestrate business logic and GenAI operations, repositories handle data persistence, and middleware provides cross-cutting concerns like authentication and logging.

GenAI integration leverages Gemini API for text generation and embeddings, LangChain for agentic workflows, and ChromaDB for semantic search, enabling sophisticated features like RAG-based Q&A, automated assessment generation, and intelligent workflow automation.

Security is built into every layer with JWT authentication, role-based access control, input validation, and protection against common vulnerabilities.

The system is designed for **scalability** with async operations, connection pooling, and stateless architecture, while remaining **cost-effective** by using free tiers of all services (Vercel, Render, Neon, Gemini API).

ER Diagram (*click here and zoom for clear view*)



Class(UML) Diagram *(click here and zoom for clear view)*