Software Architecture Document (SAD)

Hygge P2P Simulator

Version No	Description	Contributors	Date
1.0	Initial Draft	Julian Arroyave	20-07-2024
1.1	Introduction Architecture Goals and Logical architecture	Julian Arroyave	23-07-2024
1.2	Component View	Julian Arroyave	01-05-2025
1.3	Data Architecture	Julian Arroyave	09-05-2025
1.4	Update Backend Component View	Julian Arroyave	

1. Introduction

This document describes the proposed software architecture for the Hygge Power Trading Simulator. It outlines the major components, their interactions, the data model, technology stack, and deployment strategy, guided by the requirements specified in the BRD. The goal is to create a scalable, maintainable, and performant system.

2. Architectural Goals & Constraints

Goals: Modularity, Scalability (handle many nodes & long simulations), Maintainability (clear separation of concerns), Testability, Performance (efficient simulation execution), Usability (responsive UI).
Constraints: Must use React for Frontend, Python/FastAPI for Backend API, Peewee ORM, PostgreSQL database. Must implement a 3-layer backend architecture. Must support deployment across DEV, TEST, INT, PROD environments.

3. Logical Architecture

3.1. Layered View (Backend)

API Layer (FastAPI):

Responsibilities:

- Handles HTTP requests (RESTful endpoints),
- Authentication & Authorization (using JWT),
- Request Validation (using Pydantic models),
- Data Serialization/Deserialization (JSON),
- Delegates business logic to the Service Layer.
- Provides interface for the Frontend.

Service Layer (Python Classes/Modules):

Responsibilities:

- Contains the core business logic.
- Orchestrates operations like topology management, profile processing, simulation setup, running the simulation engine, applying allocation algorithms, calculating results.
- Interacts with the Data Layer to fetch/persist data.
- Stateless where possible to aid scalability.

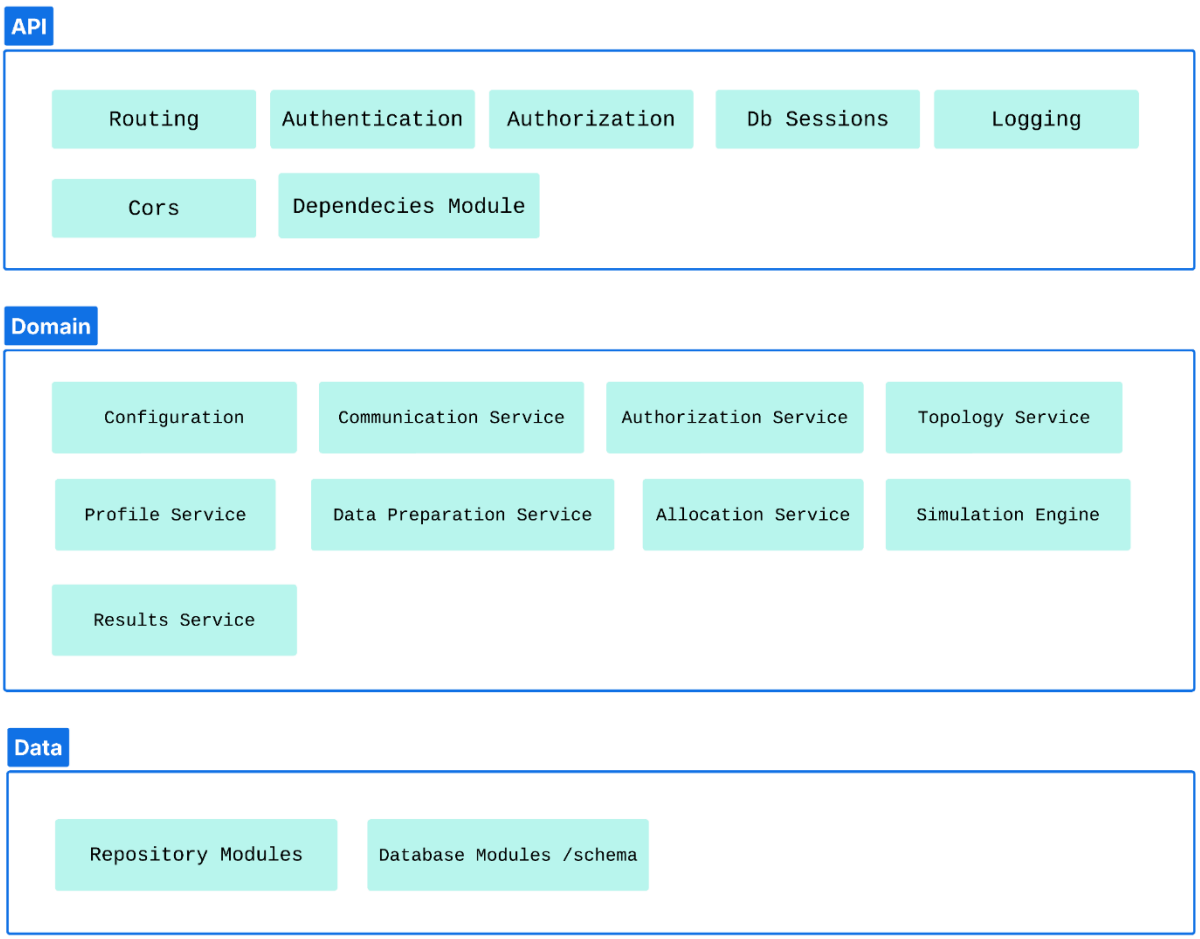
Data Layer (Python Modules with Peewee/Repository Pattern):

Responsibilities:

- Abstracts database interactions.
- Implements Repository pattern for data access (TopologyRepository, ProfileRepository, ResultsRepository).
- Defines Peewee models mapping to database tables.
- Handles all CRUD operations and complex queries.
- Ensures data integrity.

3.2. Component View

Backend Application (FastAPI):



The backend application follows a 3-layer architecture (API, Domain/Service, Data) and includes the following key components:

API Layer:

- **API Endpoints Module:** Defines routes and request handlers for various functionalities like topology, configuration, profiles, simulation, results, and authentication.

- **Authentication Module:** Handles user authentication (e.g., token validation) and authorization logic at the API level.
- **Middleware:** Manages cross-cutting concerns such as CORS, request/response logging, and database session management.
- **Dependencies Module:** Reusable functions and dependency injector container utilizing FastAPI's `Depends` system to inject dependencies (services, repositories via services, configuration objects…) into API endpoints.

Domain Layer (Service Layer):

- **Authorization Service:** Contains business logic related to user roles and permissions.
- **Topology Service:** Implements business logic for managing the network topology (creating, updating, deleting grids, transformers, houses…).
- **Profile Service:** Handles business logic for managing raw load and solar profiles (loading, storing).
- **Profile Data Preparation Service:** Implements the logic to transform raw profile data into simulation-ready time-series data (15-minute intervals for a full year).
- **Allocation Service:** Contains the business logic for the energy allocation algorithm.
- **Communication Service:** Manages any communication-related business logic (notifications, OTP sending).
- **Simulation Module:** Encapsulates the orchestration of simulation runs (setup, state management) and the core simulation engine logic (step-by-step calculations, battery management, applying allocation).
- **Results Service:** Implements business logic for querying, aggregating, and formatting simulation results for presentation.

Data Layer:

- **Repository Modules:** Abstract database interactions using the Repository pattern for different entities (User, Grid, Profile, Simulation…).
- **Database Models/Schemas:** Defines the structure of the data (e.g., Peewee models) that maps to database tables.
- **Cross-Cutting Concerns / Supporting Modules:**
- **Configuration Module:** Manages application configuration from various sources (environment variables, .toml files).
- **Utility Module:** Provides common utility functions used across the application (date/time helpers, string manipulation).
- **Exception Handling Module:** Defines custom exceptions and manages how exceptions are handled and reported.

Frontend Application (React):

UI Components (Topology viewer/editor, Configuration forms, Profile managers, Simulation setup, Results dashboards/charts).
State Management.
API Client Service (Handles communication with Backend API).
Build/Development Tools (React App).

4. Data Architecture

Database: PostgreSQL relational database.
ORM: Peewee.
Key Tables (Illustrative):
users (user_id, username, password_hash, role_id)
roles (role_id, role_name)
grids (grid_id, name, created_by, created_at)
transformers (transformer_id, grid_id, name, capacity_kw, ...)
houses (house_id, transformer_id, name, house_type, ...)
house_flags (flag_id, house_id, flag_name)
profiles (profile_id, house_id, profile_type (load/solar/...), source_type (upload/builder...), config_details_json, data_storage_ref) -> Time-series data might be stored efficiently, e.g., in related tables, JSONB, or potentially a dedicated time-series extension/DB if performance dictates.
simulation_runs (run_id, topology_snapshot_json, parameters_json, user_id, start_time, end_time, status)
simulation_results (result_id, run_id, node_id, timestamp, load_kw, generation_kw, battery_soc, net_energy_kwh, ...) -> This table can become very large. Consider partitioning or aggregation strategies.
priority_lists (list_id, name, configuration_json)
Topology Storage: While stored relationally, provide export/import functionality using JSON representation.

Time-Series Data: Needs careful consideration for storage and querying performance (indexing, partitioning).

5. Technology Stack

Frontend: OR React (latest LTS)
Backend: Python 3.11+, FastAPI
ORM: Peewee
Database: PostgreSQL 14+
API Specification: OpenAPI (auto-generated by FastAPI)
Caching (Optional): Redis (for caching results or session data if needed)

6. Deployment View

Environments: DEV, TEST, INT, PROD.
Infrastructure: Recommend containerization using Docker.
docker-compose.yml for local development (Frontend, Backend, DB).
TEST/INT/PROD environments.

Deployment:
CI/CD pipeline (GitLab CI, GitHub Actions) to automate testing, building Docker images, and deploying to respective environments.
Backend API and Frontend served as separate containers/services.
PostgreSQL typically run as a managed database service in cloud environments (AWS RDS, for scalability and reliability.)

7. Non-Functional Requirements Implementation

Performance: Efficient database queries (indexing, optimized ORM usage), potentially asynchronous simulation execution for long runs,
Profile generation logic optimized.
Scalability: Stateless backend services where possible allow horizontal scaling. Database scalability through managed services

Usability: Clean API design, responsive frontend framework, clear visual components for topology and results.
Reliability: Transaction management in DB operations, error handling and logging, repeatable simulation logic. Potential for saving simulation state periodically for long runs.
Security: JWT-based authentication, HTTPS enforcement, role-based authorization checks at the API layer, input validation.
Maintainability: Strict adherence to layered architecture, use of Repository pattern, code linting/formatting, dependency injector, unit/integration tests, comments/documentation.

8. Design Decisions & Trade-offs