Meal Planner Application

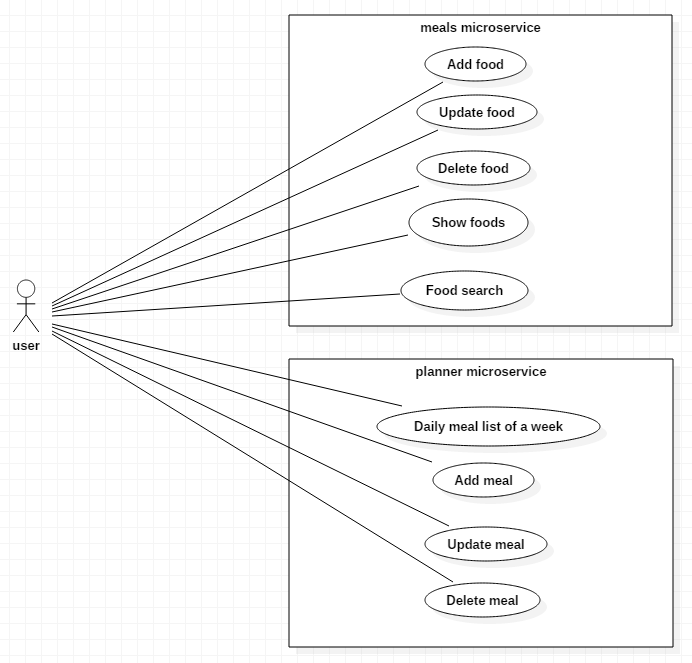
**Functional Requirements**

|  |  |
| --- | --- |
| **Title** | **Description** |
| Show meals of each of a week | Show list of meals planned for a week and daily. Initially a week containing list of days should suffice. Later there should be a calendar view showing a week and meals planned for each day of the week. |
|  |  |
| Add meal to a day | One should be able to add a meal planned to a day. |
| Update meal of a day | One should be able to update the existing meal of a day. |
| Delete meal of a day | One should be able to delete the existing meal of a day. |
| Automatic grocery generation per week | Grocery should be generated automatically when meals are planned for a week. For example, rice, carrots, beef, etc. should be added to grocery automatically. First, messaging the grocery list to Telegram is enough. |
|  |  |
| Add food | One should be able to add a food to food database to make it available for meal plans. |
| Update food | One should be able to update the food in the food database. |
| Delete food | One should be able to delete the food from the food database if not needed. |
| Search food | Text search for food/foods should be provided. |
| Show foods | Shows list of foods available in the database. |

**Non-functional requirements**

|  |  |
| --- | --- |
| **Title** | **Description** |
| Extensible | Family planner should be extensible for common planner for development extension. |
| Scalable | It should scale for multiple users using the software at the same time. |
| Secure | There should be authentication and authenticity. |

**Use Cases**



**Five Things Modern Web Applications should have**

* Internationalization
* Offline working capability
* Unit testing
* Asynchronity and Promises
* Deep linking / Promises

**System Architecture**

This section describes high-level architecture of the meal planner application.

Web UI (Angular JS)

meals service

plan service

json

json

**Design**

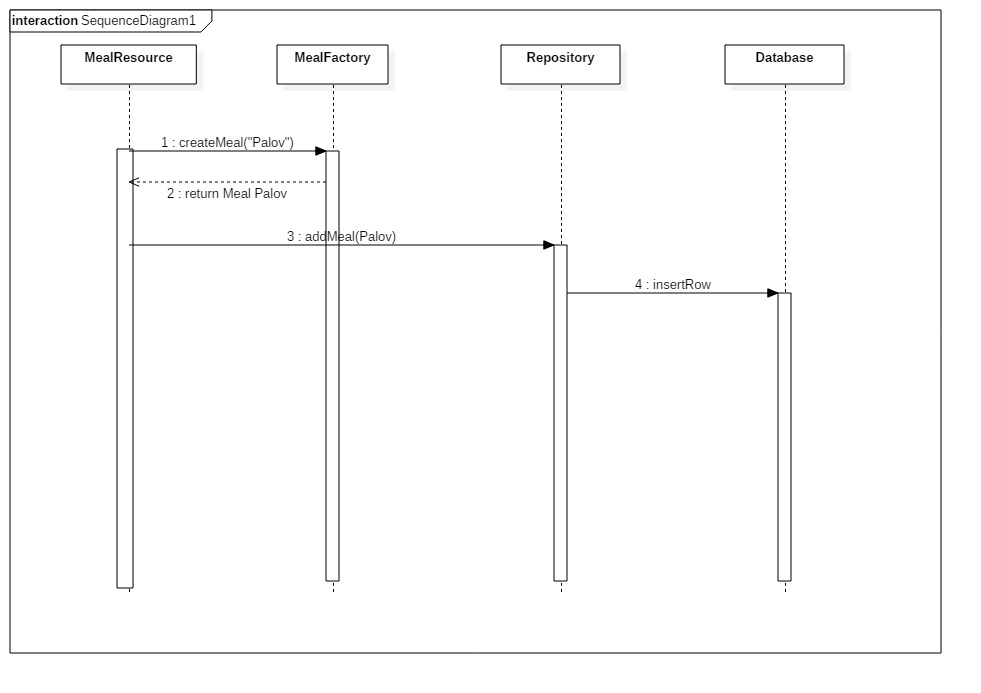
This section contains class diagram and sequential diagrams for the implementation.

**Technology stack**

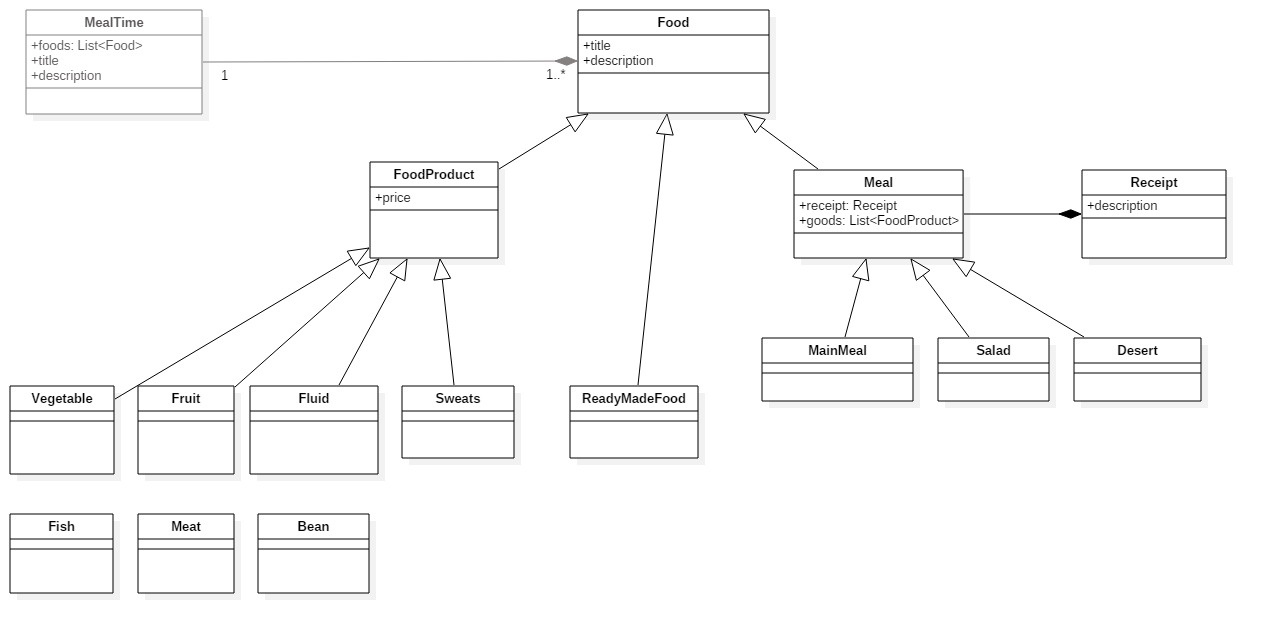
**Microservices**: Spring Boot, Hibernate, MySql ( later MongoDB)

**REST**: Spring MVC, Swagger

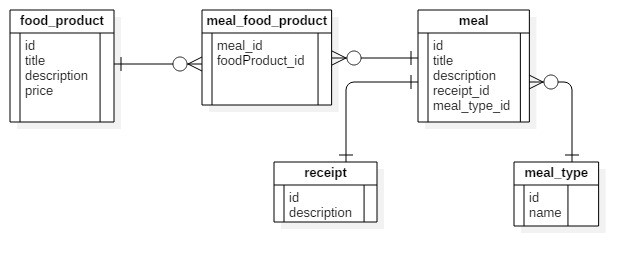
**Sequence Diagram For Repository (Data Access)**

****

**Class diagram**

****

**Entity relationship diagram**

****

**REST API design**

First we design the food service REST API.

**Meal**

Add: POST /meals

Update: POST/PATCH /meals/{id}

Delete: DELETE /meals/{id}

List: GET /meals

Get: GET /meals/{id}

**Receipt**

Add: PUT /meals/{id}/receipt

Update: POST/PATCH /meals/{id}/receipt

Delete: DELETE /meals/{id}/receipt

Get: GET /meals/{id}/receipt

**Food product**

Add: POST /foodProducts

Update: POST/PATCH /foodProducts/{id}

Delete: DELETE /foodProducts/{id}

List: GET /foodProducts

Get: GET /foodProducts/{id}

Get food products of a meal: GET /meals/{id}/foodProducts