**SOFTWARE ARCHITECTURE:**

1. Client-Server Architecture:

Client Side (Flutter):

- The Flutter frontend serves as the client in the client-server architecture.

- Flutter is responsible for providing the user interface and interacting with the user.

- In our app, Flutter components include screens like the HomePage and NutritionalConsumedScreen, where users input their information and view their weekly nutrition consumption.

Server Side (Django):

- Django backend acts as the server in the client-server architecture.

- Django handles incoming requests from the Flutter client, processes them, and sends back responses.

- Modules in Django, such as User Information, Dish Selector, and Nutritional Teller, implement the server-side logic for handling different functionalities of the app.

- Each module includes classes and functions responsible for processing data, fetching information from databases, and generating appropriate responses.

Interaction:

- When a user interacts with the Flutter client, such as submitting their profile information or viewing nutritional data, Flutter sends HTTP requests to corresponding endpoints in the Django backend.

- Django processes these requests, interacts with the database if necessary, and returns the requested data or performs the required actions.

- For example, the HomePage class in Flutter sends user information to the Django backend, which stores it in the database. The NutritionalConsumedScreen then requests weekly nutrition data from Django to display to the user.

2. MVC Services Architecture:

Model (Django Models):

- Django models represent the data structure of the application.

- In our app, Django models define entities like user profiles, dish information, nutritional data, etc.

- For example, the Model classes in Module-1 (User Information) represent the user's profile information and weekly nutrition consumption data.

View (Django Views and Flutter Widgets):

- Django views handle incoming HTTP requests and generate appropriate responses.

- In your app, Django views process requests from Flutter and return data or render templates.

- Flutter widgets represent the presentation layer of the application.

- UI components in Flutter, such as screens and widgets, display data received from the Django backend and handle user interactions.

- For example, the HomePage class in Flutter represents the view for collecting user information, while the NutritionalConsumedScreen class displays weekly nutrition data.

Controller (Django Views and Flutter Logic):

- Django views act as controllers, coordinating the interaction between models and views.

- They contain the business logic for processing requests, fetching data from models, and generating responses.

- In Flutter, logic for handling user inputs, managing state, and making HTTP requests to the backend serves as the controller.

- For example, in the IngredientInput class (Module-2), Flutter logic processes user inputs and sends them to Django to update the user's inventory.

3. Microservices Architecture:

Microservices (Django Modules):

- Each module in Django represents a microservice responsible for a specific functionality of the app.

- Modules like User Information, Dish Selector, and Nutritional Teller encapsulate related functionality and data.

- They can be developed, deployed, and scaled independently, allowing for flexibility and modularity.

- For example, the Nutritional Teller module focuses on providing nutritional information and can be developed and maintained separately from other modules.

Communication (HTTP Requests):

- Communication between microservices is achieved through HTTP requests.

- When one microservice requires data or functionality from another microservice, it sends an HTTP request to the corresponding endpoint.

- For example, the Nutritional Teller module may send requests to the Dish Selector module to fetch dish recommendations based on user preferences.

Independence (Modular Development):

- Each module operates independently, with its own set of functionalities and data.

- Changes or updates to one module do not impact the operation of other modules.

- This allows for faster development cycles, easier maintenance, and better scalability.

- For example, updates to the User Information module do not affect the functionality of the Dish Selector module.

By leveraging these software architectures, our recipe predictor app achieves scalability, maintainability, and modularity while providing a seamless user experience across both the Flutter frontend and Django backend.