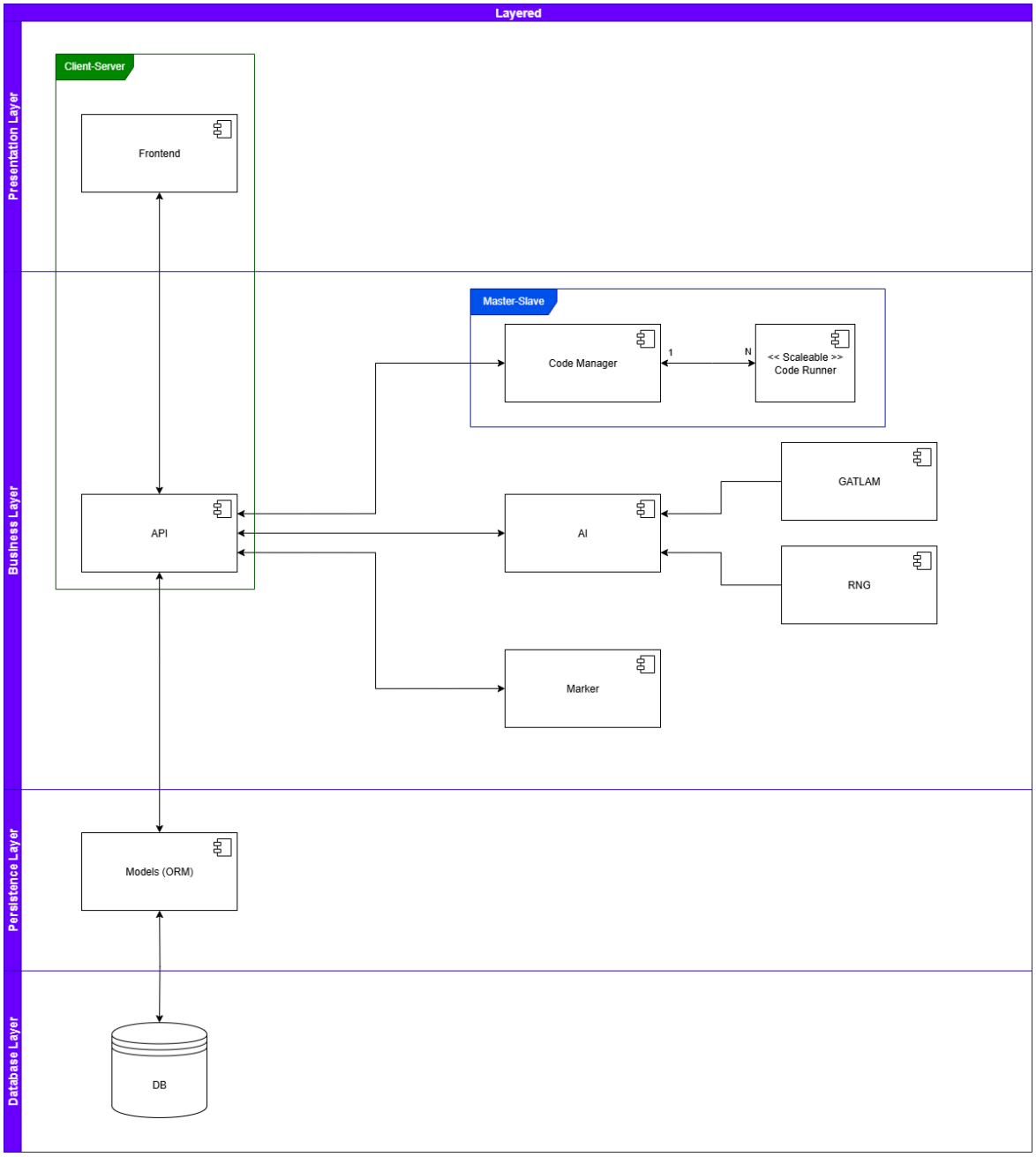
OWCA Advanced FitchFork Architectural Document

1. Overview

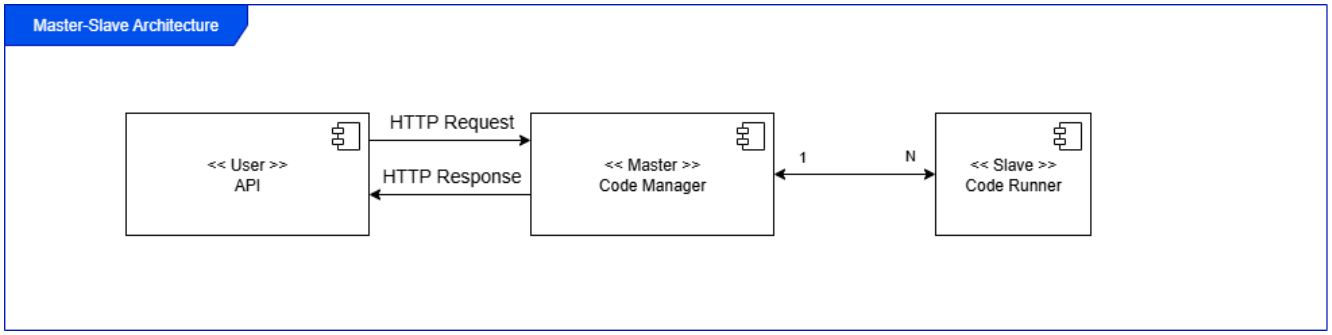
This system architecture diagram illustrates a four-tiered application structured into Presentation, Business, Persistence, and Database layers. It employs a Client-Server model, where a Frontend in the Presentation Layer communicates with a central API in the Business Layer.

The Business Layer handles the core logic, featuring an AI component and a scalable Master-Slave configuration (Code Manager and Code Runner) for distributed task processing.
Data management is handled by the Persistence Layer, which uses an Object-Relational Mapping (ORM) to abstract and manage interactions with the underlying database in the Database Layer, ensuring a modular, scalable, and maintainable system.



2. Master-Slave Architecture

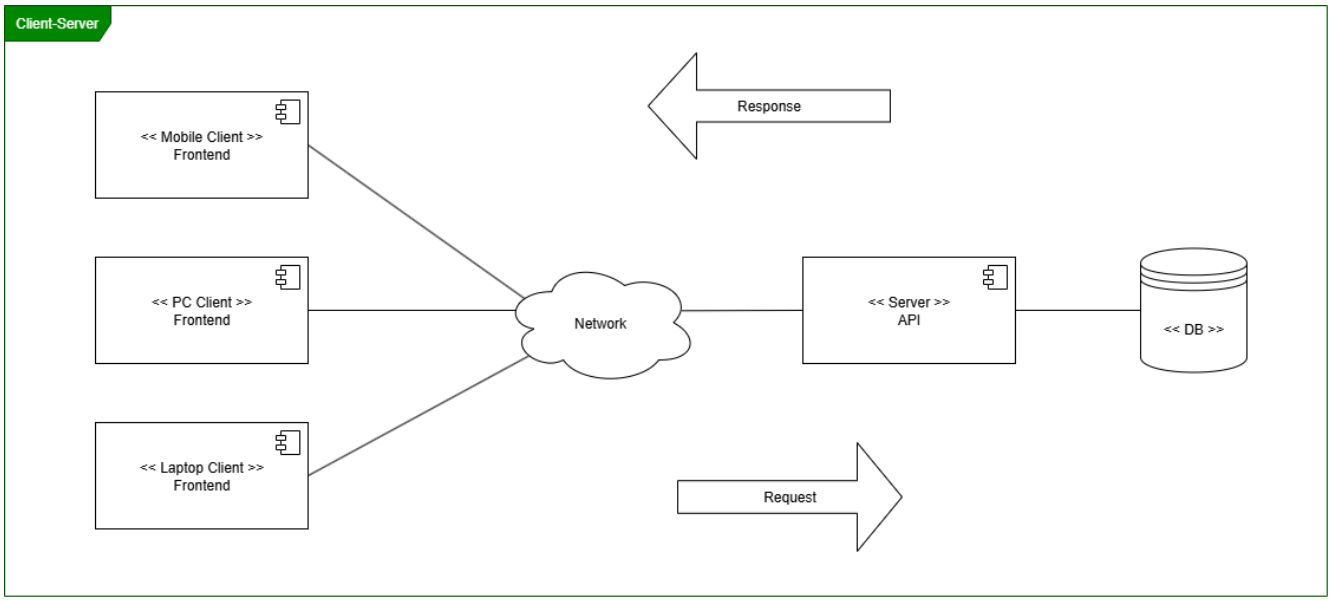
The use of a Master-Slave architecture is justified for this system as it provides an effective solution for distributing and scaling computationally intensive tasks.
By having a central Code Manager (Master) that delegates the work of code execution to multiple Code Runner instances (Slaves), the system can process numerous requests in parallel, significantly boosting performance and throughput.
This design is inherently scalable, allowing more slaves to be added to handle increased load, and it improves overall system resilience by isolating individual tasks, ensuring that the failure of one slave does not halt the entire application.



3. Client-Server Architecture

The client-server architecture is justified because it establishes a clear separation of concerns.
It allows multiple client types, such as mobile, PC, and laptop frontends, to interact with a single, centralised API server over a network.

This model centralises business logic and data management within the server, which processes all requests and interacts with the database, ensuring data consistency and security. By decoupling the user interface from the backend processing, this architecture supports independent development and scaling, allowing the application to be easily maintained and expanded to accommodate a growing number of users and diverse client platforms.



4. Layered Architecture

The use of a layered architecture is justified by its principle of separation of concerns, which divides the system into distinct tiers for presentation, business logic, and data management.
This modular approach significantly enhances maintainability and scalability by ensuring that changes in one layer, such as a user interface update in the Frontend, do not impact the core logic in the API or the data storage in the database.
This clear separation also facilitates parallel development, simplifies testing by allowing each layer to be tested independently, and provides the flexibility to update or replace components within a single layer with minimal disruption to the overall system.