Architecture & Diagrams [AITUConnect]

System Architecture (Monolith)

AITUConnect follows a monolithic backend architecture implemented as a single Go application.
All core modules (authentication, users, groups, posts, interactions, and administration) are part of one deployable unit and communicate through internal service layers.
The system exposes a RESTful HTTP API and uses PostgreSQL as a primary data store.

Architectural Layers:

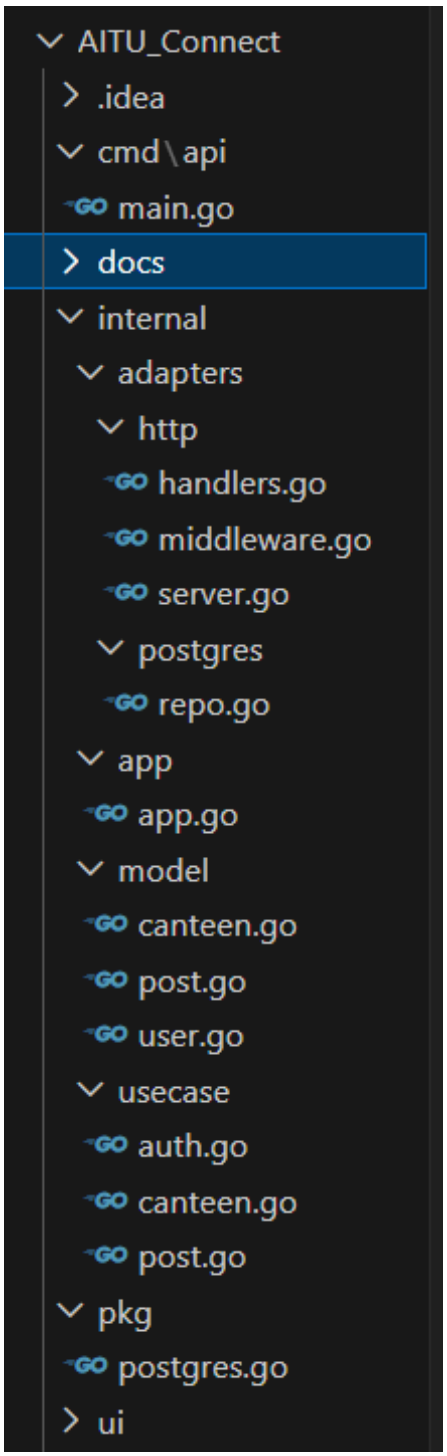
1.Client sends an HTTP request to the backend

2.The API layer handles routing and middleware

3.Business logic is executed in the application (use case) layer

4.Domain models are used to represent entities

5.Data is stored and retrieved from DB



Folder Structure and Responsibilities:

cmd/api Application entry point. Handles configuration, dependency injection, and server startup. Does not contain business logic.

internal/adapters/http Transport layer. Manages routing, middleware (JWT, logging), and request/response handling. Responsible for data validation and status codes.

internal/adapters/postgres Data access layer. Implements repository interfaces using PostgreSQL for persistent storage and retrieval of domain entities.

internal/app Bootstrap layer. Functions as the glue that initializes components and wires together the HTTP handlers with the use case services.

internal/usecase Business logic layer. Implements core application scenarios such as user authentication, post management, and group interactions.

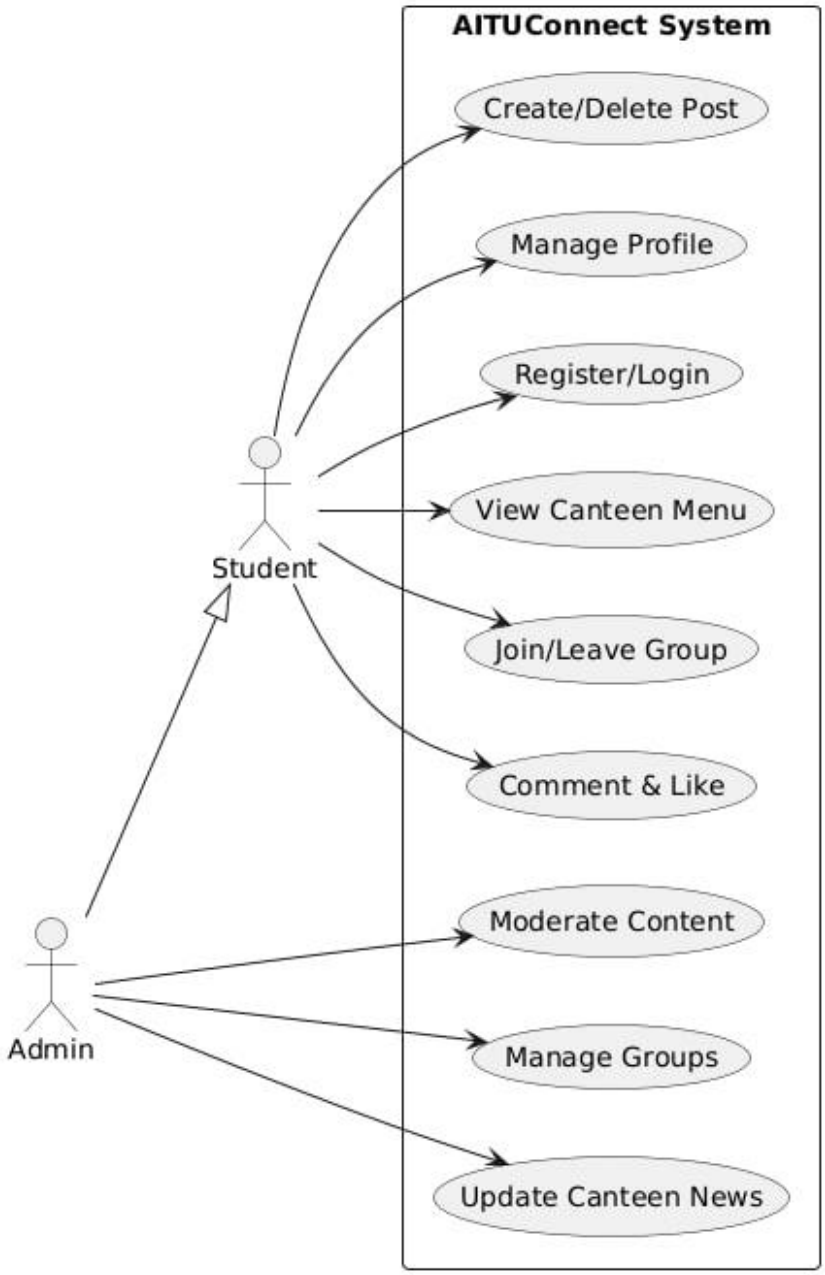
internal/model Domain layer. Defines the central data structures (User, Post, CanteenNews) used as the common language across all application layers.

pkg/postgres.go Shared infrastructure. Provides boilerplate code for database connection pooling and driver setup.

docs/ Project documentation. Contains the system architecture, ERD, Use-Case diagrams, and technical specifications.

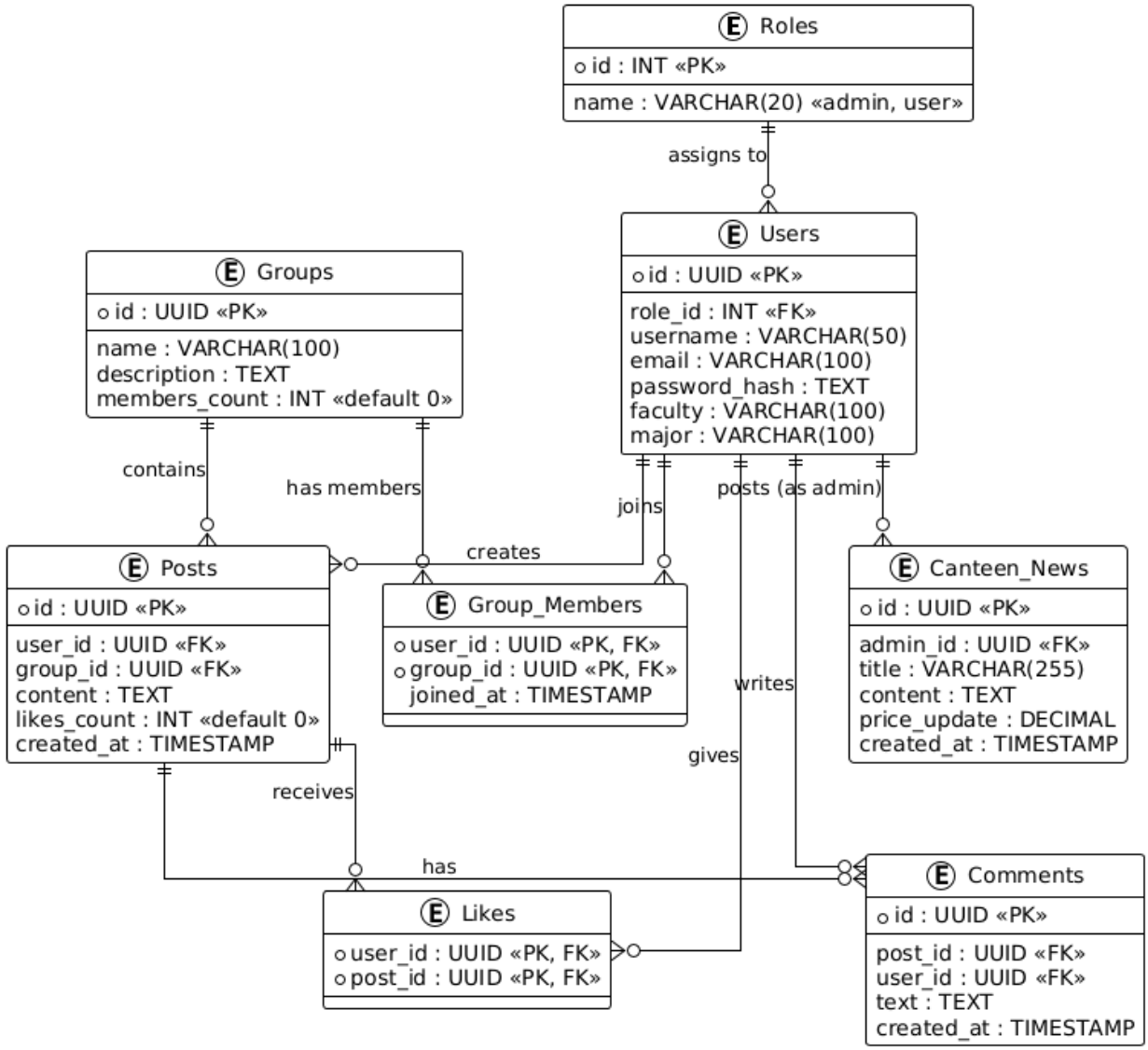
ui/ Frontend layer. A minimal web interface or static assets used to interact with the backend API.

Use-Case Diagram



The Student actor interacts primarily with the **usecase/auth.go**, **usecase/post.go**, and **usecase/canteen.go** modules. The Admin actor has extended permissions to trigger moderation and administrative functions that are restricted for regular students.

ERD Diagram



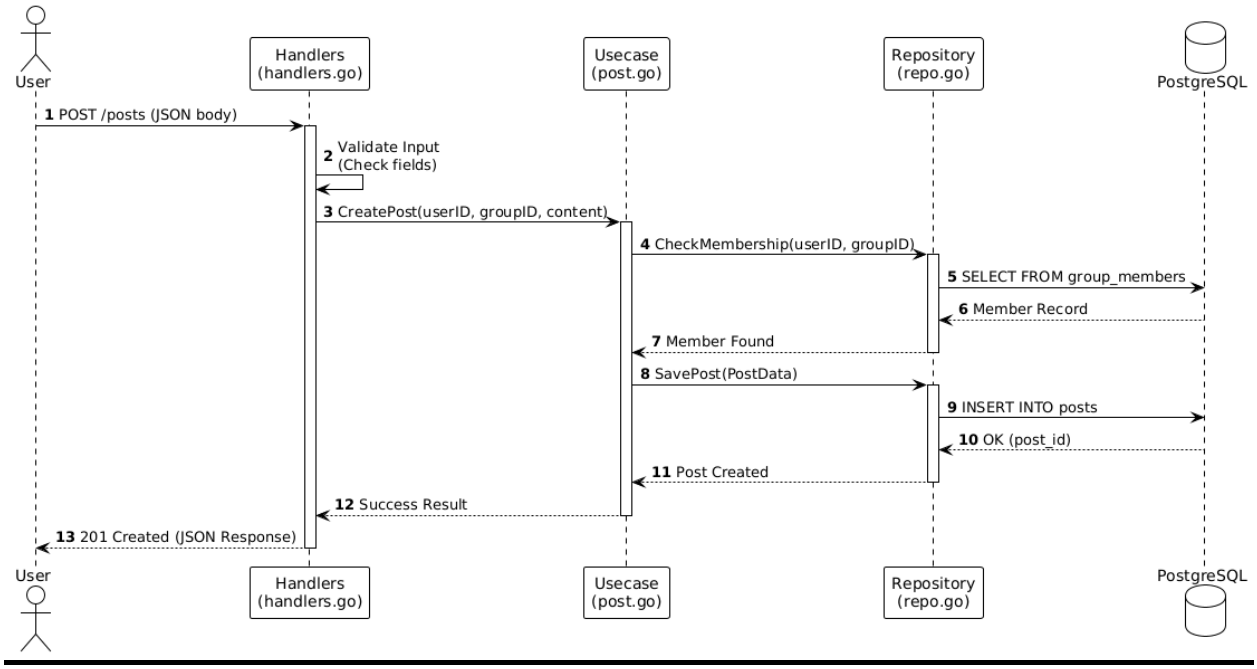
User Management: Supports two distinct roles (Admins and Users). Users can join multiple clubs, while Admins have the exclusive authority to post canteen updates.

Social Interaction: A complete social ecosystem allowing users to create Posts within specific Groups, write Comments, and Like content.

Performance: The system tracks likes_count and members_count directly in the tables.

Scalable Architecture: Uses UUIDs for identifiers to ensure data security and future scalability, making the system robust enough for a large-scale university environment.

UML Diagram of Creating a Post



The sequence diagram illustrates the lifecycle of a "Create Post" request, demonstrating the interaction between the client and the backend layers