King Saud University

COLLEGE OF COMPUTER AND INFORMATION SCIENCES

DEPARTMENT OF SOFTWARE ENGINEERING

كلية علوم الحاسب والمعلومات
قسم هندسة البرمجيات

SWE 321| Software Design and Architecture

# WHAT TO EAT

| | Student Name | ID |
|---|---|---|
| 1 | Dana Alajaji | 442202315 |
| 2 | Layan Aluwaishiq" Leader" | 442201814 |
| 3 | Nada Alkubra | 442202368 |
| 4 | Nouv B. Al-Qahtani | 442201905 |
| 5 | Weam Alahmadi | 442200412 |

**Instructor Name:**

Rawan Alabdulrahman

## Contents

## Revision Control History:

| Version Number | Date | Description and Reasons |
|---|---|---|
| 1 | 19 – 9-2023 | Functional requirements Review |
| | 21-9-2023 | Functional and non-functional requirements review |
| 2 | 20-10-2023 | Architecture style, use case, use case descriptions review |
| | 21-10-2023 | Quality assurance, UI, sequence diagram review |

## System Glossary:

| Abbreviation | Description |
|---|---|
| **OTP** | stands for "One-Time Password." It is a temporary, unique code or password that is typically used for authentication or verification purposes.[1] |
| **Prompt** | An AI Prompt is any form of text, question, information, or coding that communicates to AI what response you're looking for. Adjusting how you phrase your prompt; AI could provide varying responses.[2] |
| **Recipe** | a set of instructions for preparing a dish. |
| **Meal** | is a prepared dish. |
| **Bb** | Blackboard Architecture. |

# Phase 1

## Introduction:

Food plays a key role in our daily lives, social interactions, get-togethers, and celebrations. However, finding meals and recipes that suit our tastes can be difficult. That's where the **"What to Eat"** app comes in. It recognizes the significance of food in our lives and works to solve this issue, enhancing users' dining experiences. By offering a range of recipe possibilities, streamlining meal selections, and promoting shared experiences, the app aims to improve, diversify, and customize the dining experience for singles, couples, and families. It serves as a valuable tool in the modern world, assisting users in finding meals and recipes that match their preferences while fostering social connections and culinary creativity.

The **"What to Eat"** app is a useful resource since it helps users plan meals, encourages kitchen inventiveness, and builds social bonds around shared food experiences. By offering a wider variety of options and creating a more pleasurable atmosphere, the app aims to improve eating experiences for singles, couples, and families.

In addition to recipe recommendations, the **"What to Eat"** app facilitates social interaction among families. Users can easily share their inventive recipes and best meals, promoting a sense of community and encouraging creativity in the kitchen. The app's capability to view family members' accounts and browse through their favorite meals further strengthens social ties by enabling users to create unique dishes and deepen familial bonds through the joy of food.

## Problem Domain Analysis:

**The problem** domain of the app revolves around addressing the challenge of deciding what to eat for couples and families. The proposed solution is to develop a mobile application called "What to Eat" that randomly recommends meal options for users and provides them with recipe details, and visual representations of the meals. The app can also recommend nearby restaurants or cafes that sell the meal.

## The System Context View:

*We wrote **three** Scenarios to understand our system Cleary.*

1. **User Registration:**

The user opens the application and enters their phone number then the system sends an OTP message to the user for verification once verified, the user is successfully registered. After that the user will enter their name then finally the system will ask user to fill survey about what their preferred food cuisines? And do you have any food allergy? Then the user will fill it and the system will store it in a database to use it when the user asks it about a recipe or meal.

**Benefit**: Users can create an account, ensuring a personalized experience within the app and easy access to their saved data.

2. **Prompting Meal or Recipes:**

User logs in to the "What to Eat" app, then their selects the option to prompt meal or recipes and write What meal their want and the specific ingredients then the system generate meal suggestions based on analyzing the prompt and their updated information “the information become from the survey” after that system will display the recipe and the list of nearby places to dine in this meal or to deliver “the delivery process will be at delivery’s app ” finally the user can save it or add it in weakly plan.

**Benefit**: Users can easily save and organize their favorite recipes for quick access and reference, saving time in searching for recipes again or suitable restaurants or delivery options.

3. **Creating and Managing Weekly Planner:**

User logs in to the "What to Eat" app, their selects the option to create a new weekly planner, the system will create unique code, the user can share weekly planner with other user by using link or enter the code in app, also users in weekly planner can add recipes link or name and can leave weekly planner.

**Benefit**: Users can plan their meals in advance, organize recipes for the week, and streamline their cooking process so they can collaborate with others in planning meals, making it easier to coordinate shared meals or events.

## Functions & NFP of the System:

## Functional Requirements:

### 1. User Requirement:

1. The User shall register by his/her phone number and his/her name.
2. The User shall log in by using his/her phone number.
3. The User shall be able to edit his/her name.
4. The User shall be able to edit his/her phone number.
5. The user shall be able to answer a survey about their preferred food, cuisines, and allergies.
6. The user shall be able to edit his/her preferred food, cuisines, and allergies.
7. The User shall be able to prompt meals recipes.
8. The User shall be able to save favorite recipes.
9. The User shall be able to view favorite recipes.
10. The User shall be able to edit favorite recipes.
11. The User shall be able to create a shared weekly planner.
12. The User shall be able to Join shared weekly planner by using a code or joining link.
13. The User shall be able to specify the name or link of recipes to the weekly planner.
14. The User shall be able to assign recommended recipes/meals directly to a weekly planner.
15. The User shall be able to view his/her weekly planner.
16. The User shall be able to edit his/her plans in weekly planner.
17. The user shall be able to leave a shared weekly planner.

### 2. System Requirement:

18. The system shall send an OTP "One-Time Password" message to the user.
19. The system shall be able to display a survey for users.
20. The system shall be able to specify the user's current location using the Global Positioning System (Google maps API).
21. The system shall be able to generate meal suggestions.
22. The system shall be able to suggest places to dine in or to deliver the meal.
23. The system shall be able to redirect the user to the food delivery App.

## Non-Functional Requirements:

1. Useability |The user shall take no more than 2 minutes to be able to know how to perform all the functionality.
2. Performance |The system shall be able to provide food recommendations in less than 5 seconds.
3. Reliability | the system shall be able available 99% of the time.
4. Scalability | The system shall be able to accommodate up to 100,000 users.

## Challenges:

The challenges may we faced when we develop “what to eat” app is how can we design an effective recommendation algorithm that analyses user preferences and generates meal suggestions and its response to do collecting and integrating a comprehensive and up-to-date database of meals, recipes, and restaurants. dealing with data inconsistencies and API limitations and ensuring regular updates to keep the information relevant and accurate. Additionally, integrating with external APIs of local eateries or food delivery platforms.

## Projection:

*By the end of the semester, we expect to accomplish the following skills and knowledge in relation to the development of the "**What to Eat**" project:*

1. **Practical experience:** By choosing the best architecture for the system, we will get hands-on practice applying architectural concepts and principles to a real-world project. In the future, we will be able to make educated architectural decisions depending on project requirements thanks to this knowledge, which will be helpful in our software development attempts.

2. **Project management and teamwork:** During the development process, we will improve our project management abilities by effectively organizing tasks, keeping track of progress, and interacting with team members. To ensure the timely completion of the app, this may require employing project management tools, holding frequent meetings, and organizing activities.

3. **Integration of external APIs:** We will gain knowledge and experience in integrating external APIs to fetch data from external food services, such as restaurant menus, food delivery platforms, or local eateries. This will involve understanding API documentation, authentication, and data retrieval processes such as OpenAI API.

# Phase 2 and 3

## Introduction:

Food plays a key role in our daily lives, social interactions, get-togethers, and celebrations. However, finding meals and recipes that suit our tastes can be difficult. That's where the **"What to Eat"** app comes in. It recognizes the significance of food in our lives and works to solve this issue, enhancing users' dining experiences. By offering a range of recipe possibilities, streamlining meal selections, and promoting shared experiences, the app aims to improve, diversify, and customize the dining experience for singles, couples, and families. It serves as a valuable tool in the modern world, assisting users in finding meals and recipes that match their preferences while fostering social connections and culinary creativity.

The **"What to Eat"** app is a useful resource since it helps users plan meals, encourages kitchen inventiveness, and builds social bonds around shared food experiences. By offering a wider variety of options and creating a more pleasurable atmosphere, the app aims to improve eating experiences for singles, couples, and families.

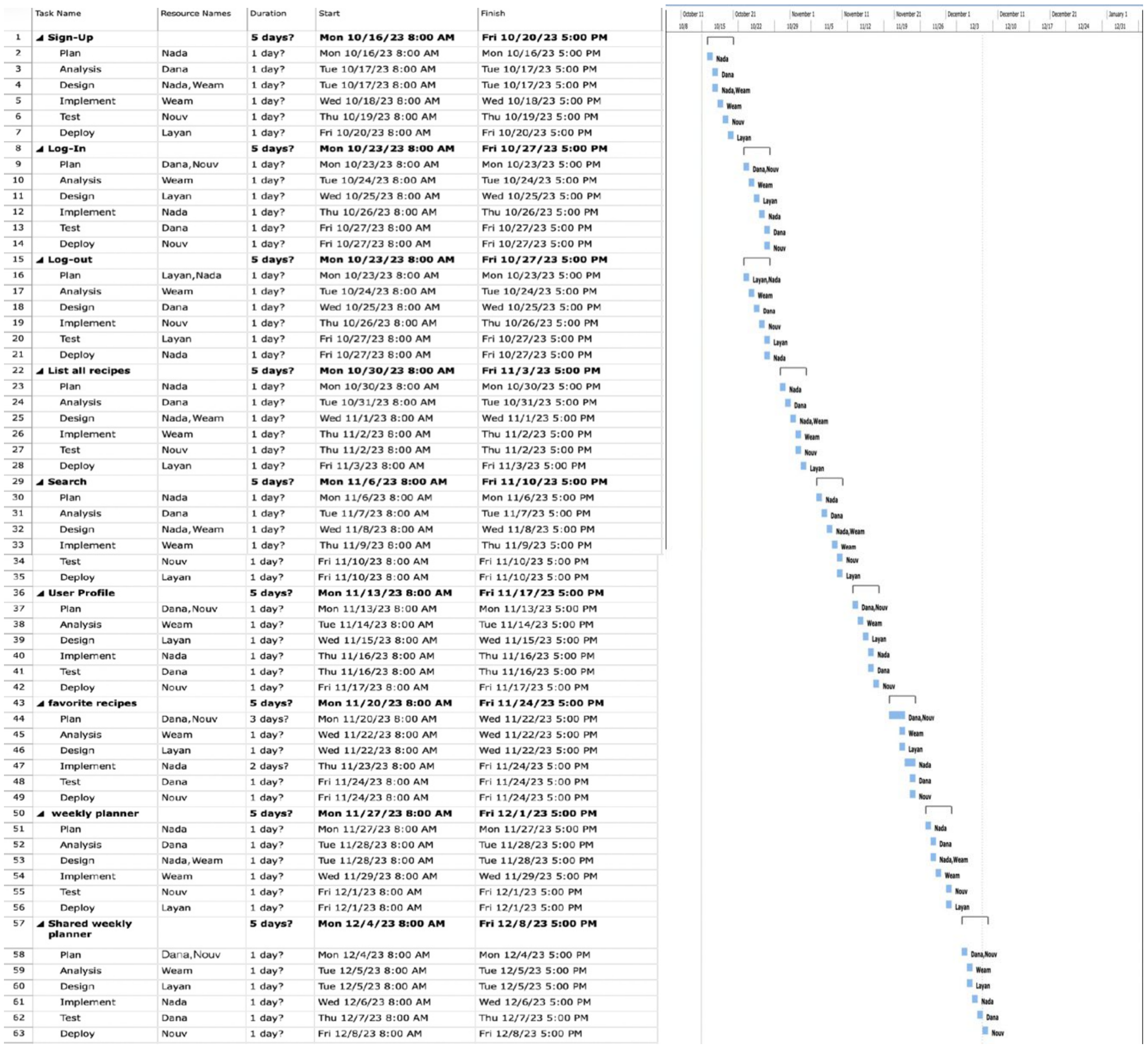
In addition to recipe recommendations, the **"What to Eat"** app facilitates social interaction among families. Users can easily share their inventive recipes and best meals, promoting a sense of community and encouraging creativity in the kitchen. The app's capability to view family members' accounts and browse through their favorite meals further strengthens social ties by enabling users to create unique dishes and deepen familial bonds through the joy of food.

## Methodology :

Agile Development is the methodology we choose after examining our system requirements, goals, and user demands.

Agile development is an iterative approach in which the total system is divided into sprints. Because modifications may be made without starting from scratch, Agile Development improves the system's adaptability and flexibility. Furthermore, it is dependent on customer collaboration so that we may receive feedback at the end of each sprint.

So, we divide the **What to Eat development into several Sprints and each Sprint we apply the SDLC steps on it.**



| | Task Name | Resource Names | Duration | Start | Finish |
|---|---|---|---|---|---|
| 1 | **◢ Sign-Up** | | **5 days?** | **Mon 10/16/23 8:00 AM** | **Fri 10/20/23 5:00 PM** |
| 2 | Plan | Nada | 1 day? | Mon 10/16/23 8:00 AM | Mon 10/16/23 5:00 PM |
| 3 | Analysis | Dana | 1 day? | Tue 10/17/23 8:00 AM | Tue 10/17/23 5:00 PM |
| 4 | Design | Nada, Weam | 1 day? | Tue 10/17/23 8:00 AM | Tue 10/17/23 5:00 PM |
| 5 | Implement | Weam | 1 day? | Wed 10/18/23 8:00 AM | Wed 10/18/23 5:00 PM |
| 6 | Test | Nouv | 1 day? | Thu 10/19/23 8:00 AM | Thu 10/19/23 5:00 PM |
| 7 | Deploy | Layan | 1 day? | Fri 10/20/23 8:00 AM | Fri 10/20/23 5:00 PM |
| 8 | **◢ Log-In** | | **5 days?** | **Mon 10/23/23 8:00 AM** | **Fri 10/27/23 5:00 PM** |
| 9 | Plan | Dana, Nouv | 1 day? | Mon 10/23/23 8:00 AM | Mon 10/23/23 5:00 PM |
| 10 | Analysis | Weam | 1 day? | Tue 10/24/23 8:00 AM | Tue 10/24/23 5:00 PM |
| 11 | Design | Layan | 1 day? | Wed 10/25/23 8:00 AM | Wed 10/25/23 5:00 PM |
| 12 | Implement | Nada | 1 day? | Thu 10/26/23 8:00 AM | Thu 10/26/23 5:00 PM |
| 13 | Test | Dana | 1 day? | Fri 10/27/23 8:00 AM | Fri 10/27/23 5:00 PM |
| 14 | Deploy | Nouv | 1 day? | Fri 10/27/23 8:00 AM | Fri 10/27/23 5:00 PM |
| 15 | **◢ Log-out** | | **5 days?** | **Mon 10/23/23 8:00 AM** | **Fri 10/27/23 5:00 PM** |
| 16 | Plan | Layan, Nada | 1 day? | Mon 10/23/23 8:00 AM | Mon 10/23/23 5:00 PM |
| 17 | Analysis | Weam | 1 day? | Tue 10/24/23 8:00 AM | Tue 10/24/23 5:00 PM |
| 18 | Design | Dana | 1 day? | Wed 10/25/23 8:00 AM | Wed 10/25/23 5:00 PM |
| 19 | Implement | Nouv | 1 day? | Thu 10/26/23 8:00 AM | Thu 10/26/23 5:00 PM |
| 20 | Test | Layan | 1 day? | Fri 10/27/23 8:00 AM | Fri 10/27/23 5:00 PM |
| 21 | Deploy | Nada | 1 day? | Fri 10/27/23 8:00 AM | Fri 10/27/23 5:00 PM |
| 22 | **◢ List all recipes** | | **5 days?** | **Mon 10/30/23 8:00 AM** | **Fri 11/3/23 5:00 PM** |
| 23 | Plan | Nada | 1 day? | Mon 10/30/23 8:00 AM | Mon 10/30/23 5:00 PM |
| 24 | Analysis | Dana | 1 day? | Tue 10/31/23 8:00 AM | Tue 10/31/23 5:00 PM |
| 25 | Design | Nada, Weam | 1 day? | Wed 11/1/23 8:00 AM | Wed 11/1/23 5:00 PM |
| 26 | Implement | Weam | 1 day? | Thu 11/2/23 8:00 AM | Thu 11/2/23 5:00 PM |
| 27 | Test | Nouv | 1 day? | Thu 11/2/23 8:00 AM | Thu 11/2/23 5:00 PM |
| 28 | Deploy | Layan | 1 day? | Fri 11/3/23 8:00 AM | Fri 11/3/23 5:00 PM |
| 29 | **◢ Search** | | **5 days?** | **Mon 11/6/23 8:00 AM** | **Fri 11/10/23 5:00 PM** |
| 30 | Plan | Nada | 1 day? | Mon 11/6/23 8:00 AM | Mon 11/6/23 5:00 PM |
| 31 | Analysis | Dana | 1 day? | Tue 11/7/23 8:00 AM | Tue 11/7/23 5:00 PM |
| 32 | Design | Nada, Weam | 1 day? | Wed 11/8/23 8:00 AM | Wed 11/8/23 5:00 PM |
| 33 | Implement | Weam | 1 day? | Thu 11/9/23 8:00 AM | Thu 11/9/23 5:00 PM |
| 34 | Test | Nouv | 1 day? | Fri 11/10/23 8:00 AM | Fri 11/10/23 5:00 PM |
| 35 | Deploy | Layan | 1 day? | Fri 11/10/23 8:00 AM | Fri 11/10/23 5:00 PM |
| 36 | **◢ User Profile** | | **5 days?** | **Mon 11/13/23 8:00 AM** | **Fri 11/17/23 5:00 PM** |
| 37 | Plan | Dana, Nouv | 1 day? | Mon 11/13/23 8:00 AM | Mon 11/13/23 5:00 PM |
| 38 | Analysis | Weam | 1 day? | Tue 11/14/23 8:00 AM | Tue 11/14/23 5:00 PM |
| 39 | Design | Layan | 1 day? | Wed 11/15/23 8:00 AM | Wed 11/15/23 5:00 PM |
| 40 | Implement | Nada | 1 day? | Thu 11/16/23 8:00 AM | Thu 11/16/23 5:00 PM |
| 41 | Test | Dana | 1 day? | Thu 11/16/23 8:00 AM | Thu 11/16/23 5:00 PM |
| 42 | Deploy | Nouv | 1 day? | Fri 11/17/23 8:00 AM | Fri 11/17/23 5:00 PM |
| 43 | **◢ favorite recipes** | | **5 days?** | **Mon 11/20/23 8:00 AM** | **Fri 11/24/23 5:00 PM** |
| 44 | Plan | Dana, Nouv | 3 days? | Mon 11/20/23 8:00 AM | Wed 11/22/23 5:00 PM |
| 45 | Analysis | Weam | 1 day? | Wed 11/22/23 8:00 AM | Wed 11/22/23 5:00 PM |
| 46 | Design | Layan | 1 day? | Wed 11/22/23 8:00 AM | Wed 11/22/23 5:00 PM |
| 47 | Implement | Nada | 2 days? | Thu 11/23/23 8:00 AM | Fri 11/24/23 5:00 PM |
| 48 | Test | Dana | 1 day? | Fri 11/24/23 8:00 AM | Fri 11/24/23 5:00 PM |
| 49 | Deploy | Nouv | 1 day? | Fri 11/24/23 8:00 AM | Fri 11/24/23 5:00 PM |
| 50 | **◢ weekly planner** | | **5 days?** | **Mon 11/27/23 8:00 AM** | **Fri 12/1/23 5:00 PM** |
| 51 | Plan | Nada | 1 day? | Mon 11/27/23 8:00 AM | Mon 11/27/23 5:00 PM |
| 52 | Analysis | Dana | 1 day? | Tue 11/28/23 8:00 AM | Tue 11/28/23 5:00 PM |
| 53 | Design | Nada, Weam | 1 day? | Tue 11/28/23 8:00 AM | Tue 11/28/23 5:00 PM |
| 54 | Implement | Weam | 1 day? | Wed 11/29/23 8:00 AM | Wed 11/29/23 5:00 PM |
| 55 | Test | Nouv | 1 day? | Fri 12/1/23 8:00 AM | Fri 12/1/23 5:00 PM |
| 56 | Deploy | Layan | 1 day? | Fri 12/1/23 8:00 AM | Fri 12/1/23 5:00 PM |
| 57 | **◢ Shared weekly planner** | | **5 days?** | **Mon 12/4/23 8:00 AM** | **Fri 12/8/23 5:00 PM** |
| 58 | Plan | Dana, Nouv | 1 day? | Mon 12/4/23 8:00 AM | Mon 12/4/23 5:00 PM |
| 59 | Analysis | Weam | 1 day? | Tue 12/5/23 8:00 AM | Tue 12/5/23 5:00 PM |
| 60 | Design | Layan | 1 day? | Tue 12/5/23 8:00 AM | Tue 12/5/23 5:00 PM |
| 61 | Implement | Nada | 1 day? | Wed 12/6/23 8:00 AM | Wed 12/6/23 5:00 PM |
| 62 | Test | Dana | 1 day? | Thu 12/7/23 8:00 AM | Thu 12/7/23 5:00 PM |
| 63 | Deploy | Nouv | 1 day? | Fri 12/8/23 8:00 AM | Fri 12/8/23 5:00 PM |

Figure 1: Detailed plan for What to Eat development using Gantt chart [5].

## System Architecture:

## Design decisions:

### Design Issue 1: How what to eat app will store the user's data and how user view the shared plans?

We chose **Amazon DynamoDB** from Amazon Web Services as the database option for our "What to Eat" app to offer efficient storage, scalability, and real-time synchronization of weekly plans, both individually and in groups. Due to DynamoDB's distributed architecture and automated scaling, which can support high levels of concurrency, multiple users can simultaneously upload and change their plans. Because everyone can view the plans at once and contribute while staying in sync, the group is able to coordinate and collaborate in real time. By employing DynamoDB's built-in synchronization mechanisms, particularly through DynamoDB Streams, updates to plans are quickly reflected, ensuring a seamless and up-to-date experience for all users. Additionally, because DynamoDB changes resources in the database automatically, its performance and responsiveness are improved.[3]

### Design Question 2: How will the What to Eat app recommend food recipes based on the information and the user's preferences?

We will use the **OpenAI API** and **Amazon DynamoDB** to make sure that the system generates food recipe suggestions depending on user preferences and the submitted form. Natural language processing capabilities will be used to extract the user's unique food preferences from the submitted form and analyze them using the OpenAI API. This data will be processed, and queries made against the Amazon DynamoDB recipe database will use the results. By matching the user's choices with pertinent recipes, DynamoDB will be the base for creating tailored suggestions. [6]

### Design Question 3: How will the What to Eat app control who has access to the shared planner?

When users join into shared weekly plans, it has been determined to create a **code access** or **joining link** way to ensure the security of the "What to Eat" app. The code access system adds an extra layer of security and only permits authorized users access by asking users to input a special code that was provided to them by the planner creator to access the shared planner. An alternative is for the planner creator to create a joining link that, when securely shared with intended users. enables access to the shared planner.

### Design Question 4: How will the What to Eat app authenticate users while logging in?

We have decided to use **OTP (One-Time Password)** authentication to ensure the safety of the "What to Eat" app throughout the login process. This choice is essential for protecting user accounts and limiting illegal access. The program will create a one-of-a-kind, time-limited OTP and send it to the user's registered phone number whenever they attempt to log in. The software confirms users' ownership of the registered phone number by requesting the right OTP during the login procedure. [1]

### Design Question 5: How will restaurants be suggested by the What to Eat app to users based on the meals they have requested?

We decided to leverage the **Google Maps API** to generate restaurant connections based on user requests and their current location to expand the functionality of our "What to Eat" app. Using the Google Maps API, we can find the user's current position and utilize it to find local eateries. Along with links to these restaurants' own websites or ordering platforms, we can obtain information about these eateries. By means of this integration, we can provide users with convenient access to restaurant details and order options that are tailored to their current location, thereby enhancing their overall dining experience. [12]

## Domain model:

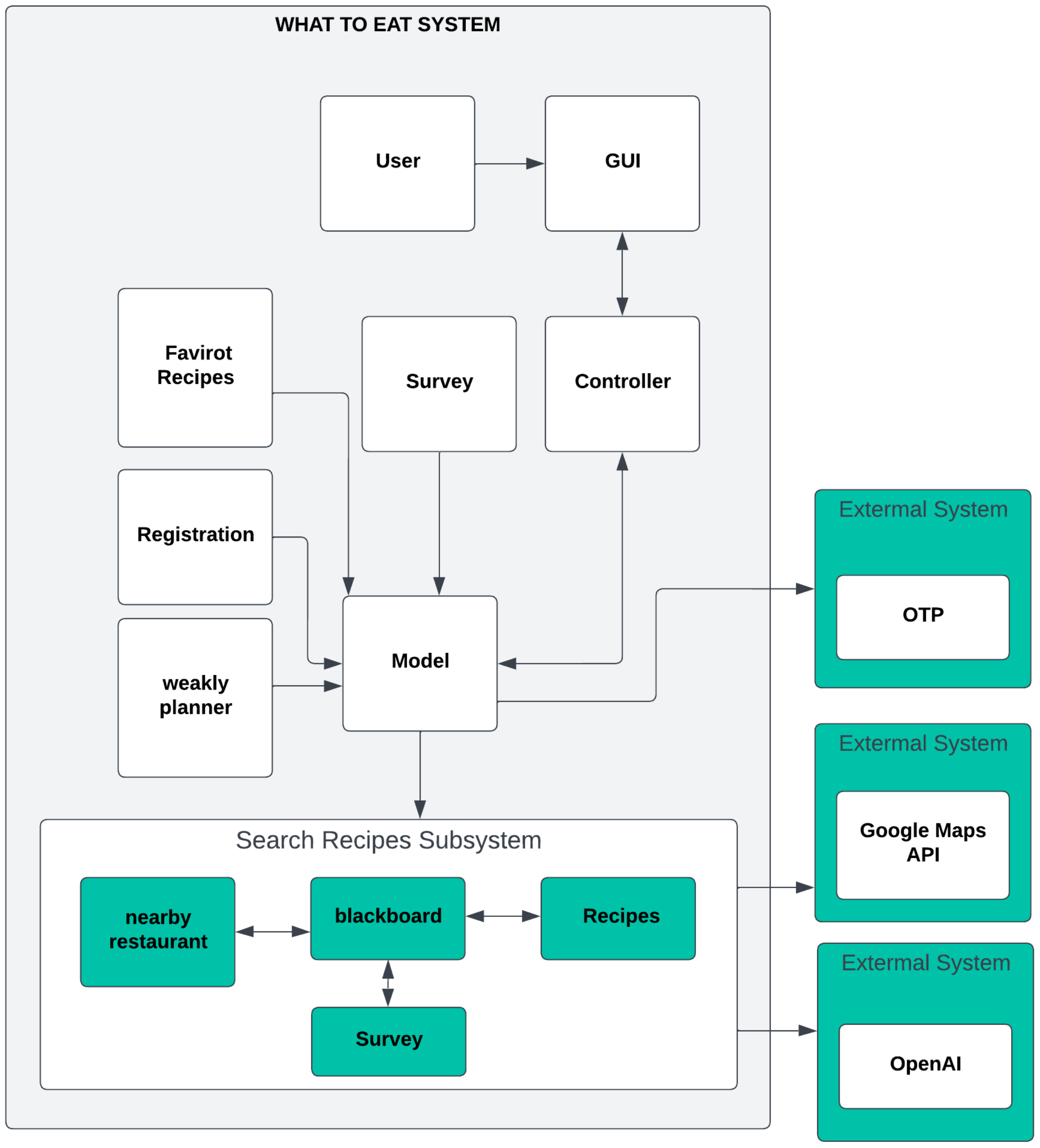


Figure 2: Domain model of What to Eat System.

**What to Eat system is divided into these subsystems:**

- **User:** a registered user who can access the system and all its functionalities.
- **GUI:** The view component which displays the GUI.
- **Controller:** The controller component which controls the way the user interacts with the system.
- **Model:** the model component which contains the main system modules.
- **Location:** a public space that is generally open and accessible to people.
- **Registration:** a component responsible for registering/signing in users.
- **Weakly Planner:** a schedule of your meal plans and activities over a week.
- **OTP "External System":** One-time password (OTP) systems provide a mechanism for logging on to a network or service using a unique password that can only be used once. [1]
- **Google maps API "External System":** offers location services for motorists that use the Global Positioning System (GPS) location. [12]
- **OpenAI "External System":** used to analysis and suggest recipes by user prompt.
- **Search Recipes Sub-system:** is subsystem that suggests recipes to user.
- **Survey:** is component contains form that user fills it to filter any illegal ingredient.
- **Blackboard:** Will provide recipe that matches with user requirement "use survey and favorite Recipes" and suggests nearby restaurant.
- **Nearby restaurant:** is component that suggests nearby restaurant that provide recipe.
- **Recipes:** is component contains recipes that the user saved at weakly planner.
- **Favorite Recipes:** is component add and save recipes that user want it.

## Architectural Style:

Based on our app what to eat description, we have chosen to integrate the "blackboard" concept with the Model-View-Controller (MVC) architecture for many reasons, the MVC architecture having a clear separation between the model, view, and controller makes it easier to scale and add new features or components. It is a powerful URL-mapping component using which we can build applications that have comprehensible and searchable URLs. Design for flexibility, it is usually quite easy to change the UI by changing the view, the controller, or both. Codes are easy to maintain, and they can be extended easily. The main reason to choose the MVC architecture is specifically used in applications where user interfaces are prone to data changes all the time (this is what our app needs). The MVC architecture also typically supports “look and feel” features in GUI application. We are using AI or recommendation algorithms to suggest meals, the separation of concerns in MVC can make it easier to integrate and update these algorithms as needed. Blackboard is used as a global database for sharing different information as input data, partial solutions, alternatives, and final solutions. Blackboard applications tend to have complex blackboard structures, with multiple levels of analysis or abstraction. Define the structure and format of the data that will be shared on the blackboard. This includes the Choose an appropriate storage solution for the blackboard data. This could be an Amazon DynamoDB based on what to eat app requirements. types of information, their relationships, and how they are organized. AI algorithms and recommendation engines can use this data to provide personalized meal suggestions.

## High level:

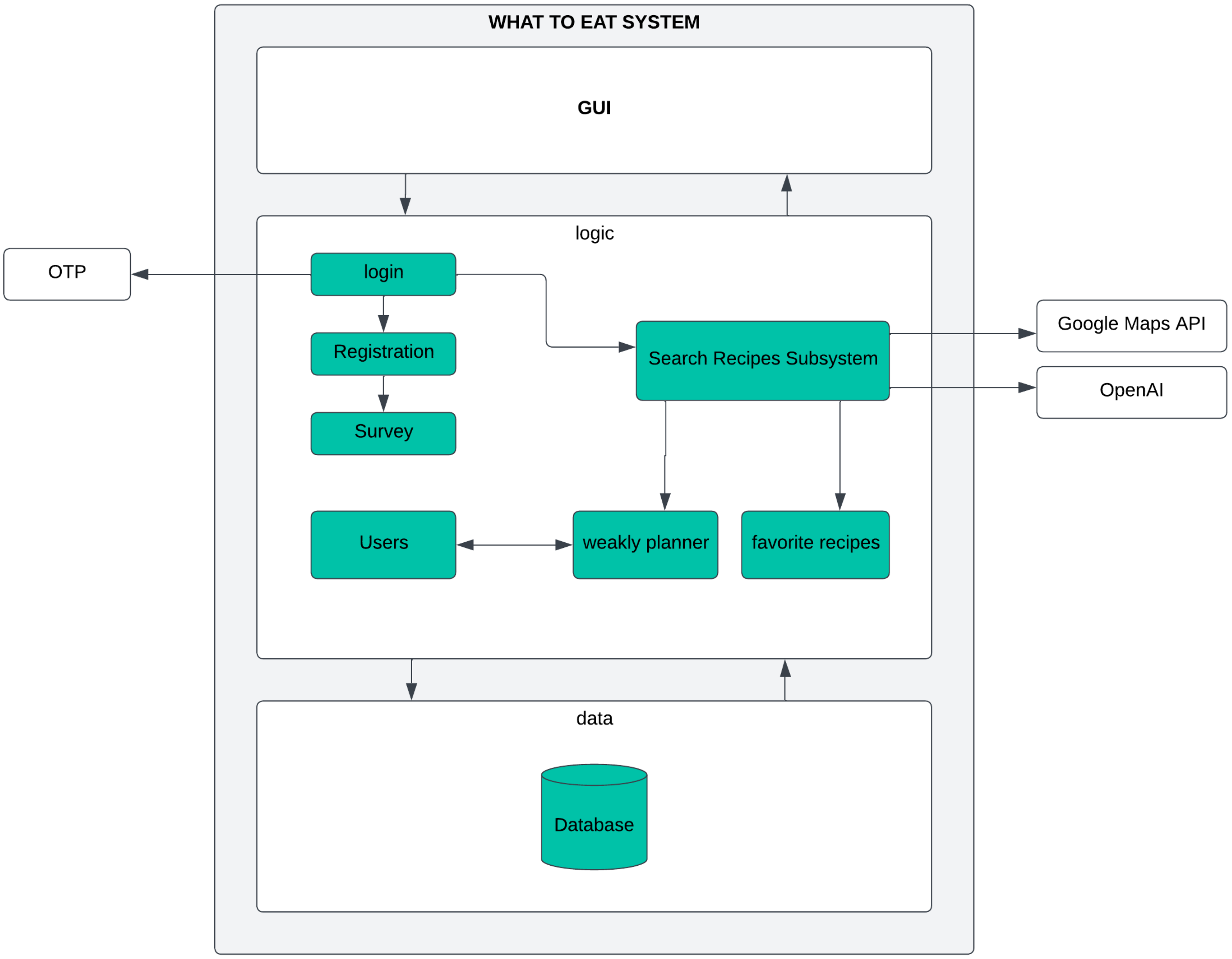


Figure 3: High Level diagram of What to Eat System.

*Note, we describe* ***Search Recipes Sub-System*** *at Domian Model.*

## Class Diagram:

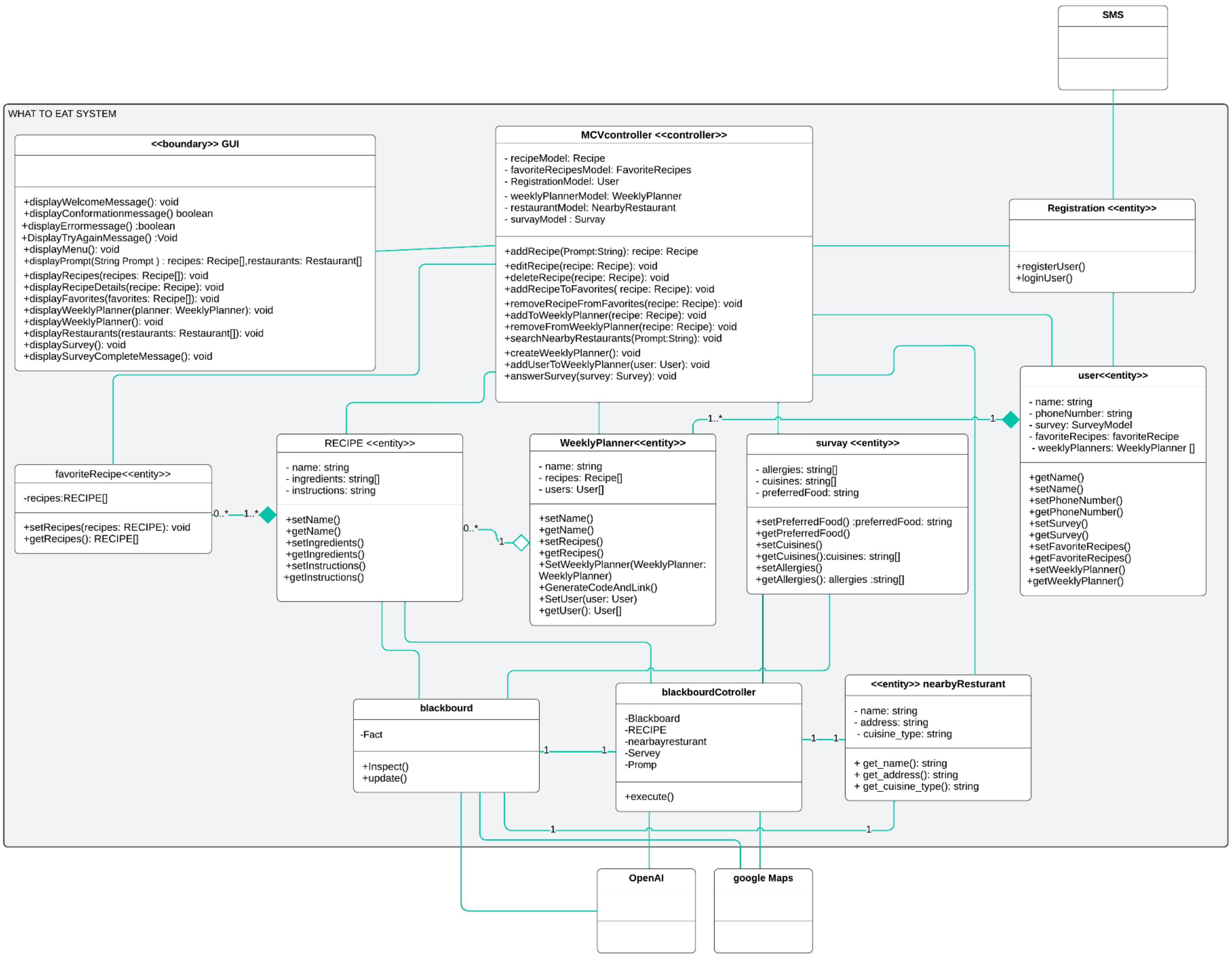

SMS
WHAT TO EAT SYSTEM
<<boundary>> GUI
+displayWelcomeMessage(): void
+displayConformationmessage() boolean
+displayErrormessage() :boolean
+DisplayTryAgainMessage() :Void
+displayMenu(): void
+displayPrompt(String Prompt ) : recipes: Recipe[],restaurants: Restaurant[]
+displayRecipes(recipes: Recipe[]): void
+displayRecipeDetails(recipe: Recipe): void
+displayFavorites(favorites: Recipe[]): void
+displayWeeklyPlanner(planner: WeeklyPlanner): void
+displayWeeklyPlanner(): void
+displayRestaurants(restaurants: Restaurant[]): void
+displaySurvey(): void
+displaySurveyCompleteMessage(): void
MCVcontroller <<controller>>
- recipeModel: Recipe
- favoriteRecipesModel: FavoriteRecipes
- RegistrationModel: User
- weeklyPlannerModel: WeeklyPlanner
- restaurantModel: NearbyRestaurant
- survayModel : Survay
+addRecipe(Prompt:String): recipe: Recipe
+editRecipe(recipe: Recipe): void
+deleteRecipe(recipe: Recipe): void
+addRecipeToFavorites( recipe: Recipe): void
+removeRecipeFromFavorites(recipe: Recipe): void
+addToWeeklyPlanner(recipe: Recipe): void
+removeFromWeeklyPlanner(recipe: Recipe): void
+searchNearbyRestaurants(Prompt:String): void
+createWeeklyPlanner(): void
+addUserToWeeklyPlanner(user: User): void
+answerSurvey(survey: Survey): void
Registration <<entity>>
+registerUser()
+loginUser()
user<<entity>>
- name: string
- phoneNumber: string
- survey: SurveyModel
- favoriteRecipes: favoriteRecipe
- weeklyPlanners: WeeklyPlanner []
+getName()
+setName()
+setPhoneNumber()
+getPhoneNumber()
+setSurvey()
+getSurvey()
+setFavoriteRecipes()
+getFavoriteRecipes()
+setWeeklyPlanner()
+getWeeklyPlanner()
favoriteRecipe<<entity>>
-recipes:RECIPE[]
+setRecipes(recipes: RECIPE): void
+getRecipes(): RECIPE[]
RECIPE <<entity>>
- name: string
- ingredients: string[]
- instructions: string
+setName()
+getName()
+setIngredients()
+getIngredients()
+setInstructions()
+getInstructions()
WeeklyPlanner<<entity>>
- name: string
- recipes: Recipe[]
- users: User[]
+setName()
+getName()
+setRecipes()
+getRecipes()
+SetWeeklyPlanner(WeeklyPlanner: WeeklyPlanner)
+GenerateCodeAndLink()
+SetUser(user: User)
+getUser(): User[]
survay <<entity>>
- allergies: string[]
- cuisines: string[]
- preferredFood: string
+setPreferredFood() :preferredFood: string
+getPreferredFood()
+setCuisines()
+getCuisines():cuisines: string[]
+setAllergies()
+getAllergies(): allergies :string[]
0..*
1..*
0..*
1
1..*
1
blackbourd
-Fact
+Inspect()
+update()
blackbourdCotroller
-Blackboard
-RECIPE
-nearbayresturant
-Servey
-Promp
+execute()
<<entity>> nearbyResturant
- name: string
- address: string
- cuisine_type: string
+ get_name(): string
+ get_address(): string
+ get_cuisine_type(): string
1
1
1
1
1
1
OpenAI
google Maps

Figure 4: Class diagram of What to Eat System.

## Component Diagram:

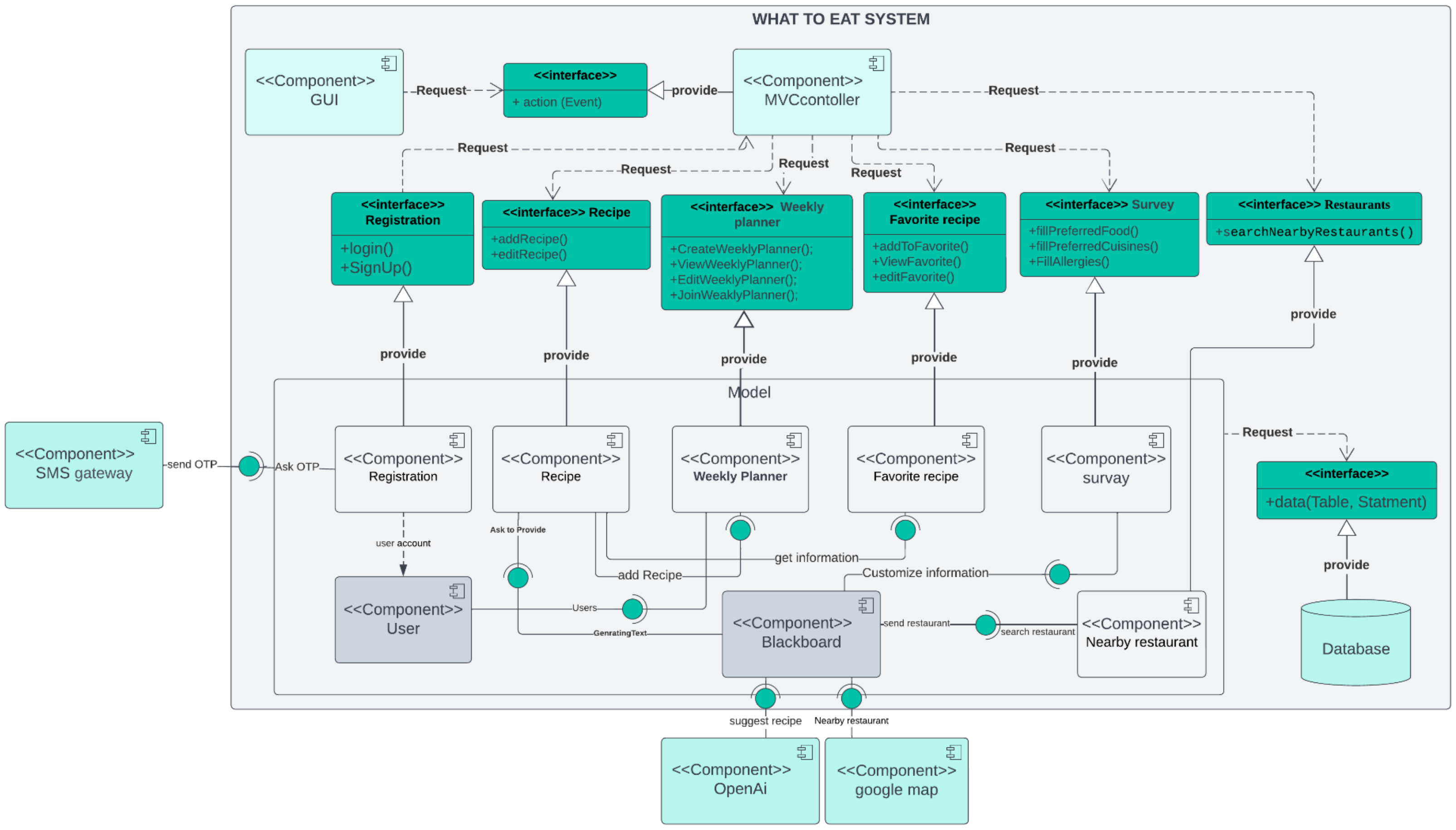


Figure 5: component diagram of What to Eat System.

| **Component Name** | **Registration** |
|---|---|
| **Description** | A component that provides sign up and log in functions. |
| **Behavior/functionality** | It makes user login, register, and saves their data securely. |
| **Connectors and Interfaces** | - Required interface with a functionality of sending OTP to the user trying to log in and register. |
| **Dependencies:** | Depends on the User to get user account and add new account. |
| **Properties/Data:** | The username and phone number. |
| **Resources:** | Amazon Database: to get user accounts from User "component " |

| **Component Name** | **SMS gateway** |
|---|---|
| **Description** | An external component that helps with log in functionality. |
| **Behavior/functionality** | Sends an SMS containing OTP used to help the user log in and Registration. |
| **Connectors and Interfaces** | Provided interface with a functionality of sending OTP. |
| **Dependencies:** | - |
| **Properties/Data:** | OTP code. |
| **Resources:** | SMS gateway server |

| **Component Name** | **Survey** |
|---|---|
| **Description** | The Survey Component collects user preferences for recipes and allergies, enhancing app personalization for tailored recipe recommendations. |
| **Behavior/functionality** | -Received the request to fill survey.<br>-sends user provided data to the amazon database for storage. |
| **Connectors and Interfaces** | **Provide** interfaces: Survey <interface> to MVC controller.<br>Customize information<interface> to blackboard. |
| **Dependencies:** | - |
| **Properties/Data:** | Optimizes app functionality by utilizing the transmitted survey data, extracting insights into user preferences for personalized recommendations. |
| **Resources:** | Amazon Database: Dedicated to storing and retrieving survey responses. |

| **Component Name** | **Favorite recipes** |
|---|---|
| **Description** | This component allows users to save their favorite recipes and easily access them in a dedicated "Favorites Recipes" section. |
| **Behavior/functionality** | -Received the request to add recipe to Favorite section.<br>-List Favorite recipes. |
| **Connectors and Interfaces** | - **Require** interfaces: Get information.<br>- **Provide** interfaces: Favorite recipes interface. |
| **Dependencies:** | depends on the Recipes component to providing Recipes information. |
| **Properties/Data:** | acts as a repository for users to store and retrieve their preferred recipes. |
| **Resources:** | - Amazon Database: Dedicated to storing and retrieving users preferred recipes within the "Favorite Recipes" component.<br>- Memory: Utilized to manage temporary data associated with the dynamic state of the "Favorite Recipes" user interface. |

| **Component Name** | **Weekly Planner** |
|---|---|
| **Description** | The Weekly Planner Component lets users collectively share and organize recipes. |
| **Behavior/functionality** | -Received the request to add recipe in weekly planner.<br>-assignee the recipe to weekly planner. |
| **Connectors and Interfaces** | - **Provide** interfaces: Weekly Planner <interface> to MVCcontroller.<br>- **Require** interfaces: Add recipes. |
| **Dependencies:** | depends on the Recipes component to providing Recipes information. |
| **Properties/Data:** | Manages shared recipes and coordinated meal plans in the weekly planner. |
| **Resources:** | Amazon Database: storing and retrieving weekly planner data, ensuring efficient management of shared recipes and coordinated meal plans. . |

| **Component Name** | **Recipe** |
|---|---|
| **Description** | represents the data structure and logic for storing and managing recipe-related information. It provides methods that enable the MVC controller to perform actions such as adding and editing recipes. |
| **Behavior/functionality** | -Provides methods for adding and editing recipes.<br>-Manages the storage and retrieval of recipe data. |
| **Connectors and Interfaces** | **Provide:** Interfaces to MVC controller to manage the recipes (add, edit, delete) and for the weekly planner functionality (add recipes to the weekly planner) to ensure inclusion in meal planning. Additionally, provides an interface for managing favorite recipes, allowing the addition of recipes to the favorites list for easy user access. |
| **Dependencies:** | The Recipe entity can be dependent on the database for data storage and retrieval. |
| **Properties/Data:** | Recipe data (e.g., name, ingredients, instructions) stored in the database. |
| **Resources:** | -CPU resources for executing recipe-related operations and calculations.<br>-Memory resources (Amazon DynamoDB server) storing recipe data during runtime. |

| **Component Name** | **Nearby restaurant** |
|---|---|
| **Description** | Nearby restaurant is component that suggests nearby restaurant that provide recipe. |
| **Behavior/functionality** | add restaurant information and store it. |
| **Connectors and Interfaces** | - Provided interface with a functionality Controller of nearby restaurant list.<br>- Required interface with a functionality of Blackboard to search nearby restaurant that match will prompt. |
| **Dependencies:** | - depend on Blackboard component to get data. |
| **Properties/Data:** | Restaurant information. |

| **Component Name** | **MVC Controller** |
|---|---|
| **Description** | The MVC (Model-View-Controller) controller component is responsible for handling user interaction and coordinating the flow of data between the model and the view. It receives input from the user via the GUI and interacts with the models entities to perform operations. |
| **Behavior/functionality** | 1-Receives in events and input from the GUI.<br>2-Interacts with the (model) to perform actions such as adding and editing recipes.<br>3. Manages the data flow between the view and the model.<br>4. Modifies the view in response to model modifications. |
| **Connectors and Interfaces** | - **Require:** An interface with the functionality of adding, editing, and deleting recipes from the *recipe (model),* managing the user's favorites list by allowing additions and removals of recipes from the *favorite recipes (model)*, handling the addition and removal of recipes from the weekly planner from *weekly planner (model)*, performing searches for nearby restaurants based on user prompts from the *nearby restaurant (model)*, creating and managing the weekly planner, adding users to the weekly planner, and handling user responses to surveys.<br>- **Provide:** An interface to the *GUI (view)* with the functionality of communication and listening for events and actions triggered by the GUI. |
| **Dependencies:** | - GUI: Depends on the GUI for receiving user input and events.<br>- Recipe: Depends on the recipe model for managing recipes (add, edit, delete).<br>- favoriteRecipes: Depends on the favoriteRecipes model for managing favorite recipes (add, remove).<br>- registration: Depends on the registration model for user-related operations such as user registration and login.<br>- weeklyPlanner: Depends on the weeklyPlanner model for managing the weekly planner (add, remove recipes). |

| | |
|---|---|
| | - Nearby restaurants: Depends on the nearby restaurants model for performing restaurant searches based on user prompts.<br>- survey: Depends on the survey model for handling user survey responses. |
| **Properties/Data:** | - The controller component primarily focuses on coordinating actions and data flow rather than storing persistent data. |
| **Resources:** | - CPU resources to manage data flow and handle user input.<br>- Memory resources to hold temporary data while controller activities are being carried out. |

| **Component Name** | **Open AI** |
|---|---|
| **Description** | **"External System":** used to analysis and suggest recipes by user prompt. |
| **Behavior/functionality** | This component is responsible to search for recipes. |
| **Connectors and Interfaces** | **Provide**: The Blackboard to select all information that needed. |
| **Dependencies:** | - |
| **Properties/Data:** | - |
| **Resources:** | CPU to execute any information needed and return it to blackboard. |

| **Component Name** | **Google map** |
|---|---|
| **Description** | **"External System":** offers location services for motorists that use the Global Positioning System (GPS) location. |
| **Behavior/functionality** | This component is responsible to add and store information about places |
| **Connectors and Interfaces** | Requires: The Blackboard to select all information that needed. |
| **Dependencies:** | - |
| **Properties/Data:** | It will store all locations of nearby restaurants and users. |
| **Resources:** | CPU resources for executing locations of users and nearby |

| | |
|---|---|
| | restaurants.<br>Memory resources (Amazon DynamoDB server) storing locations. |

| **Component Name** | **Blackboard** |
|---|---|
| **Description** | Will provide recipes that matches with user Prompt "use survey and favorite Recipes" and suggests nearby restaurant. |
| **Behavior/functionality** | This component will take information from survey and Google maps and openAI to provide recipes and nearby restaurants. |
| **Connectors and Interfaces** | Provide: recipes and nearby restaurants.<br>Require: survey, google API and the openAI |
| **Dependencies:** | Depends on the survey and the location of the user from google maps and the information from openAI. |
| **Properties/Data:** | It will generate information of recipes, nearby restaurants, locations, and survey. |
| **Resources:** | CPU resources for executing surveys and locations and openAI API.<br>Memory resources to hold all information till it needed. |

| **Component Name** | **User** |
|---|---|
| **Description** | a registered user who can access the system and all its functionalities. |
| **Behavior/functionality** | Register, create or join or leave weekly planner, view recipe, view favorite recipes, view nearby restaurants, fill the survey, prompt meal recipe. |
| **Connectors and Interfaces** | Require interface that provide the user info from registration |
| **Dependencies:** | Depends on the registration. |
| **Properties/Data:** | The username and phone number. |
| **Resources:** | Blackboard " Amazon Database ". |

## State machine Diagram

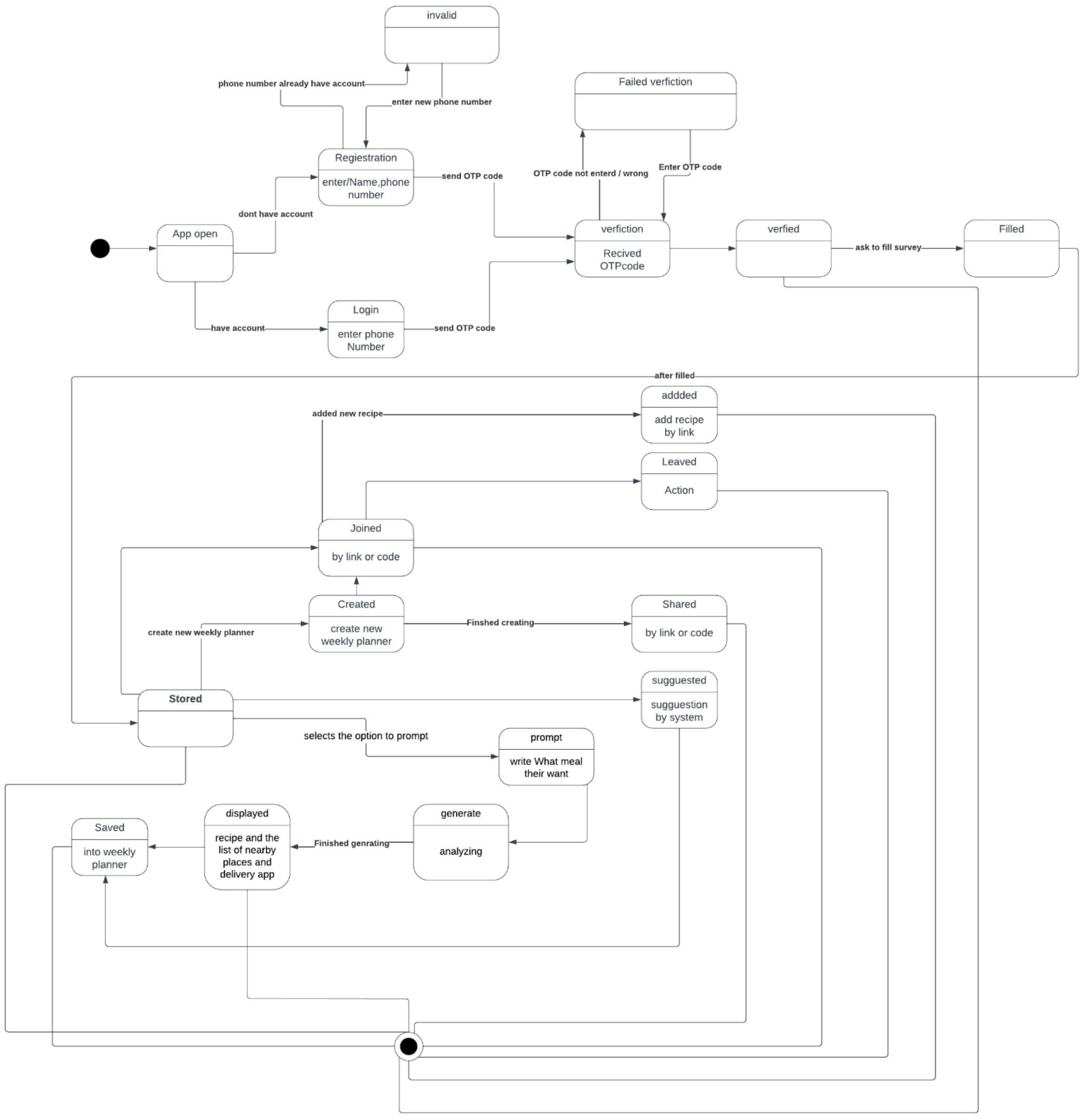


Figure 6: State machine Diagram of Whole What to Eat System.

## Deployment Diagram:

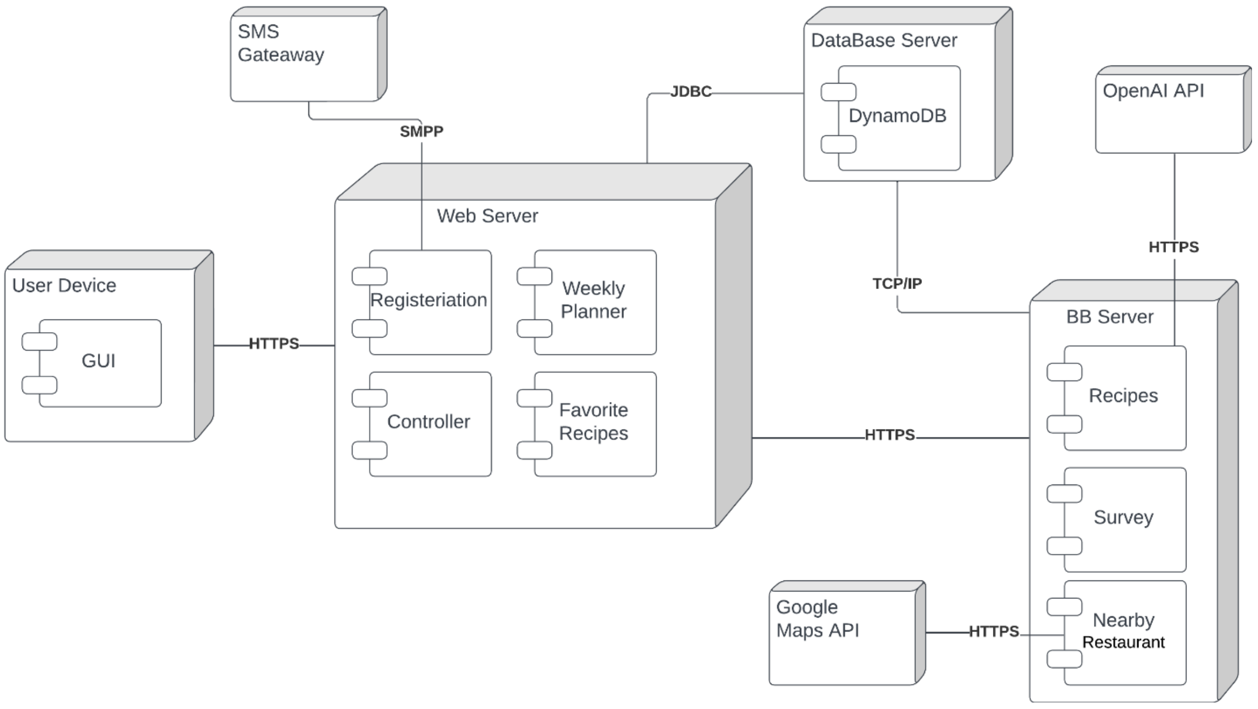


Figure 7: Deployment diagram of What to Eat System.

## Use Case Diagram:

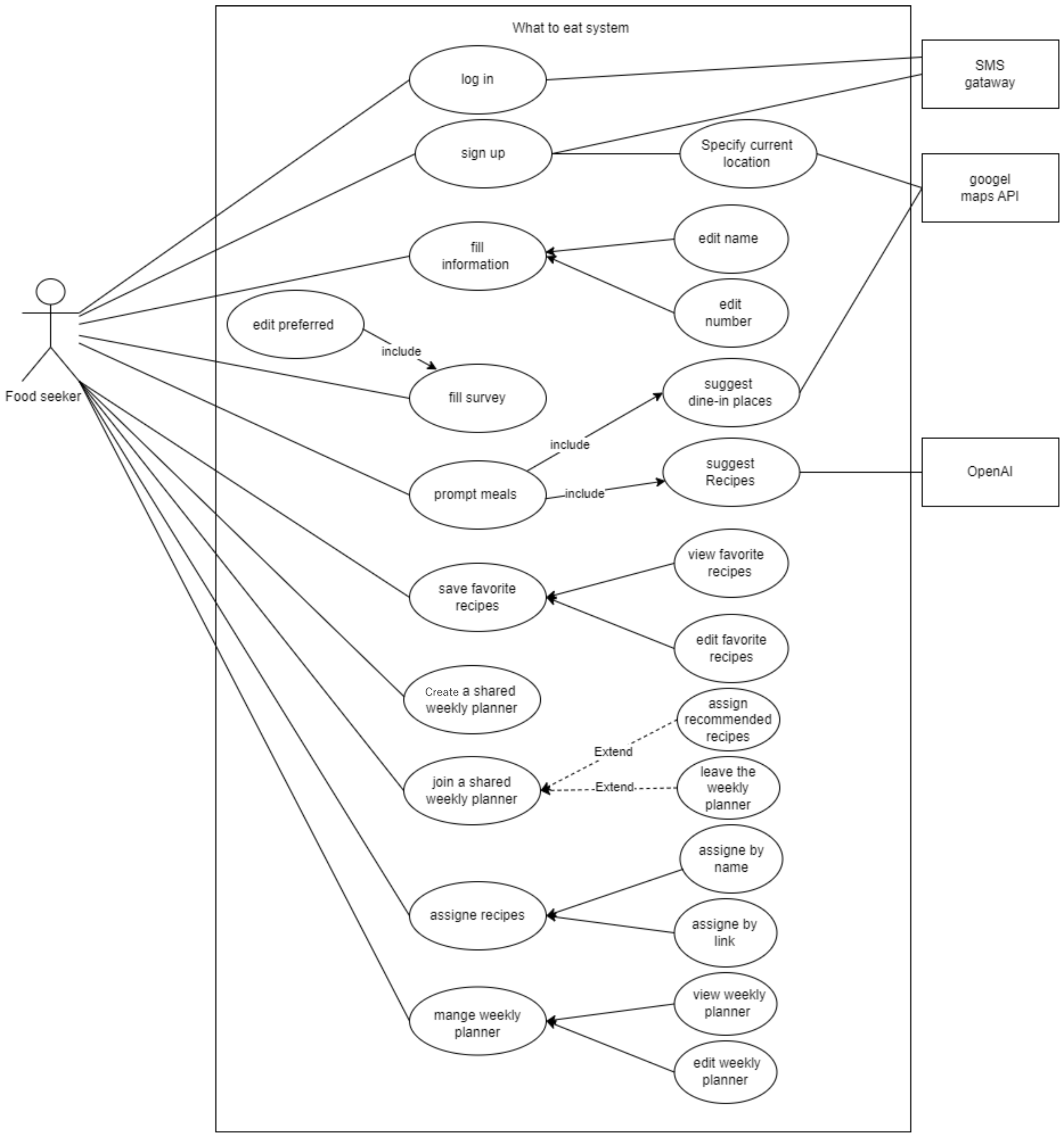


Figure 8: Use Case Diagram of What to Eat System.

## Use Case Descriptions:

| Use Case Description | | |
|---|---|---|
| System: What to eat | | |
| Use Case name: prompt meals | | |
| Primary actor: user | Secondary actor(s): Google Maps system. | |
| Description: This use case describes how a user prompts a meal. | | |
| Relationships<br>Includes: suggest dine-in places, suggest Recipes.<br>Extends: None<br>Generalization: None | | |
| Pre-conditions:<br>1. The user has logged in to the "What to Eat" app successfully.<br>2. The user has provided their preferences and updated information through a survey. | | |
| Steps: | | |
| Primary Actor (user) | System | Secondary Actor(s) (Google Maps system.) |
| 1. The user selects the option to prompt a meal or recipe suggestion.<br>2.The user input their desired meal or specific ingredients. | 3.the System analyzes the prompt and the user's updated information from the survey.<br>4.the System generates meal suggestions based on the analysis.<br>5.The system displays the recipe, including the list of ingredients and cooking instructions, to the user.<br>6. The system sends the user's current location or specified location parameters, along with the name of the meal, to the Google Maps system.<br>9.The system receives the list of nearby dining or delivery options from the Google Maps system.<br>10.the System displays the list of nearby places to dine in or deliver the meal. | 7. The Google Maps system processes the location and meal information and conducts a search for nearby restaurants that provide the specified meal or offer delivery services.<br>8. Google Maps system sends the list of nearby restaurants to the system. |
| **Alternative flows and exceptional flows:** none | | |
| Post-conditions:<br>Successful condition: the user receives meal suggestions based on the prompt and their updated information.<br>Failure condition: the user encounters an error message or a lack of suitable meal suggestions. | | |

| Use Case Description | |
|---|---|
| **System:** What to eat | |
| **Use Case name:** Save to Favorite. | |
| **Primary actor:** user | |
| **Description:** This use case describes how user save recipes to Favorite Section. | |
| Relationships<br>**Includes:** None.<br>**Extends:** None.<br>**Generalization:** View favorite recipes, edit favorite recipes. | |
| **Pre-conditions:**<br>1. The user has logged in to the "What to Eat" app successfully.<br>2. The user has already prompted recipe recommendations and received a list of suggested recipes. | |
| **Steps:** | |
| **Primary Actor (user)** | **System** |
| 1. The user will click on the favorite icon on the recipe he/she wants to add it to favorite recipes section. | 2. The system will add the recipe to the favorite recipes section. |
| **Alternative flows and exceptional flows:**<br>If the user device does not have enough storage space to save the recipe in the favorites section.<br>2.1- the system can display an error message indicating the storage issue and suggest the user free up space. | |
| **Post-conditions:**<br>**Successful condition:** the user saves meal to she/he favorite section.<br>**Failure condition:** the user encounters an error message about failed during the save operation. | |

| | |
|---|---|
| **Use Case Description** | |
| **System:** What to eat | |
| **Use Case name:** Create a shared weekly planner | |
| **Primary actor:** user | **Secondary actor(s):** - |
| **Description:** This use case describes how a user creates a shared weekly planner. | |
| **Relationships**<br>**Includes:** None<br>**Extends:** None<br>**Generalization:** None | |
| **Pre-conditions: The** user has logged in to the "What to Eat" app successfully. | |
| **Steps:** | |
| **Primary Actor (user)** | **System** |
| 1.The user selects the option to create a new weekly planner.<br>2.the user enters the name of the shared weekly planner.<br>4.The user selects the option to share the weekly planner by either sharing a link provided by the system or unique code with other users. | 3. The System generates a unique code or Link for the weekly planner. |
| **Alternative and exceptional flows:**<br>If the system fails to generate a unique code for the weekly planner:<br>3.1-System displays an error message and suggests the user try again later. | |
| **Post-conditions:**<br>**Successful condition:** the user creates a new weekly planner and shares it with others.<br>**Failure condition:** the user encounters an error message during the process, or the weekly planner sharing/validation fails. | |

## Sequence and VOPC Diagrams:

### Sequence and VOPC Prompt Meal Functionality:

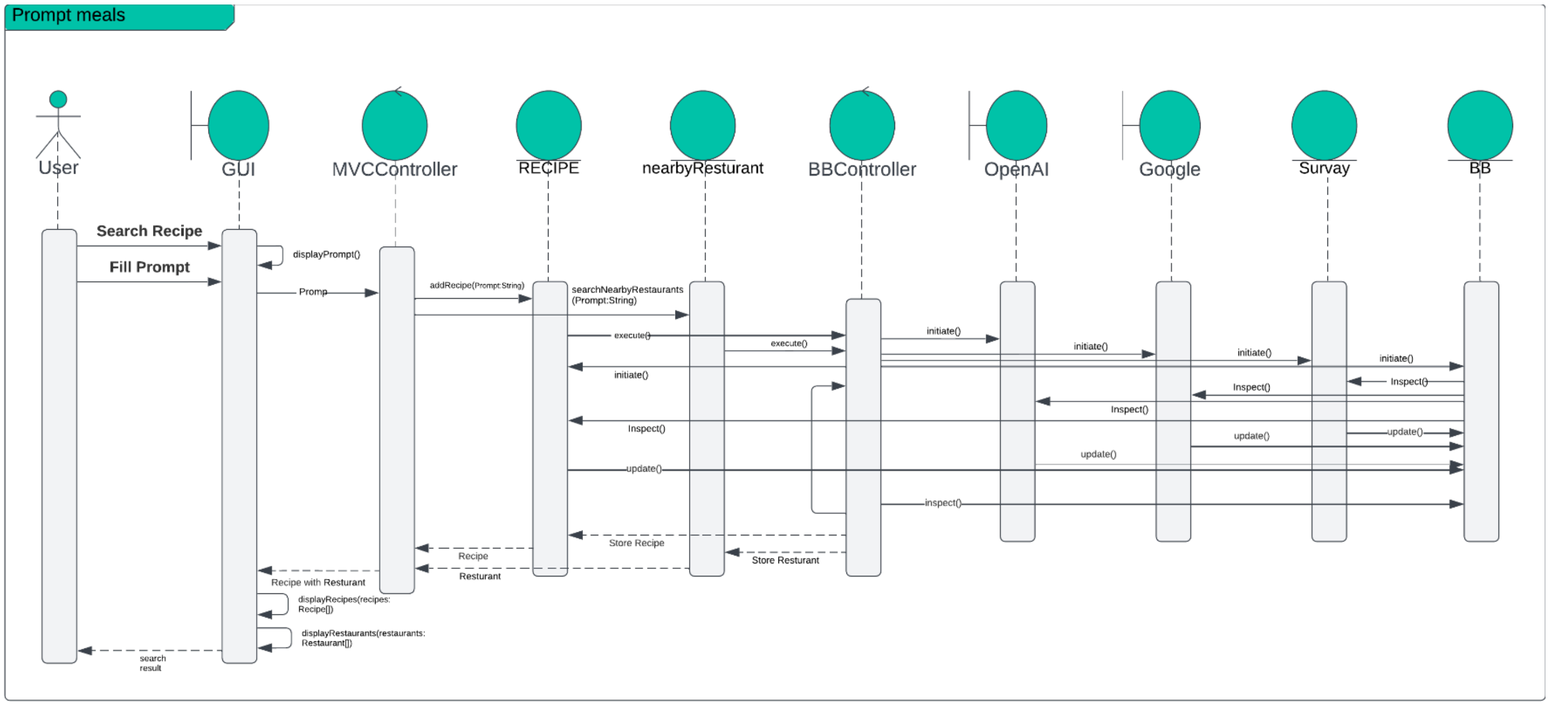


Figure 9: Sequence Diagram of Prompt Meal Functionality.

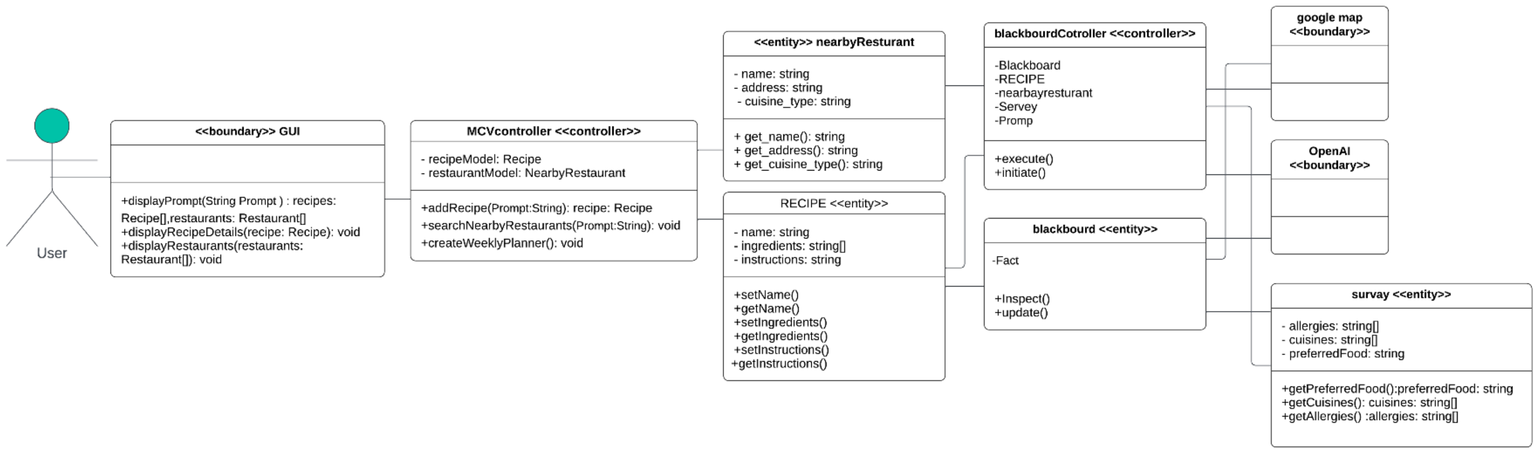

User
<<boundary>> GUI
+displayPrompt(String Prompt ) : recipes: Recipe[],restaurants: Restaurant[]
+displayRecipeDetails(recipe: Recipe): void
+displayRestaurants(restaurants: Restaurant[]): void
MCVcontroller <<controller>>
- recipeModel: Recipe
- restaurantModel: NearbyRestaurant
+addRecipe(Prompt:String): recipe: Recipe
+searchNearbyRestaurants(Prompt:String): void
+createWeeklyPlanner(): void
<<entity>> nearbyResturant
- name: string
- address: string
- cuisine_type: string
+ get_name(): string
+ get_address(): string
+ get_cuisine_type(): string
RECIPE <<entity>>
- name: string
- ingredients: string[]
- instructions: string
+setName()
+getName()
+setIngredients()
+getIngredients()
+setInstructions()
+getInstructions()
blackbourdCotroller <<controller>>
-Blackboard
-RECIPE
-nearbayresturant
-Servey
-Promp
+execute()
+initiate()
blackbourd <<entity>>
-Fact
+Inspect()
+update()
google map <<boundary>>
OpenAI <<boundary>>
survay <<entity>>
- allergies: string[]
- cuisines: string[]
- preferredFood: string
+getPreferredFood():preferredFood: string
+getCuisines(): cuisines: string[]
+getAllergies() :allergies: string[]

Figure 10: VOPC Diagram of Prompt Meal Functionality.

## Sequence and VOPC of Save to Favorite Functionality:

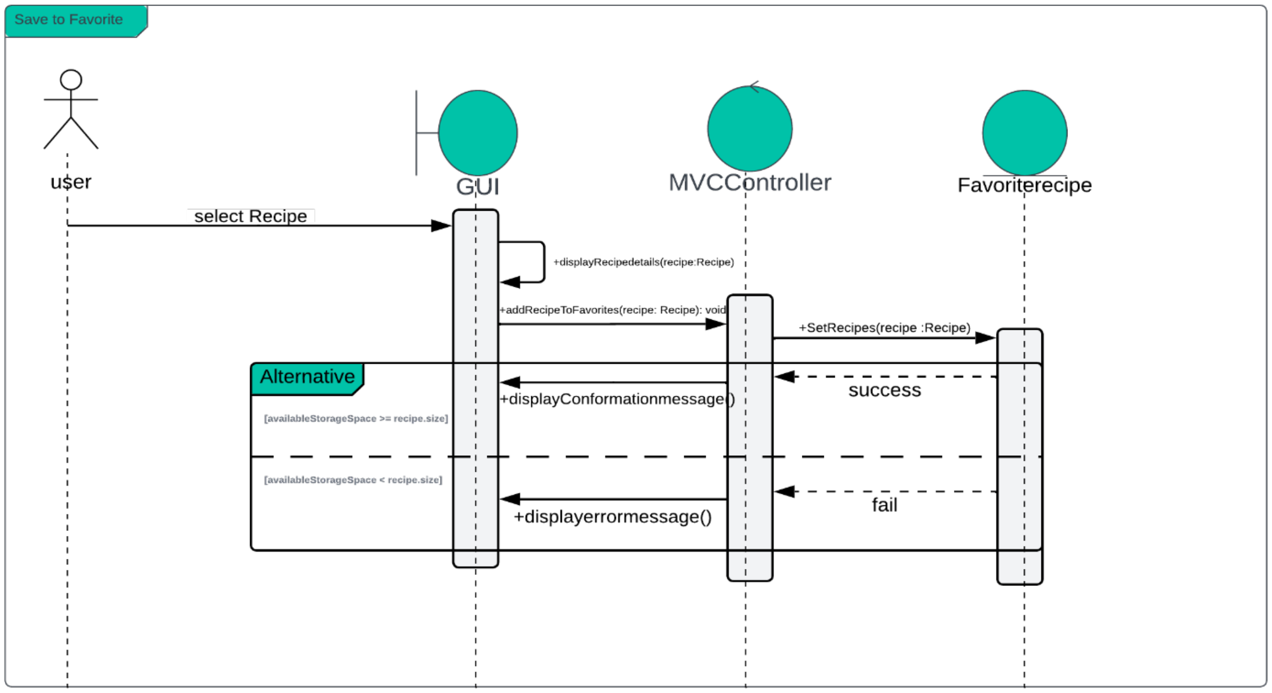

Save to Favorite
user
GUI
MVCController
Favoriterecipe
select Recipe
+displayRecipedetails(recipe:Recipe)
+addRecipeToFavorites(recipe: Recipe): void
+SetRecipes(recipe :Recipe)
Alternative
[availableStorageSpace >= recipe.size]
+displayConformationmessage()
success
[availableStorageSpace < recipe.size]
+displayerrormessage()
fail

Figure 11: Sequence Diagram of Save to Favorite Functionality.

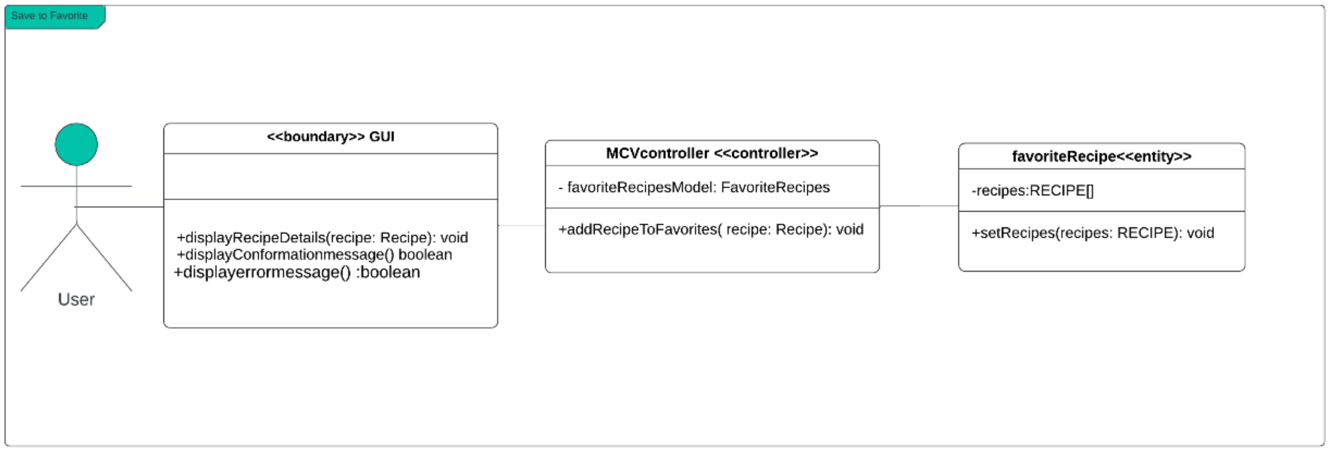

Save to Favorite
User
<<boundary>> GUI
+displayRecipeDetails(recipe: Recipe): void
+displayConformationmessage() boolean
+displayerrormessage() :boolean
MCVcontroller <<controller>>
- favoriteRecipesModel: FavoriteRecipes
+addRecipeToFavorites( recipe: Recipe): void
favoriteRecipe<<entity>>
-recipes:RECIPE[]
+setRecipes(recipes: RECIPE): void

Figure 12: VOPC Diagram of Save to Favorite Functionality.

## Sequence and VOPC of Create a shared weekly planner:

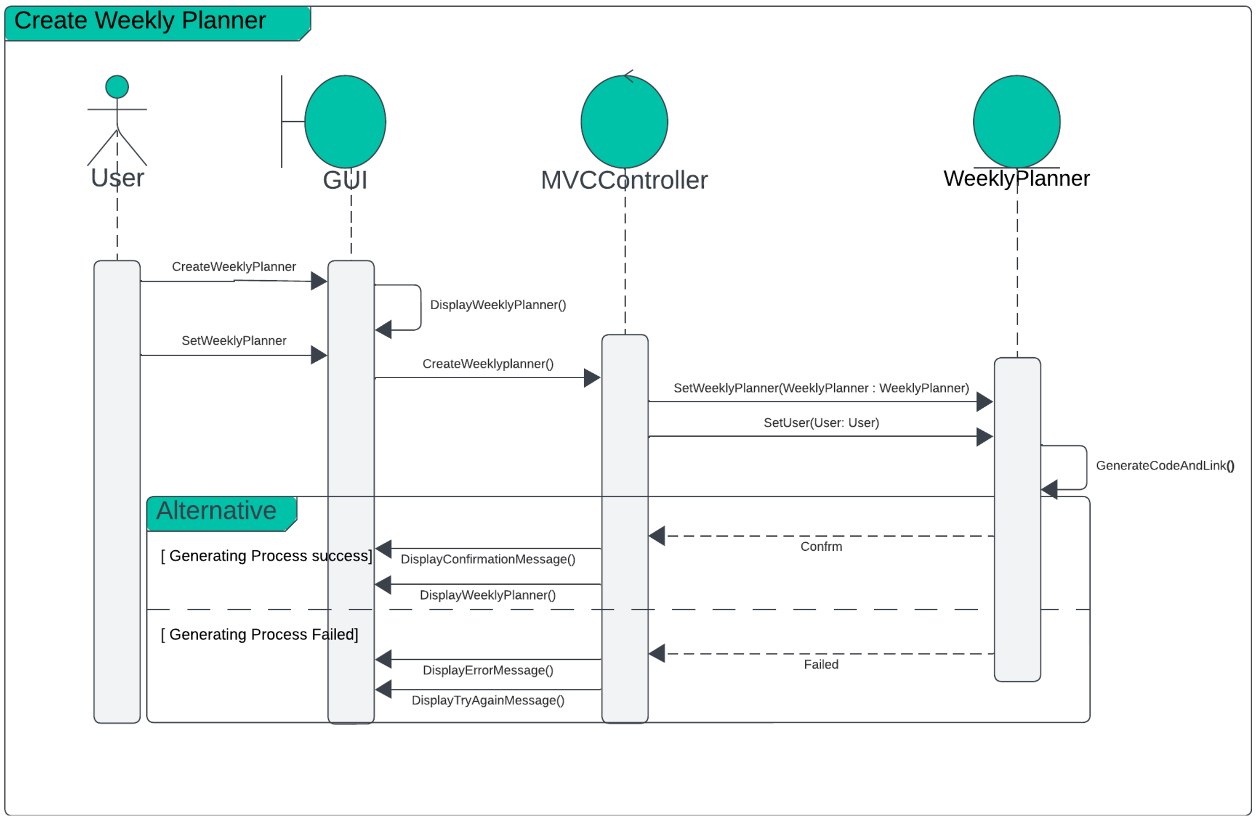


Figure 13: Sequence Diagram of Create weekly Planner Functionality.

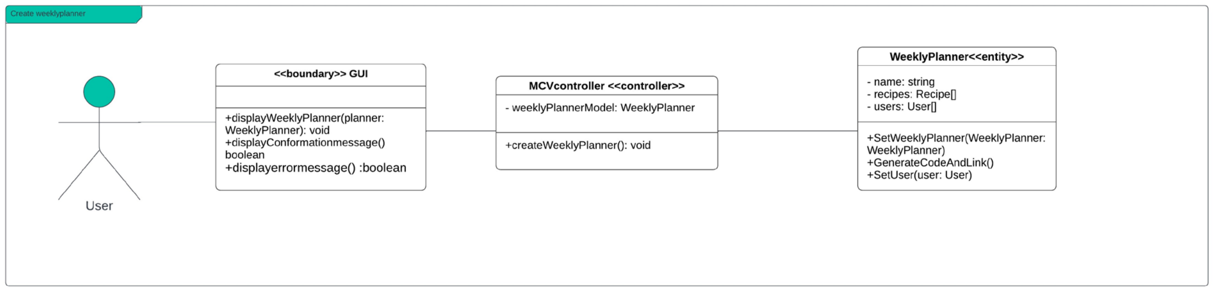

Create weeklyplanner
User
<<boundary>> GUI
+displayWeeklyPlanner(planner: WeeklyPlanner): void
+displayConformationmessage() boolean
+displayerrormessage() :boolean
MCVcontroller <<controller>>
- weeklyPlannerModel: WeeklyPlanner
+createWeeklyPlanner(): void
WeeklyPlanner<<entity>>
- name: string
- recipes: Recipe[]
- users: User[]
+SetWeeklyPlanner(WeeklyPlanner: WeeklyPlanner)
+GenerateCodeAndLink()
+SetUser(user: User)

Figure 14: VOPC diagram of Create weekly Planner Functionality.

## VOPC of general system:

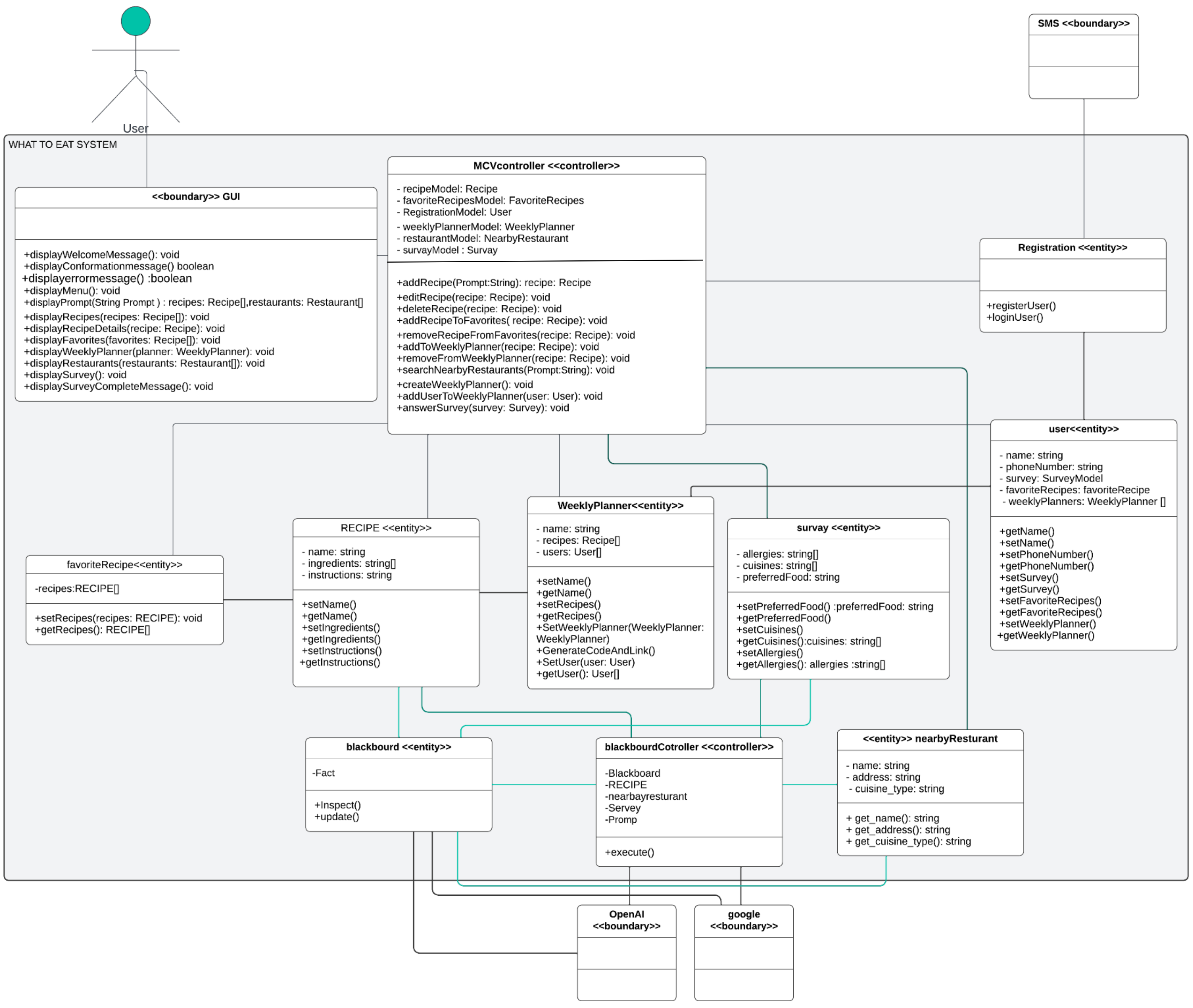

User
SMS <<boundary>>
WHAT TO EAT SYSTEM
<<boundary>> GUI
+displayWelcomeMessage(): void
+displayConformationmessage() boolean
+displayerrormessage() :boolean
+displayMenu(): void
+displayPrompt(String Prompt ) : recipes: Recipe[],restaurants: Restaurant[]
+displayRecipes(recipes: Recipe[]): void
+displayRecipeDetails(recipe: Recipe): void
+displayFavorites(favorites: Recipe[]): void
+displayWeeklyPlanner(planner: WeeklyPlanner): void
+displayRestaurants(restaurants: Restaurant[]): void
+displaySurvey(): void
+displaySurveyCompleteMessage(): void
MCVcontroller <<controller>>
- recipeModel: Recipe
- favoriteRecipesModel: FavoriteRecipes
- RegistrationModel: User
- weeklyPlannerModel: WeeklyPlanner
- restaurantModel: NearbyRestaurant
- survayModel : Survay
+addRecipe(Prompt:String): recipe: Recipe
+editRecipe(recipe: Recipe): void
+deleteRecipe(recipe: Recipe): void
+addRecipeToFavorites( recipe: Recipe): void
+removeRecipeFromFavorites(recipe: Recipe): void
+addToWeeklyPlanner(recipe: Recipe): void
+removeFromWeeklyPlanner(recipe: Recipe): void
+searchNearbyRestaurants(Prompt:String): void
+createWeeklyPlanner(): void
+addUserToWeeklyPlanner(user: User): void
+answerSurvey(survey: Survey): void
Registration <<entity>>
+registerUser()
+loginUser()
user<<entity>>
- name: string
- phoneNumber: string
- survey: SurveyModel
- favoriteRecipes: favoriteRecipe
- weeklyPlanners: WeeklyPlanner []
+getName()
+setName()
+setPhoneNumber()
+getPhoneNumber()
+setSurvey()
+getSurvey()
+setFavoriteRecipes()
+getFavoriteRecipes()
+setWeeklyPlanner()
+getWeeklyPlanner()
favoriteRecipe<<entity>>
-recipes:RECIPE[]
+setRecipes(recipes: RECIPE): void
+getRecipes(): RECIPE[]
RECIPE <<entity>>
- name: string
- ingredients: string[]
- instructions: string
+setName()
+getName()
+setIngredients()
+getIngredients()
+setInstructions()
+getInstructions()
WeeklyPlanner<<entity>>
- name: string
- recipes: Recipe[]
- users: User[]
+setName()
+getName()
+setRecipes()
+getRecipes()
+SetWeeklyPlanner(WeeklyPlanner: WeeklyPlanner)
+GenerateCodeAndLink()
+SetUser(user: User)
+getUser(): User[]
survay <<entity>>
- allergies: string[]
- cuisines: string[]
- preferredFood: string
+setPreferredFood() :preferredFood: string
+getPreferredFood()
+setCuisines()
+getCuisines():cuisines: string[]
+setAllergies()
+getAllergies(): allergies :string[]
blackbourd <<entity>>
-Fact
+Inspect()
+update()
blackbourdCotroller <<controller>>
-Blackboard
-RECIPE
-nearbayresturant
-Servey
-Promp
+execute()
<<entity>> nearbyResturant
- name: string
- address: string
- cuisine_type: string
+ get_name(): string
+ get_address(): string
+ get_cuisine_type(): string
OpenAI <<boundary>>
google <<boundary>>

Figure 15: VOPC of general of What to Eat System.

## User interfase:

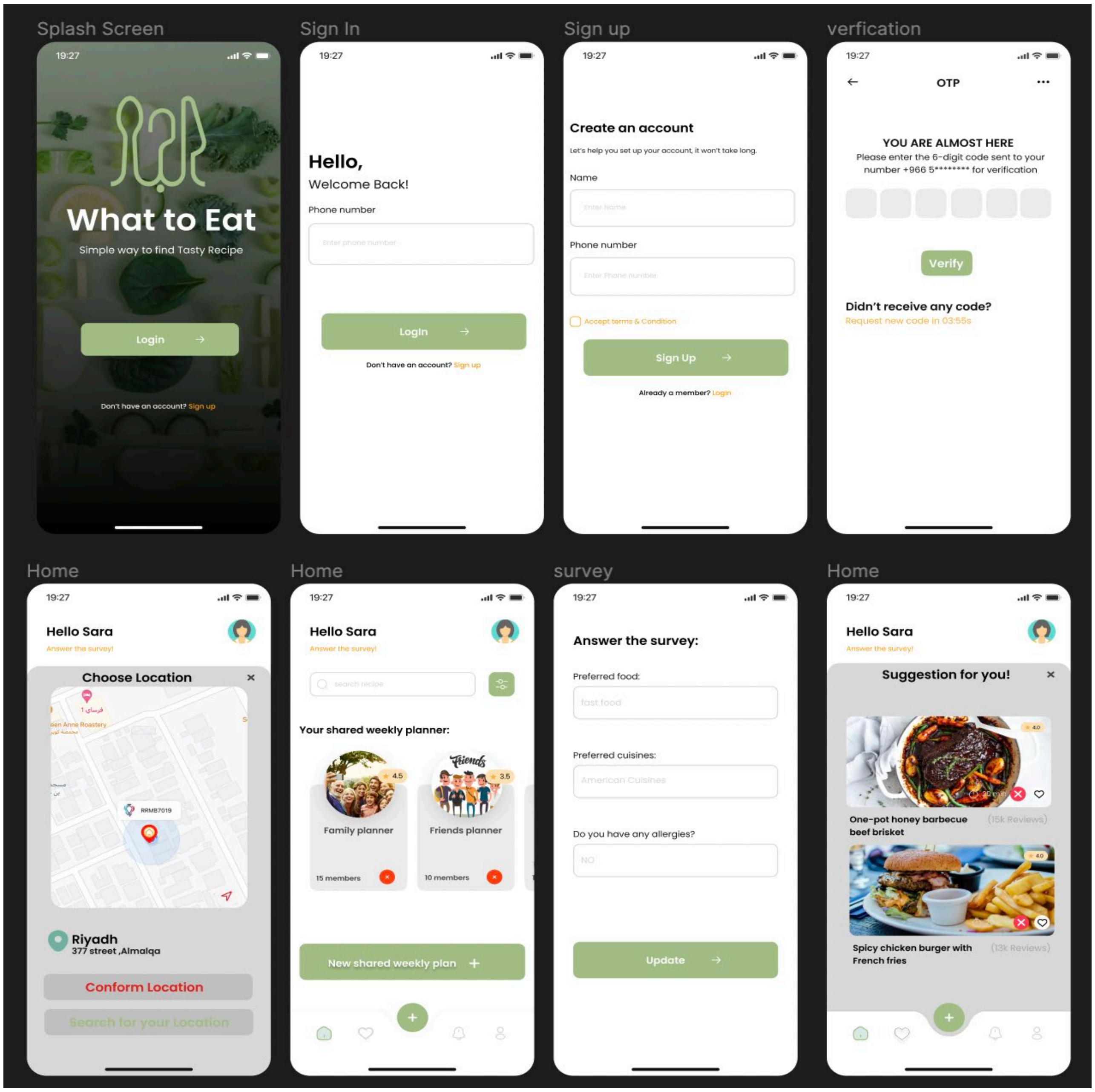


Figure 16: these interfaces show how the user login and register and answer the survey and locates his/her location and then view the home page.

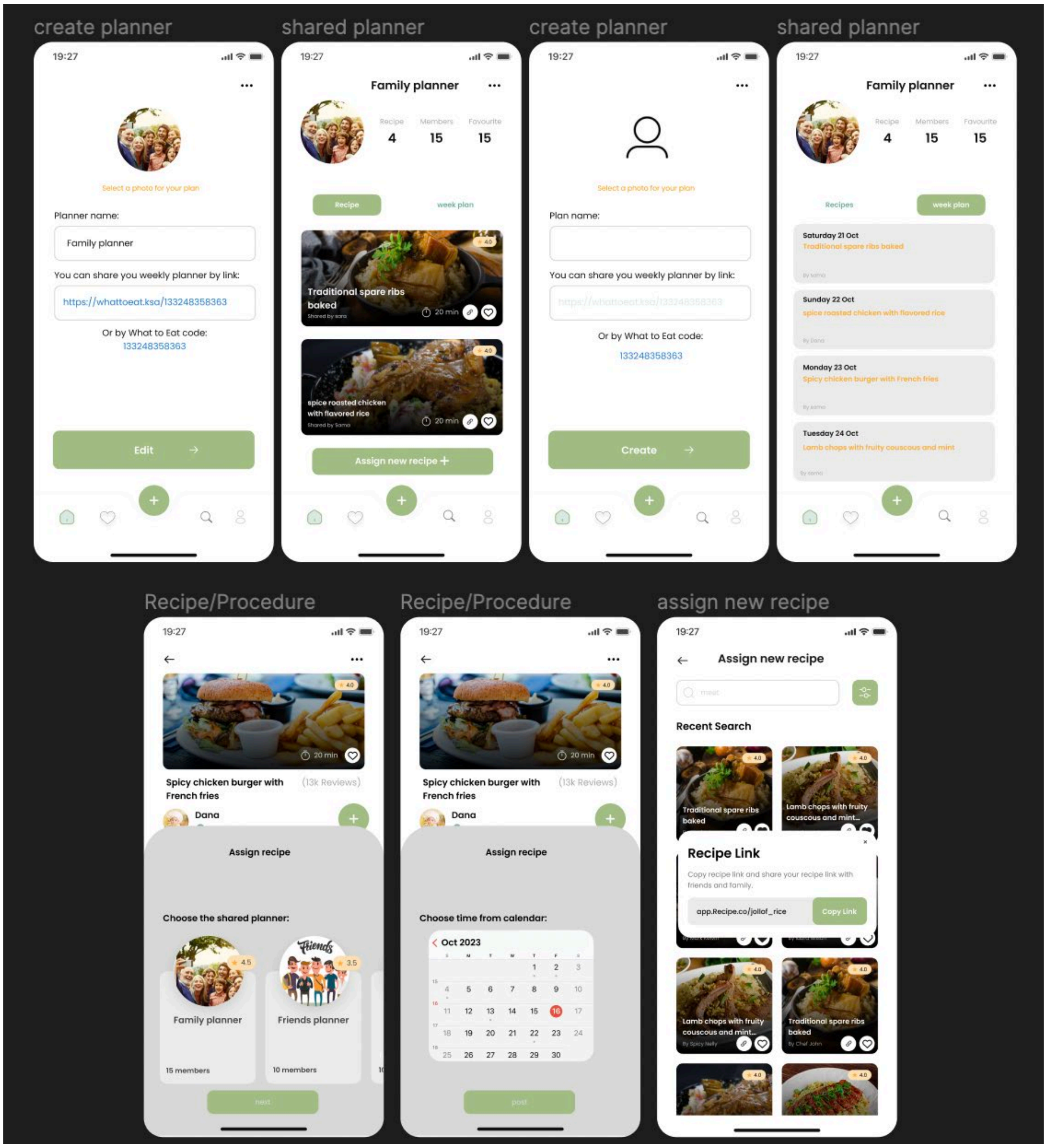

create planner
shared planner
create planner
shared planner
Select a photo for your plan
Planner name:
Family planner
You can share you weekly planner by link:
https://whattoeat.ksa/133248358363
Or by What to Eat code:
133248358363
Edit
Family planner
Recipe 4
Members 15
Favourite 15
Recipe
week plan
Traditional spare ribs baked
20 min
spice roasted chicken with flavored rice
20 min
Assign new recipe +
Plan name:
Create
Recipes
week plan
Saturday 21 Oct
Traditional spare ribs baked
Sunday 22 Oct
spice roasted chicken with flavored rice
Monday 23 Oct
Spicy chicken burger with French fries
Tuesday 24 Oct
Lamb chops with fruity couscous and mint
Recipe/Procedure
Recipe/Procedure
assign new recipe
Spicy chicken burger with French fries
(13k Reviews)
Dana
Assign recipe
Choose the shared planner:
Family planner
15 members
Friends planner
10 members
next
Choose time from calendar:
Oct 2023
post
Assign new recipe
Recent Search
Recipe Link
Copy recipe link and share your recipe link with friends and family.
app.Recipe.co/jollof_rice
Copy Link
Lamb chops with fruity couscous and mint...
Traditional spare ribs baked

Figure 17: these interfaces show the shared weekly plan interfaces and how to create new one and assign recipes to it by searching for recipe and choosing the shared weekly planner and assign the time then post it.

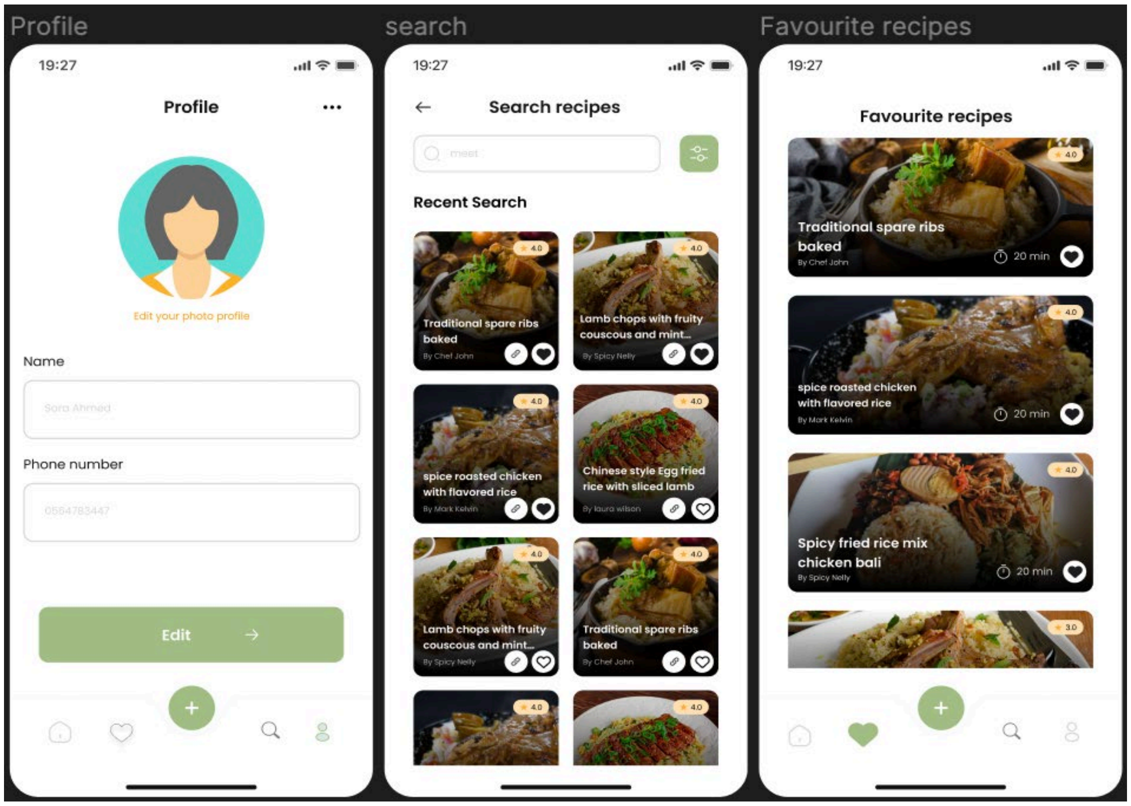

Profile
search
Favourite recipes
Profile
Edit your photo profile
Name
Sara Ahmed
Phone number
0564783447
Edit
Search recipes
meet
Recent Search
Traditional spare ribs baked
Lamb chops with fruity couscous and mint...
spice roasted chicken with flavored rice
Chinese style Egg fried rice with sliced lamb
Favourite recipes
Spicy fried rice mix chicken bali
20 min

Figure 18: these interfaces show the profile and search and favorite interfaces.

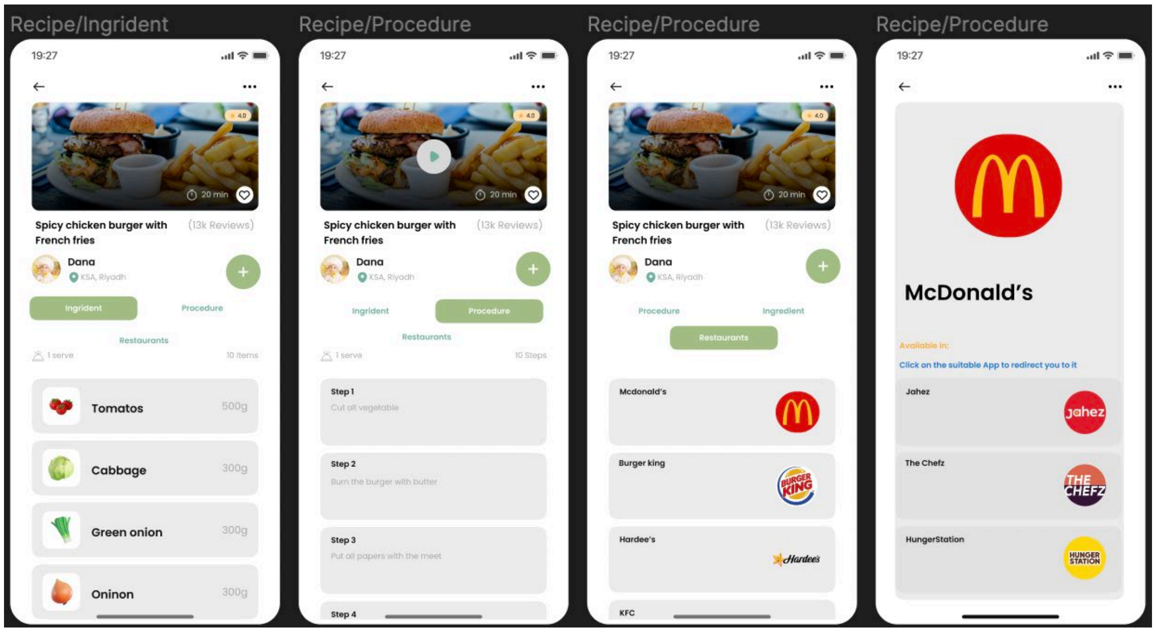


Figure 19: these interfaces show how the user views the recipe and see the ingredients and procedure and find the available restaurants to dine in or the system will direct the user to the delivery App.

## Non-Functional Properties:

We will adopt MVC and blackboard architectural styles for the need to ensure some key quality factors such as:

### Useability:

The "What to Eat" System should have a user-friendly design that can be easily understood and learned by individuals who are not familiar with technology, all within a short time limit of less than 10 minutes. The interface should be uncomplicated and instinctive, allowing users to navigate effortlessly. This simplicity should extend to the search functionality, ensuring that users can easily find the desired recipes without any difficulty.

### Performance:

Our system must provide food recommendations in less than 5 seconds. A slow system can lead to a poor user experience, resulting in frustration and decreased user engagement.

### Reliability:

For the "What to Eat" app to continue operating consistently and with system performance, reliability is essential. Because it ensures that changes or errors made in one area do not influence other areas, reliability is crucial to maintaining the independence and isolation of the Model, View, and Controller components in the Model-View-Controller architecture. Moreover, this division allows for efficient parallel processing, enabling the timely completion of tasks.[10]

### Extensions:

facilitate adapting the applications to new requirements, which is an important aspect of maintenance. Extensions can be built as a module or group of modules that can take advantage of the MVC architecture which Is this architecture style we build our system "What to Eat" on it. Any developer in our team can build an extension based on the MVC module structure, then put it publicly and others will be able to download it and append it to our application without any effect on the rest of the application.[11]

## Quality Assurance:

### Reviews:

In our system "what to eat" development team will conduct these **walkthrough reviews** at various stages of the software development lifecycle. Requirements and design walkthroughs will be performed during the initial stages, while code and documentation walkthroughs will occur during the implementation and finalization phases: [7]

1. **requirements walkthrough**: an in-depth understanding of user needs, revealing any ambiguities or gaps and leading to a precise solution.
2. **Design walkthrough**: Collaborative evaluation of database architecture, user interface, and system architecture to match users' expectations and improve usability.
3. **Code walkthrough**: Making sure coding guidelines are followed, finding errors or inefficiencies, and creating a dependable system.
4. **Documentation walkthrough**: Examining technical specifications and manuals for completeness and accuracy and giving users clear instructions to empower them.

### Verification:

In our system "what to eat" we use Code review to verify the quality assurance (QA) aspects. Code review is a systematic and collaborative process in software development, where one or more developers examine and evaluate another developer's code to identify any issues, provide feedback, and ensure that the code meets quality standards. The primary goal is to improve the maintainability, quality, and reliability of the codebase, resulting in the overall success of a project, while also promoting knowledge sharing and learning among team members.[9]

Types of code reviews we use include:
**Checklist Review:** In a checklist review, a predefined checklist is used to evaluate the code. Reviewers go through the list and check off items as they review the code.
**Ad Hoc Review:** Ad hoc reviews are informal and spontaneous. Developers may spontaneously ask team members to take a quick look at their code or discuss changes without following a formal process.

**Formal Inspection:** Formal inspections are a structured form of code review that follows a predefined process. They often involve a dedicated inspection team and detailed documentation.[9]

## Validation:

We will conduct validation to make sure the What to Eat App lives up to the expectations of its stakeholders.
Validation is the process of testing a software product. The question "Are we developing the right software product?" is answered by it. Moreover, it guarantees the detection of flaws that were overlooked throughout the verification procedure.

### 1. Black box testing:

Black box testing is primarily concerned with ensuring that the software's functioning fulfills the requirements and specifications. And it is carried out by testers who are unfamiliar with programming languages.

### 2.White box testing:

White box testing is primarily concerned with ensuring that the software's internal code is correct and efficient.
programming languages, software architecture, and design patterns are all required.[13]

## Acceptance Criteria:

*We will Acceptance criteria for how the user installs the software* ***by follows structured****:*

**Procedures**:
The installation process will be easily accessible to the user and clearly documented such as used familiar to users words and wrote by steps , also it will be distributed through an app store where installation is handled by the platform.
**Tasting**:
Various types of testing will be conducted to ensure the quality of the software:

- **User Acceptance Testing (UAT):** The software is tested in a real-world scenario by the end-user to make sure it works as expected.
- **Integration Testing:** The interaction between different components of the software is tested.

**Training**:

Training sessions can be conducted to familiarize users with the software. This could be in the form of live training sessions, online tutorials, or instructional videos.

**Documentation:**

Comprehensive technical documentation will be provided, which includes:

- **Technical Specifications:** This provides a detailed description of how the software works, its architecture, and its interaction with other systems.
- **User Manual:** This provides step-by-step instructions on how to use the software.
- **FAQs and Troubleshooting Guide:** This provides solutions to common problems that users may encounter while using the software.

## Future Considerations:

Since The system should be designed to accommodate future changes and additions. The modular and flexible nature of the MVC architecture supports this.
Possible future changes or additions to the system may include integrating additional external APIs or services to enhance recipe recommendations or delivery options, expanding social features for recipe sharing by adding chatting space, or incorporating personalized notifications and reminders for weekly planner activities. The MVC architecture facilitates the addition of new functionalities without major architectural changes, as each component can be extended or modified independently, It enables the seamless integration of external systems, ensuring that the app can leverage these services effectively.[8]

It is worth mentioning that when considering the feasibility of certain parts, it is important to identify any potential risks or challenges. For example, the integration with external APIs or services may pose risks related to data security, reliability, or compatibility. Feasibility can be checked by conducting thorough research and analysis of the specific APIs or services, considering factors such as documentation, support, and user reviews. Prototyping and testing can also help assess the feasibility and effectiveness of new features or functionalities before their full implementation.

The key parts of the system include the User component, Survey component, Recipe component, Weekly Planner component, as well as the Controller and View and the Model components. These components interact to manage user interactions, store and retrieve data, and present the user interface. Each part plays a crucial role in the overall functionality of the system.

While the current architecture provides a solid foundation, it is essential to continuously evaluate and adapt it as needed. As the system evolves, certain parts may need to be redesigned or optimized to improve performance or accommodate new requirements. Regular code reviews, architectural reviews, and feedback from users can help identify areas that may need adjustment or enhancement.

## References:

- [1] ""What is OTP and how does it relate to my Smart-ID contact information? - Smart-ID," *Smart-ID*, Mar. 12, 2021. [Online]. Available: https://www.smart-id.com/help/faq/smart-id-notifications/what-is-otp-and-how-does-it-relate-to-my-smart-id-contact-information/
- [2] CoSchedule, "What is an AI Prompt? - Ultimate Marketing Dictionary," *CoSchedule Blog*, May 17, 2023. [Online]. Available: https://coschedule.com/marketing-terms-definitions/ai-prompt
- [3] *What is Amazon DynamoDB?* (1979) *Amazon*. Available at: https://aws.amazon.com/pm/dynamodb/?trk=ea7859bf-5130-492d-935b-5c4c13c3a9fe (Accessed: 18 October 2023).
- [4] https://www.geeksforgeeks.org/mvc-framework-introduction/
- [5] https://www.projectplan365.com/
- [6] Bhadoria, S. and Mayani, P. (2023) *Benefits of integrating openai in mobile app development*, *SolGuruz*. Available at: https://solguruz.com/blog/integrating-openai-in-mobile-app-development/ (Accessed: 18 October 2023).
- [7] *Structured walkthrough* (no date) *What is Structured Walkthrough? |Professionalqa.com*. Available at: https://professionalqa.com/structured-walkthrough#:~:text=Walkthrough%20is%20one%20of%20the,the%20quality%20of%20the%20product. (Accessed: 18 October 2023).
- [8] S. Sharma, "Benefits and Drawbacks of MVC Architecture," S2 Labs, May 11, 2021. [Online]. Available: https://shreysharma.com/benefits-and-drawbacks-of-mvc-architecture/. [Accessed: October 21, 2023].
- [9] "What Is Code Review?" BrowserStack, [Online]. Available: https://www.browserstack.com/guide/what-is-code-review . [Accessed: October 20, 2023]
- [10] Soni, N. (2021) *Benefits and drawbacks of MVC Architecture*, *S2 Labs By Shrey Sharma*. Available at: https://shreysharma.com/benefits-and-drawbacks-of-mvc-architecture/ (Accessed: 21 October 2023).
- [11] Nahhas, S. (2021) MVC Architecture from Maintenance Quality Attributes Perspective, Faculty of Computing and Information Technology, King Abdulaziz University. Available at: https://www.cscjournals.org (Accessed: 21 October 2023).
- [12] TechTarget. (n.d.). Google Maps. WhatIs.com. Available: https://www.techtarget.com/whatis/definition/Google-Maps. Accessed: October 21, 2023.
- [13] BrowserStack . BrowserStack. Available: https://www.browserstack.com/. Accessed: October 21, 2023.
- [14] Figma. Available: https://www.figma.com Accessed: October 21, 2023.