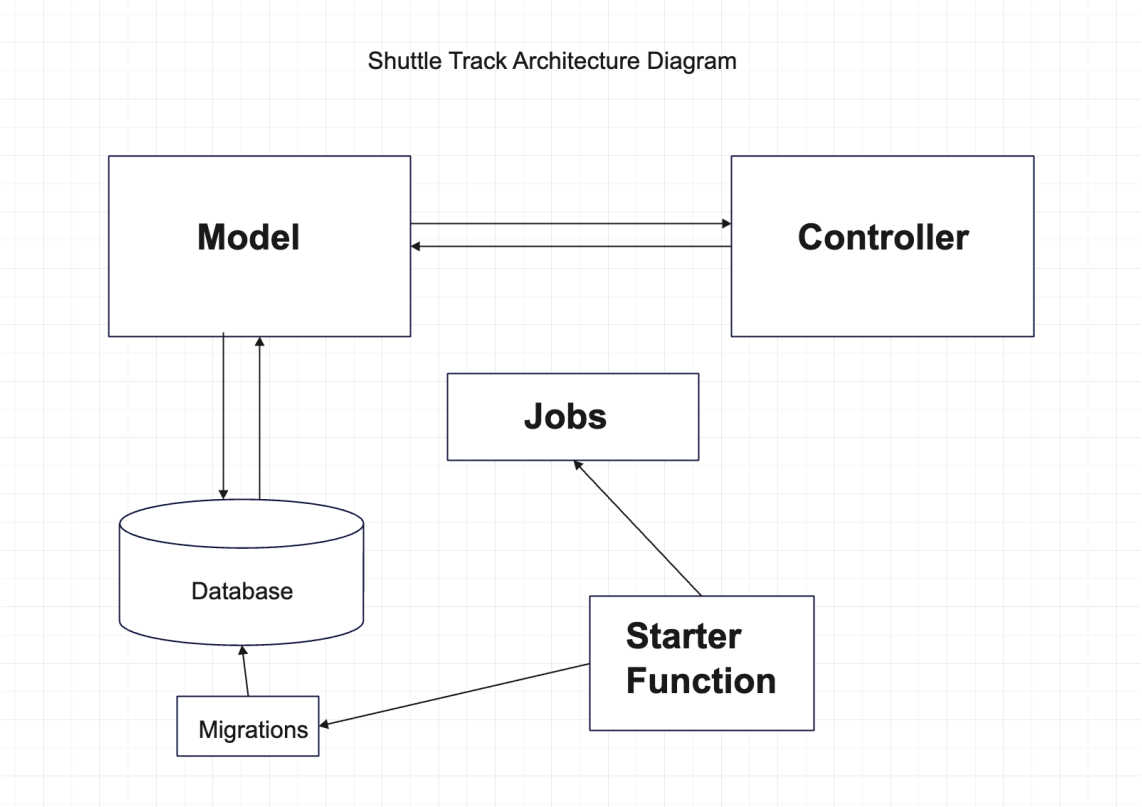
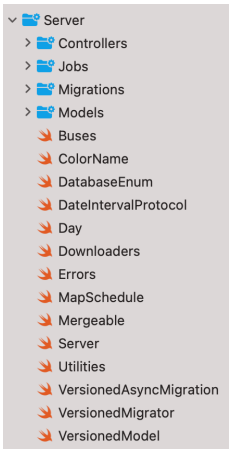
1. High-Level Overview

1. Architecture: Provide a visual representation of the system's architecture, including databases, services, and their interactions.
    a. Architecture Pattern:
        i. Model: The Model corresponds to the structure of the data stored in the database. It defines the type of data stored within the database. The model corresponds with the Controller and Database Component.
        ii. Controller: Controllers act as an interface with the Model component to process all the business logic and incoming requests, manipulate data using the Model component and interact with the Views to render the final output.
        iii. Other
        iv. Migrations:Migrations are a way of versioning the database schema and managing its evolution over time. Migrations allow developers to define changes to the database schema in code form, making it easier to apply changes across different environments consistently (development, staging, production) and to collaborate with other developers by keeping the database schema changes under version control.
        v. Jobs: Jobs, refer to operations or tasks that are executed outside the main flow of a web application, usually asynchronously. These can include sending emails, processing files, batch data processing, scheduled tasks, and more.



2. Technology stack:
    a. Swift
        i. Vapor is a web framework for Swift that allows developers to build web applications and APIs. It provides tools and libraries for routing, middleware, authentication, templating, and more.
        ii. Vapor is built on top of SwiftNIO, Apple's asynchronous event-driven networking framework, which enables high-performance, non-blocking I/O operations.
        iii. It follows the MVC (Model-View-Controller) pattern for organizing code and separating concerns.
        iv. Vapor supports asynchronous programming paradigms, making it efficient for handling large numbers of concurrent connections.
        v. It is open-source and has an active community contributing to its development and improvement.
    b. Fluent:
        i. Fluent is an ORM (Object-Relational Mapping) framework for Swift, designed specifically for working with databases in server-side applications.
        ii. It provides a fluent interface for interacting with databases, allowing developers to define models, query data, and perform database operations using Swift's type-safety and expressive syntax.
        iii. Fluent supports multiple database backends such as PostgreSQL, MySQL, SQLite, and more, enabling developers to choose the database that best fits their application requirements.
        iv. It handles tasks such as database migrations, schema creation, and data manipulation, abstracting away the complexity of working directly with raw SQL.
        v. Fluent seamlessly integrates with Vapor, making it easy to build database-driven web applications and APIs using Swift.
3. Code organization: Describe how the code is organized (e.g., directory structure) and the rationale behind it.



2. Key Components

Document the key components of your system, such as:

1. Core modules: Describe the purpose and functionality of each module.
2. Data flow: Explain how data moves through the system, highlighting any significant interactions between components.
3. External integrations: Document any external services or APIs the system integrates with and how those integrations work.

3. API Documentation

If your project exposes an API, provide detailed API documentation that includes:

1. Endpoints: List all the API endpoints, their URLs, and methods (GET, POST, etc.).
2. Parameters: Describe any parameters that can be passed to each endpoint, including data types and whether they are required or optional.
3. Responses: Document the possible responses for each endpoint, including status codes and data structures.