College Support
Multi-Agent Plan

Readable project plan with implementation-ready structure

Project Context: Student support platform (Registration + IT)

Architecture: 1 Orchestrator + 2 Agents (Knowledge + Workflow)

Generated: 2/23/2026, 10:03:03 AM

This project has been created as part of the 42 curriculum by <login1>, <login2>, <login3>, <login4>.

College Support Multi-Agent Portal

Description

College Support Portal is a web application for student support in Registration and IT.

It uses a multi-agent architecture:

- Orchestrator Agent: classifies intent and routes requests.
- Knowledge Agent: answers from official FAQ/documents.
- Workflow Agent: creates and tracks tickets.

Core objective: reduce reception load by guiding students and escalating uncertain requests to staff.

Instructions

Prerequisites

- Docker + Docker Compose
- Node.js 22+ (if running without Docker)

Run with Docker (recommended)

1. Copy .env.example to .env and fill required values (at minimum JWT_SECRET, optionally GEMINI_API_KEY).

2. Run:

```
docker compose up --build
```

3. Open:

- App: http://localhost
- API health: http://localhost/api/health

Default bootstrap admin (change in .env for real deployments):

- School ID: 00000001
- Password: Admin1234!

Run locally without Docker

1. API:

```
cd api
cp .env.example .env
npm install
npm run prisma:generate
npx prisma db push
npm run start:dev
```

2. Web:

```
cd web
cp .env.example .env.local
npm install
npm run dev
```

3. Open:

- Web: http://localhost:3000
- API: http://localhost:4000/api

No-admin local database (portable PostgreSQL)

If Docker/PostgreSQL is not installed system-wide, use the bundled portable setup:

```
powershell -ExecutionPolicy Bypass -File scripts/start-postgres-portable.ps1
```

Check status:

```
powershell -ExecutionPolicy Bypass -File scripts/status-postgres-portable.ps1
```

Stop database:

```
powershell -ExecutionPolicy Bypass -File scripts/stop-postgres-portable.ps1
```

Team Information

- Product Owner (PO): _to be assigned_

- Project Manager / Scrum Master: _to be assigned_
- Technical Lead / Architect: _to be assigned_
- Developers: _all members_

Update this section with real team role distribution.

Project Management

- Task organization: GitHub Issues / project board
- Communication: Discord/Slack
- Cadence: weekly planning + review + retrospective

Technical Stack

- Frontend: Next.js (TypeScript)
- Backend: NestJS (TypeScript)
- ORM: Prisma
- Database: PostgreSQL 16 + pgvector extension
- AI provider: Gemini API
- Deployment: Docker Compose (web, api, db, reverse-proxy)
- Reverse proxy: Nginx

Database Schema

Main entities:

- users, profiles
- tickets, ticket_events, attachments
- knowledge_documents, knowledge_chunks, faq_entries
- orchestrator_traces
- api_keys
- notifications

Relations are implemented in api/prisma/schema.prisma.

Features List

- Student registration/login with 8-digit school ID and password.
- Role-based access (STUDENT, STAFF, ADMIN).
- Chat-first assistant endpoint with orchestrator routing.
- Knowledge retrieval from FAQ + uploaded documents.
- Confidence-governed escalation to ticket flow.
- Ticket creation, queue, claim, status updates.
- Attachment upload/delete with validation.
- Notification unread count.
- Admin tools: FAQ/doc ingestion, staff provisioning, API key generation, trace review.

- Public API (API-key protected) with 5 required endpoints.
- Privacy Policy and Terms of Service pages.

Modules

- Major (2 pts each)
- Web frameworks (frontend + backend)
- Standard user management/authentication
- Advanced permissions system
- AI RAG system
- AI LLM system interface
- Public API with security/rate limiting + 5 endpoints
- Custom module: multi-agent orchestration with confidence policy
- Minor (1 pt each)
- ORM (Prisma)
- File upload and management
- Advanced search

Total target: 17 points.

API Overview

Internal JWT API

- POST /api/auth/register-student
- POST /api/auth/login
- POST /api/auth/admin/provision (admin)
- GET /api/me
- POST /api/assistant/message
- GET /api/tickets/my
- GET /api/tickets/queue (staff/admin)
- POST /api/tickets/:id/claim (staff/admin)
- PATCH /api/tickets/:id/status (staff/admin)
- POST /api/admin/knowledge/documents (admin)
- POST /api/admin/faq (admin)

External Public API (API key)

- GET /api/public/docs
- GET /api/public/knowledge/search
- POST /api/public/tickets
- GET /api/public/tickets/:ticketId
- PUT /api/public/tickets/:ticketId
- DELETE /api/public/tickets/:ticketId/attachments/:attachmentId

Individual Contributions

- _To be completed by team before submission._
- Include concrete features/modules per person and challenges solved.

Resources

- NestJS docs: https://docs.nestjs.com
- Next.js docs: https://nextjs.org/docs
- Prisma docs: https://www.prisma.io/docs
- pgvector docs: https://github.com/pgvector/pgvector
- Gemini API docs: https://ai.google.dev

AI Usage Disclosure

AI was used for:

- Initial architecture drafting and module mapping
- Boilerplate generation and refactoring assistance
- DTO/service/controller skeleton creation
- Documentation acceleration

All generated output was reviewed, edited, and validated by the team before use.