For our system, we broadly split it into clients, servers and databases.

**CLIENT**

We have two clients here:

1. Web browser (web-based client)
2. Desktop-based client

For Web browser (web-based client) implementation, we have used the following:

* HTML
* CSS
* JavaScript

For Desktop-based client, we used the following:

* Java Swing

**SERVER**

For Server components implementation, we have used the following:

* Java Servlets

We are using web servers, and we run our server components on Apache Tomcat Server.

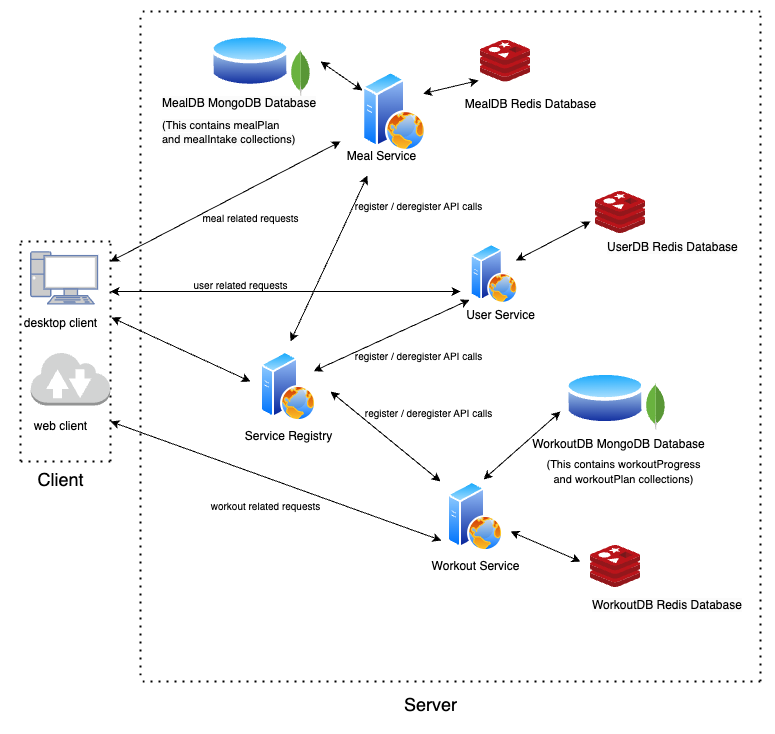
For Database implementation, we have used the following:

* MongoDB
* Redis

(This is covered extensively in the Database Design section)

Server components and Client components use HTTP Protocol for communication

Following is the high-level diagram of the system architecture:



Following are the client Components (where clients are the web browser and Desktop-based client):

1. Meal Plan UI

This is for viewing current meal plan, updating meal plan)

1. Meal Intake tracking UI

This if for updating meal intake, and tracking meal intake

1. Meal UI (only admins can access this)

This is for adding a new meal item (food item).

1. Workout Plan UI

This is for viewing current workout and updating the workout plan

1. Workout Progress UI

This is for tracking and updating the workout progress

1. User UI

This is for user registration and login into the app

1. Workout UI (only admins can access this)

This is for adding a new workout.

UI for above components is added in the “Use-cases and UI sketches” section.

Following are the different services with their respective components:

**WorkoutService**

This service takes care of handling the operations for workout, workout plan and workout progress related use-cases.

1. WorkoutServlet

This servlet exposes the APIs (GET and POST) related to workouts, workout plans, workout progress respectively.

1. WorkoutPlanAdapter

This contains the adapter functions used by the WorkoutServlet servlet to interact with Redis and MongoDB for CRUD operations on the Workout and workout plan related entities.

Databases used by WorkoutService:

1. Redis:

This is used to store Workout and WorkoutPlan entity records.

The following are the connection details:  
redis-14128.c267.us-east-1-4.ec2.cloud.redislabs.com:14128

Username : default

Password : 58qS8gtaj4hDaBwHVzS7BeqdEiVMaHBu

1. MongoDB:

This is used to store WorkoutPlanRecord and UserWorkoutRecord entity records.

The following are the connection details:

mongodb+srv://siddumsp:nzWdQDkRvi6sWq7F@sid-cluster.iavykex.mongodb.net/

**UserService**

This service takes care of handling the operations for user related use-cases.

1. UserServlet

This servlet exposes the APIs related to user registration, user details update and user login

1. UserAdapter

This contains the adapter functions used by the UserServlet to interact with Redis and MongoDB for CRUD operations on the user entity.

Databases used by UserService:

1. Redis:

This is used to store User entity records.

The following are the connection details:

redis-12523.c324.us-east-1-3.ec2.cloud.redislabs.com:12523

Username: default

Password: kcVyd8McAclJ39nlKrmPxtcjckoRo1Z9

**MealService**

This service takes care of handling the operations for meal, meal plan and meal intake related use-cases.

1. MealServlet

This servlet exposes the APIs (GET and POST) related to meals and meal plans and meal intake tracking respectively.

1. MealPlanAdapter

This contains the adapter functions used by the MealServlet servlet to interact with Redis and MongoDB for CRUD operations on the Meal and meal plan related entities.

Databases used by MealService:

1. Redis:

This is used to store Meal and MealPlan entity records.

The following are the connection details:  
redis-13901.c8.us-east-1-3.ec2.cloud.redislabs.com:13901

Username : default

Password : GdSrWOdsZAT7xPKFOet2L1zQxo36jTHb

1. MongoDB:

This is used to store MealPlanRecord and UserMealIntakeRecord entity records.

The following are the connection details:

<mongodb+srv://siddumsp:H14pFa76a5M8cbsC@storeapp.ss4cnol.mongodb.net/>

**ServiceRegistry**

This service exposes APIs (GET and POST) related to the following:

* Registering a new service
* Deregistering a service
* Fetch the address (port number) of a given service

ServiceRegistry helps route the client request to the appropriate microservice (by providing the service address).

Each microservice, on startup, it makes a call to the Service registry to get registered to the Service registry with their address (port number). Also, when a microservice goes down, it makes a call to the Service registry to get deregistered to the Service registry.

More about the APIs exposed by each of the services are discussed in the API documentation.

More about the above-mentioned databases is elucidated in the Database design documentation.

The following is the diagram that elucidates the flow of request between client, microservices and their respective databases:

