CHOMPTRACK

SCS2 Team
Chester Chiow, Lee Choonggi, Daven Poon, Sheng Jia-Qi,
Ng Jia Xin, Yeo Jie Sheng

1

Introduction

- Problem
- Solution + Target Users
- API Used
- Tech Stack
- Main Functionalities
- Use Case Diagram
- Live Demo

Problem

1. **Fast-paced lifestyle in Singapore**
   - No time to keep track of their diet or prioritised balance nutrition
   - Many opt for convenient food, which lack the essential nutrients
   - Hence, **fall short of meeting daily nutritional requirements**

2. **Rising cost of living and fluctuating grocery prices**
   - **Difficult to manage a budget**

Solution: ChompTrack

Your one-stop meal planner website, designed to make healthy eating and budgeting easy. With a vast selection of easy-to-follow recipes, users can conveniently plan daily meals while tracking their nutrition intake to ensure they meet their dietary goals. ChompTrack also reflects grocery prices, allowing users to manage their food budget efficiently.

Target Audience:

- Busy working professionals and students
- Budget-conscious shoppers

API USED

What is Spoonacular?

- Recipe API
- key features
  - Vast array of Recipes
  - Nutrition Information
  - Provides real-time groceries prices

TECH STACK

Front-end

HTML

CSS

JavaScript

REST API

Back-end

Python

Flask

mysql
.connector

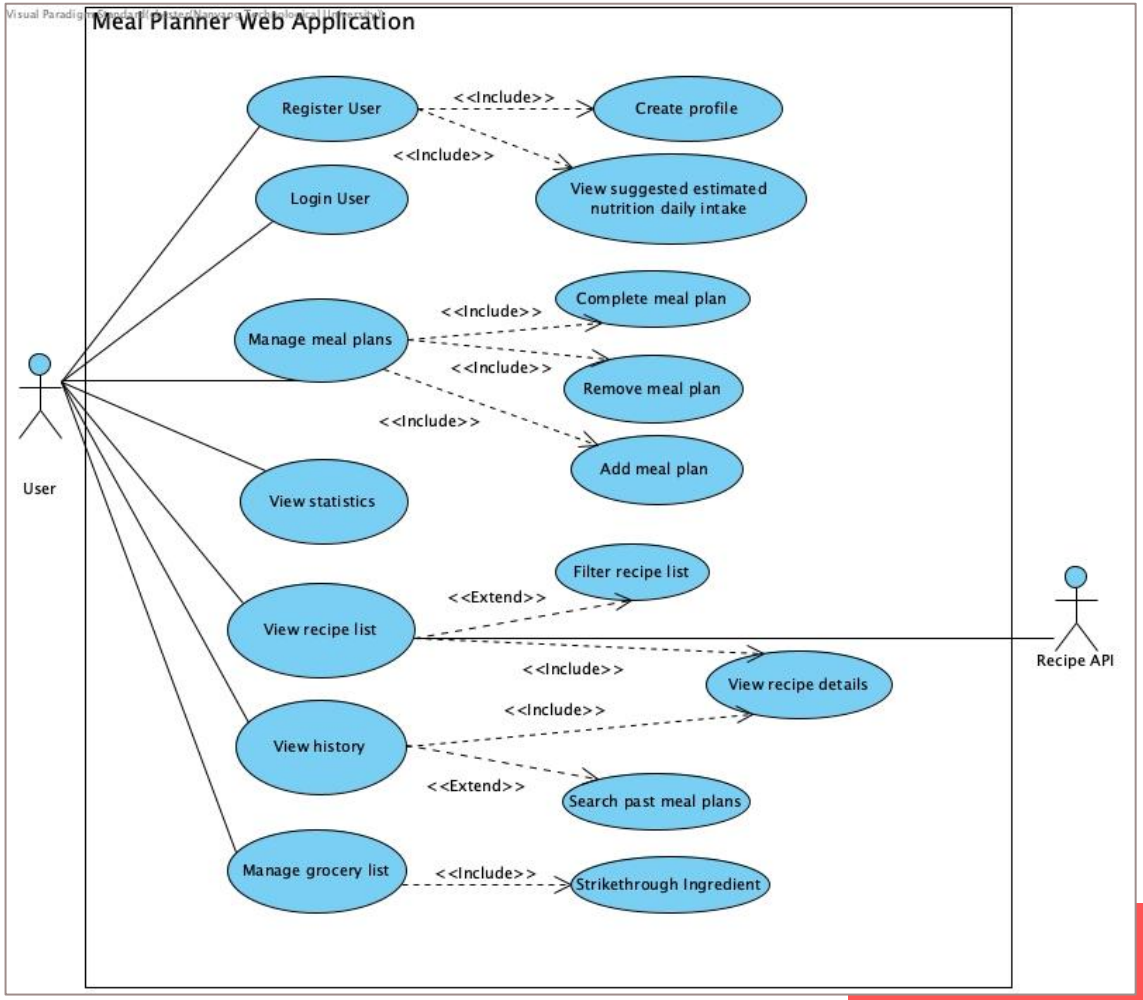
Database

MySQL

Main Functionalities

- **Authentication**
  - Login
  - Register
- **Profile Creation**
  - Calculate suggested daily nutrition intake
- **Meal Planning**
  - Add meals
  - Complete meals
  - Remove meals
- **Nutrition Tracking**
- **Recipe Selection**
  - View Recipe List
  - Filter recipe list
- **Past Recipes View**
  - Search past recipes by name
  - Filter past recipes by date
- **Grocery list**
  - Strikethrough an ingredient

Use Case Diagram



Live Demo!

2

Good SWE Practices

- Documentation
  - Pydoc
- Reusability & Refactoring

Documentation: Code Comments

- Ensure code readability
- Enhance communication and collaboration

```python
class Recipe:
    """
    A class representing a Recipe with attributes for meal types, nutrition, and cooking details.

    Attributes:
        recipe_id (int): Unique identifier for the recipe.
        recipe_name (str): Name of the recipe.
        image_link (str): URL to the image of the recipe.
        cuisine_type (str): Type of cuisine for the recipe.
        breakfast (bool): True if suitable for breakfast, False otherwise.
        lunch (bool): True if suitable for lunch, False otherwise.
        dinner (bool): True if suitable for dinner, False otherwise.
        snack (bool): True if suitable for snack, False otherwise.
        cooking_time (int): Time in minutes required to cook the recipe.
        total_price (float): Total price for the recipe in dollars.
        protein (int): Amount of protein in the recipe (grams).
        fats (int): Amount of fat in the recipe (grams).
        carbohydrates (int): Amount of carbohydrates in the recipe (grams).
        calories (int): Amount of calories in the recipe.
        recipe_instructions (str): Step-by-step cooking instructions.
        ingredients (list): List of ingredients for the recipe.
    """

    def __init__(self, recipe_id: int = None, recipe_name: str = None, image_link: str = None, cuisine_type: str = None,
                 breakfast: bool = False, lunch: bool = False, dinner: bool = False, snack: bool = False,
                 cooking_time: int = 0, total_price: float = 0.00, protein: int = 0, fats: int = 0,
                 carbohydrates: int = 0, calories: int = 0, recipe_instructions: str = "",
                 ingredients=None) -> None:
        """
        Initializes a Recipe instance with the provided details.

        Args:
            recipe_id (int, optional): Unique identifier for the recipe.
            recipe_name (str, optional): Name of the recipe.
            image_link (str, optional): URL to the image of the recipe.
            cuisine_type (str, optional): Type of cuisine for the recipe.
            breakfast (bool, optional): True if the recipe is suitable for breakfast.
            lunch (bool, optional): True if the recipe is suitable for lunch.
            dinner (bool, optional): True if the recipe is suitable for dinner.
            snack (bool, optional): True if the recipe is suitable for a snack.
            cooking_time (int, optional): Time required to cook the recipe (in minutes).
            total_price (float, optional): Total price for the recipe in dollars.
            protein (int, optional): Amount of protein in the recipe (grams).
            fats (int, optional): Amount of fat in the recipe (grams).
```

Documentation: Code Comments

- Easily and quickly view documentation for Python modules, functions, classes, and methods
- Documentation helps developers navigate the code easily, enhancing future extensibility

Recipe

Modules

pydoc

Classes

builtins.object

Recipe

class **Recipe**(builtins.object)

```
Recipe(recipe_id: Optional[int] = None, recipe_name: str = '', cuisine_type: ChompTrack.server.enums.Cuisine.Cuis

A class to represent a recipe.

Attributes:
----------
recipe_id : Optional[int]
    The unique identifier for the recipe.
recipe_name : str
    The name of the recipe.
cuisine_type : CuisineType
    The type of cuisine the recipe belongs to.
meal_type : MealType
    The type of meal (e.g., breakfast, lunch, dinner).
cooking_time : int
    The time required to cook the recipe in minutes.
total_price : float
    The estimated total price of the recipe ingredients.
recipe_instructions : str
    Instructions on how to prepare the recipe.
```

Methods defined here:

__init__(self, recipe_id: Optional[int] = None, recipe_name: str = '', cuisine_type: ChompTrack.server.enums.Cuisine.CuisineType = <C

```
Initializes a Recipe instance with the provided parameters.

Parameters:
----------
recipe_id : Optional[int], optional
    Unique identifier for the recipe (default is None).
recipe_name : str, optional
    Name of the recipe (default is empty string).
cuisine_type : CuisineType, optional
    The cuisine type (default is CuisineType.AMERICAN).
meal_type : MealType, optional
    The type of meal (default is MealType.DINNER).
cooking_time : int, optional
    Cooking time in minutes (default is 0).
total_price : float, optional
    Total price of the recipe (default is 0.0).
```

Reusability & Refactoring

Navigation bar template code reused for multiple interfaces

```html
<!-- navbar.html -->
<nav class="navbar">
    <div class="container">
        <div class="navbar-content">
            <a href="/" class="logo">
                <img src="../static/img/ChomptrackLogoWhite.png" alt="ChompTrack Logo">
                ChompTrack
            </a>
            <button class="menu-toggle" aria-label="Toggle menu">≡</button>
            <div class="nav-links">
                <a href="/my-plan">My Plan</a>
                <a href="/discover">Discover</a>
                <a href="/grocery-list">Grocery List</a>
                <a href="/history">History</a>
                <button class="logout-btn" id="logoutBtn">Logout</button>
            </div>
            <div class="user-icon">
                <img src="../static/img/userIcon.png" alt="User Icon">
            </div>
        </div>
    </div>
</nav>
```

```javascript
// loadNavbar.js
fetch('/navbar')  // Path to your navbar.html file
  .then(response => response.text())  // Get the HTML content
  .then(data => {
      document.getElementById('navbar-container').innerHTML = data;  // Insert navbar into the placeholder

      // Add event listener for menu toggle button
      document.querySelector('.menu-toggle').addEventListener('click', function() {
          document.querySelector('.nav-links').classList.toggle('active');
      });

      // Add event listener for logout button
      document.getElementById('logoutBtn').addEventListener('click', function() {
          if (confirm('Are you sure you want to log out?')) {
              // Perform logout action here
              alert('You have been logged out.');
              // Redirect to login page or perform other logout actions
              window.location.href = "/logout";
          }
      });
  })
  .catch(error => console.error('Error loading navbar:', error));  // Log any errors
```

In **myPlanInterface, groceryListInterface, discoverInterface, historyInterface, recipeInterface:**

```html
<link rel="stylesheet" href="../static/css/navbar.css">
<link rel="stylesheet" href="../static/css/discover.css">
```

```html
<script src="../static/js/loadNavbar.js"></script>
<script src="../static/js/discover.js" cahrset="utf-8"></script>
```

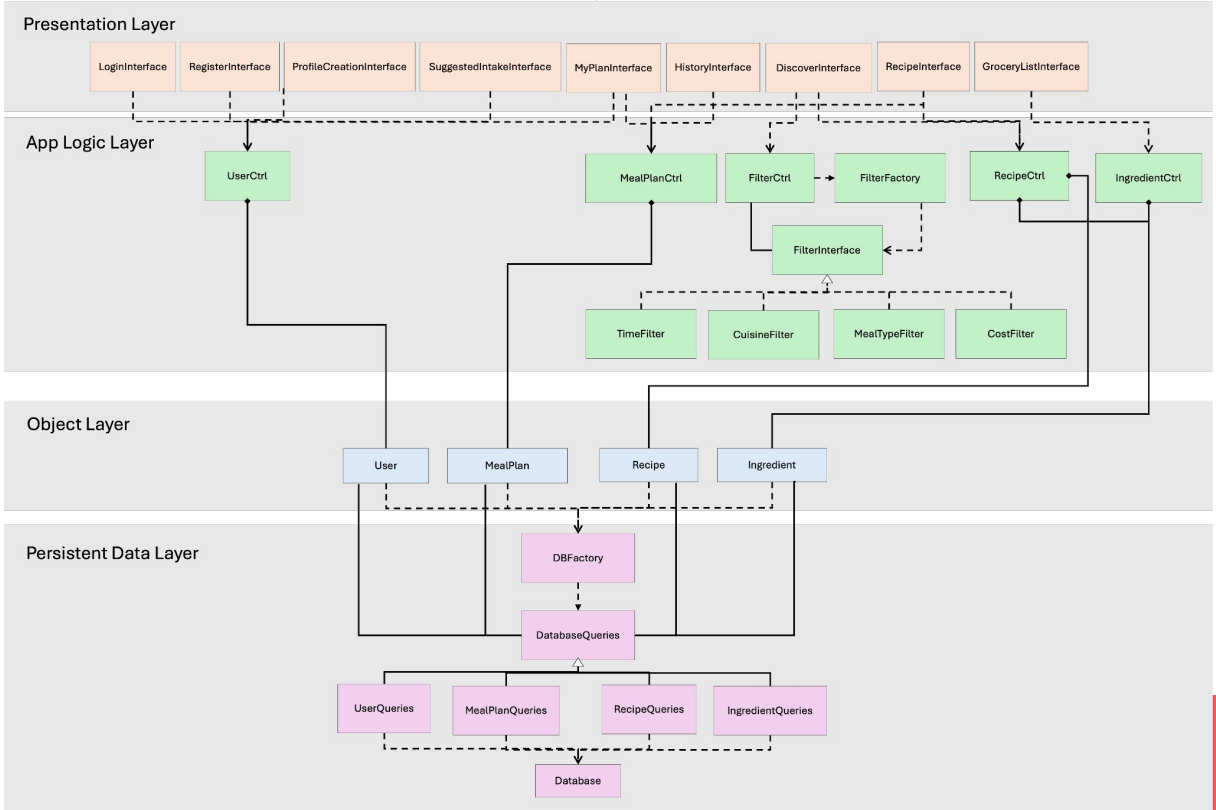
```html
<body>
    <!-- Placeholder for the navbar -->
    <div id="navbar-container"></div>

    <main class="container">
        <h1>Discover</h1>
        <hr>
```

3

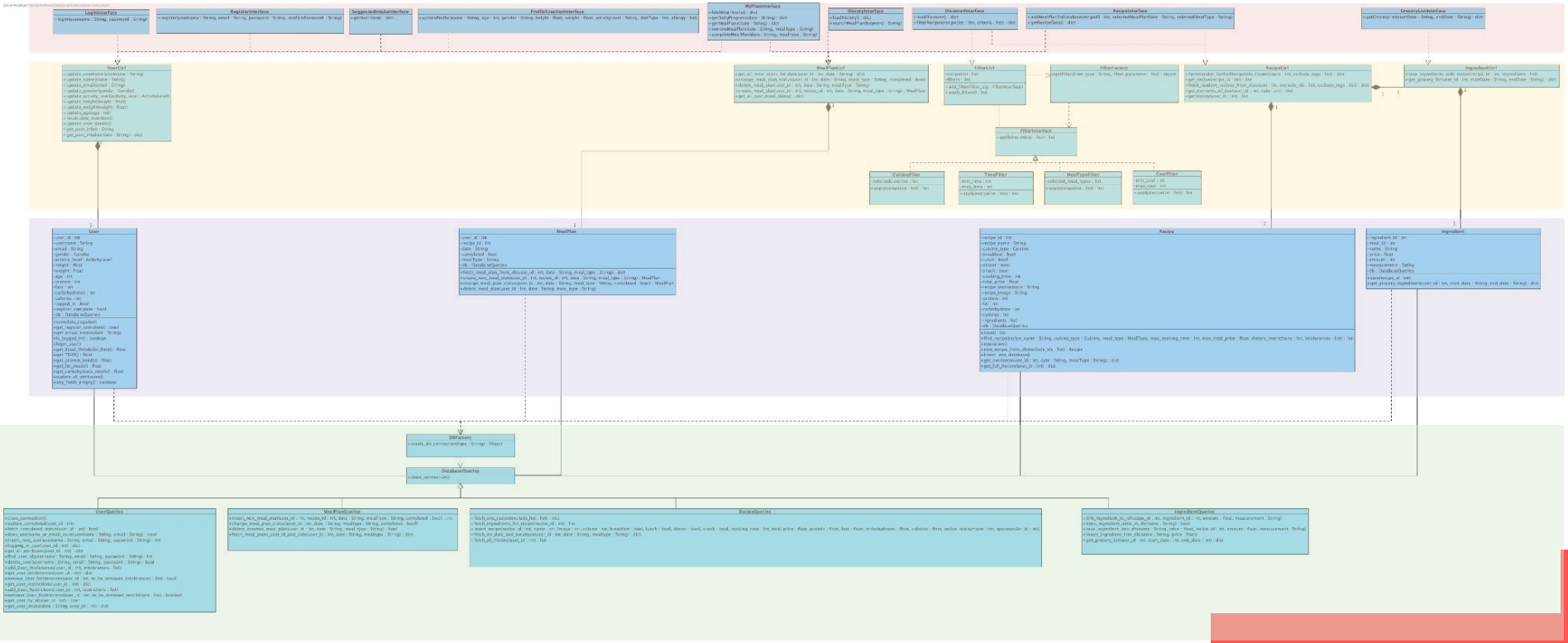
System Design

- System Architecture
- Class Diagram
- Design Pattern
- Design Principles

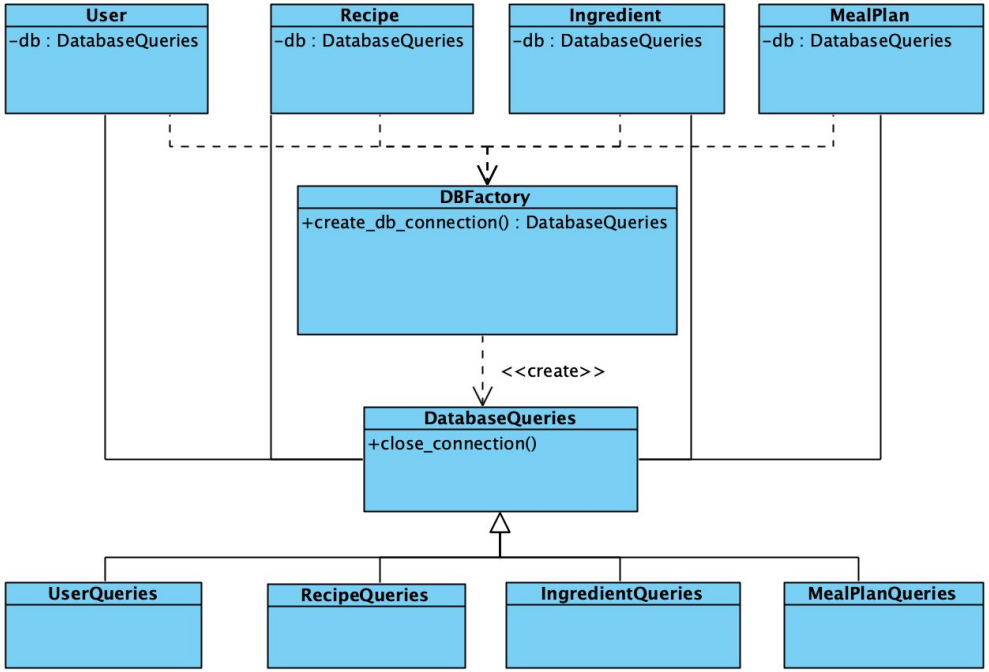
System Architecture (Layered Architecture)



Class Diagram



Factory Pattern: Database




```python
def create_db_connection(self, type: str) -> object:
    from server.DatabaseCtrl.MealPlanQueries import MealPlanQueries
    from server.DatabaseCtrl.RecipeQueries import RecipeQueries
    from server.DatabaseCtrl.UserQueries import UserQueries
    from server.DatabaseCtrl.IngredientQueries import IngredientQueries
    """
    Create a database connection based on types provided
    Args:
        type: string of the type of connection required. Ingredient, Recipe, MealPlan and User.

    Returns: Connection Object with queries to db.

    """
    if type == "Ingredient":
        return IngredientQueries()
    elif type == "Recipe":
        return RecipeQueries()
    elif type == "MealPlan":
        return MealPlanQueries()
    elif type == "User":
        return UserQueries()
    else:
        return None
```

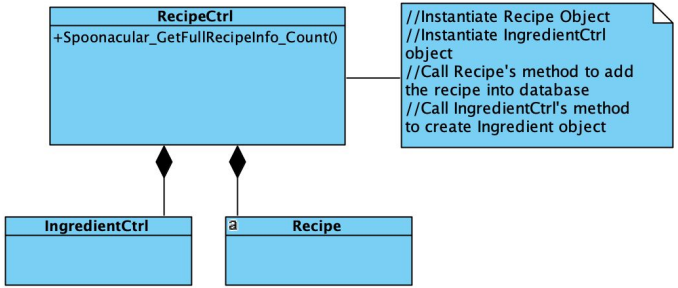
Facade Pattern: RecipeCtrl

API call method

```python
def Spoonacular_GetFullRecipeInfo_Count(self, count: int, exclude_tags: list[str] = '') -> dict:
    """
    Fetch detailed recipe information from Spoonacular API.

    :param count: int
        Number of recipes to retrieve.

    :param exclude_tags: list[str], optional
        Tags to exclude (e.g., dietary restrictions like 'vegetarian').
```

⋮

```python
recipe = Recipe(**recipe_data)
recipe.insert_into_database()
ctrl = IngredientsCtrl()
ctrl.save_ingredients_with_recipe(recipe.recipe_id, ingredients)
```



Design Principles (SOLID)

Single Responsibility Principle

Different packages containing specific classes are created. Each class is responsible for a **specific logic or task**

Open-Closed Principle

The system is **open to extension** through the use of **Factory pattern**, which leverage abstraction.

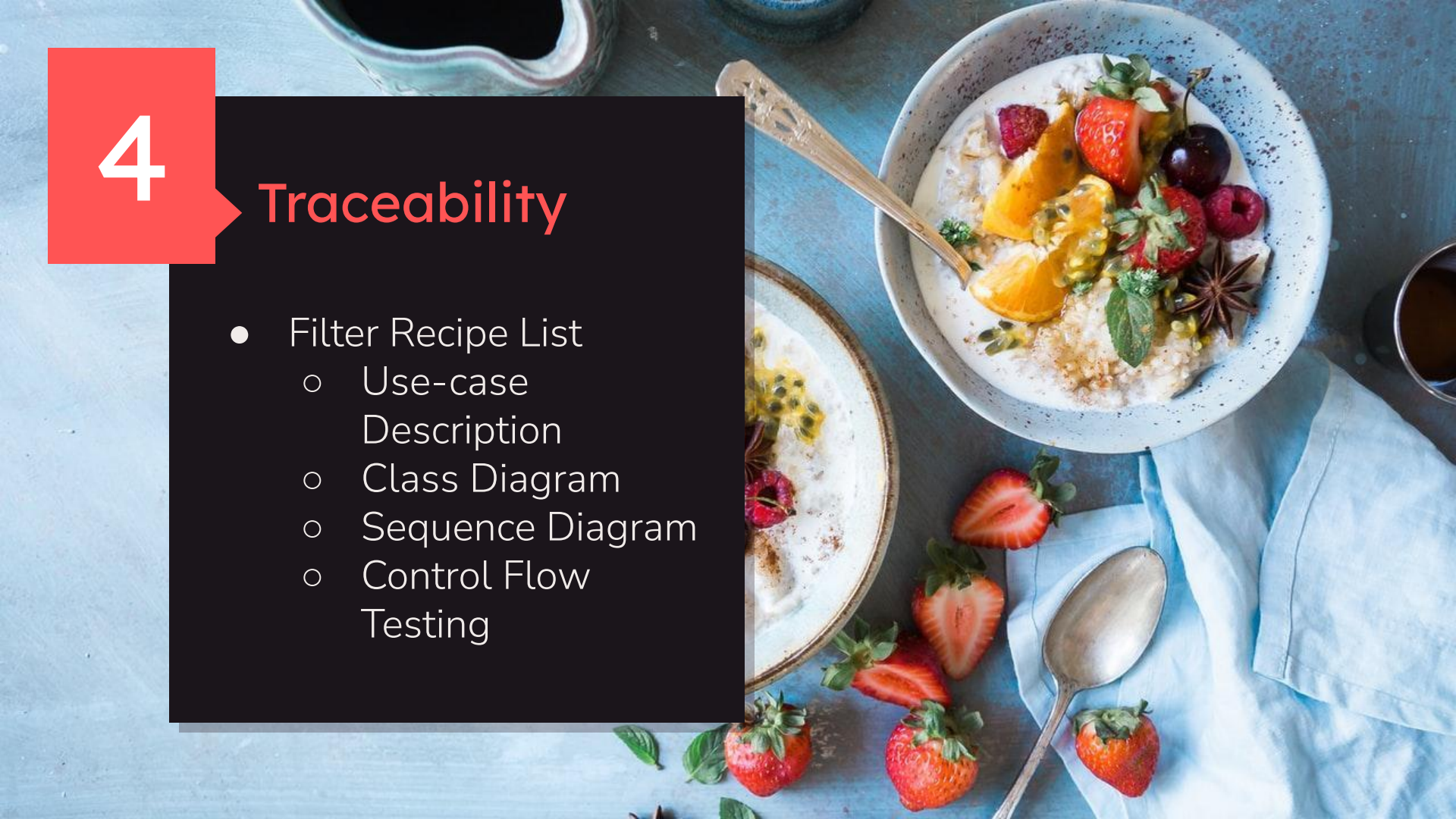
Dependency-Injection Principle

High level and low level modules depend on **abstractions** through the **use of interfaces**

4

Traceability

- Filter Recipe List
  - Use-case Description
  - Class Diagram
  - Sequence Diagram
  - Control Flow Testing

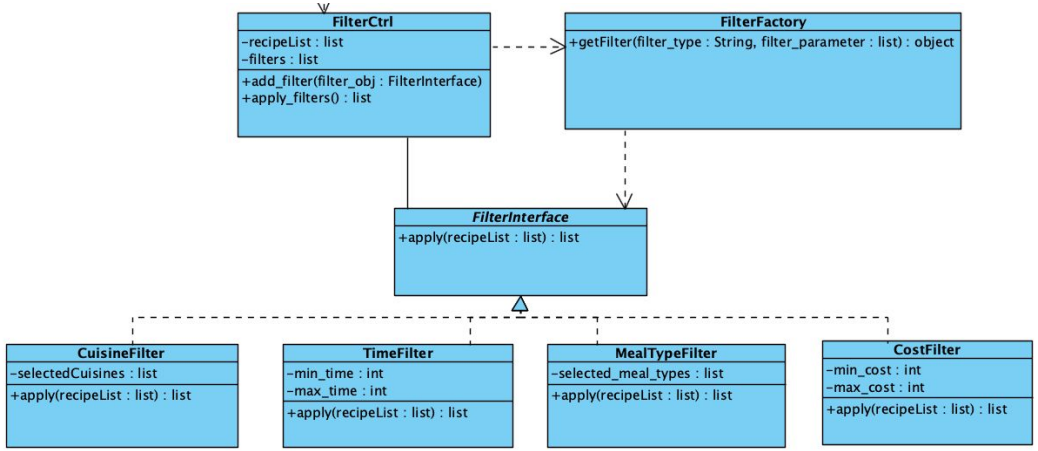


Filter Recipe List: Use Case

Use Case ID:	D2		
Use Case Name:	Filter recipe list		
Created By:	Chester Chiow	Last Updated By:	
Date Created:	30/08/2024	Date Last Updated:	

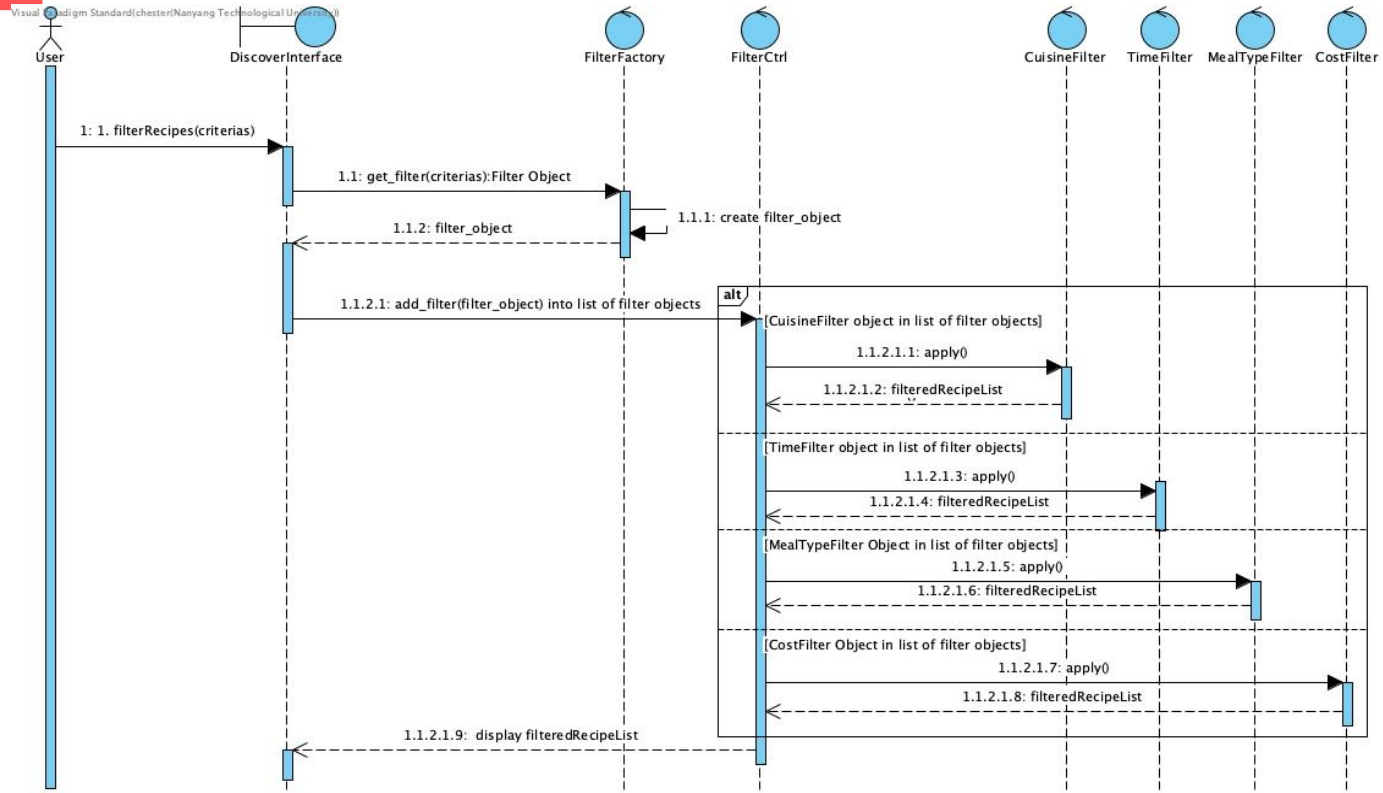
Actor:	User
Description:	User is able to filter the list of recipes.
Pre-conditions:	1. User is logged in 2. User is at the 'Discover' page
Post-conditions:	System displays an updated list of recipes according to a set of options selected by the user.
Priority:	High
Frequency of Use:	Medium
Flow of Events:	1. The system will display a list of filters (cost, cuisine, course, estimated preparation time) 2. User inputs the options to filter the list of recipes. 3. System updates the list of recipes displayed.
Alternative Flows:	Nil
Exceptions:	Nil
Includes:	Nil
Special Requirements:	Nil
Assumptions:	Nil
Notes and Issues:	Nil

Filter Recipe List: Factory Pattern

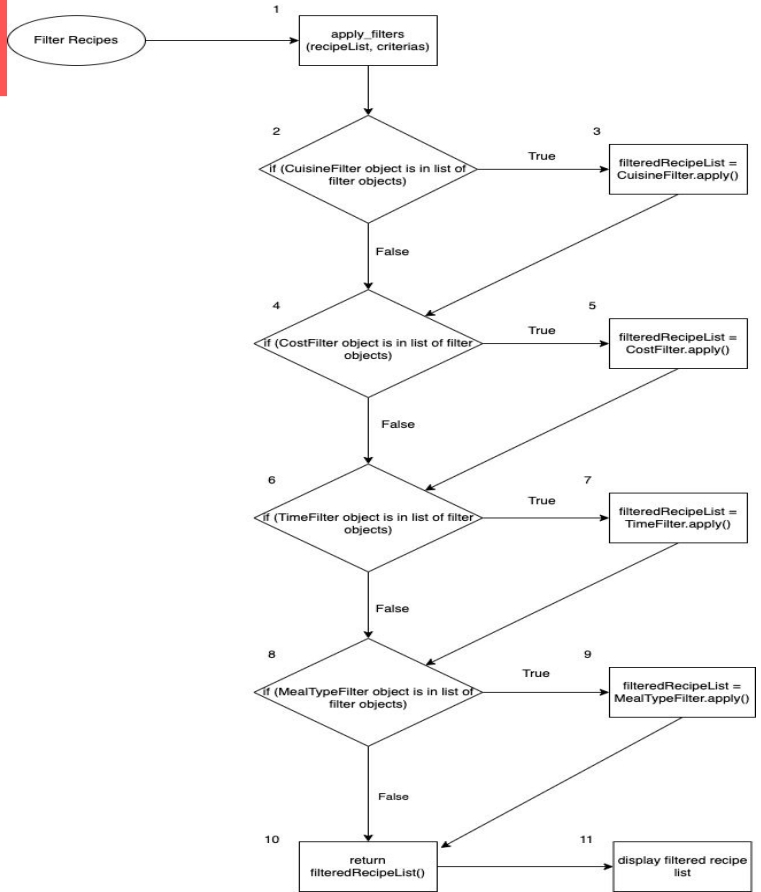



```python
def get_filter(filter_type:str, filter_parameter:list) -> object:
    """
    Creates and returns a filter object based on the specified filter type.

    Args:
        filter_type (str): The type of filter to create. Options include 'cuisine', 'mealType', 'cost', and 'time'.
        filterParameter (list): The parameters required by the specified filter type.

    Returns:
        object: An instance of a filterInterface class (e.g., `CuisineFilter`, `MealTypeFilter`, `CostFilter`, or `TimeFilter`).

    Raises:
        ValueError: If the specified filter type is not recognized.
    """
    if filter_type == 'cuisine':
        return CuisineFilter(filter_parameter)
    elif filter_type == 'mealType':
        return MealTypeFilter(filter_parameter)
    elif filter_type == 'cost':
        return CostFilter(filter_parameter)
    elif filter_type == 'time':
        return TimeFilter(filter_parameter)
    else:
        raise ValueError(f"Unknown filter type: {filter_type}")
```

Filter Recipe List: Sequence Diagram



Testing: apply_filters()



<u>Basis Path Testing</u>

Cyclomatic Complexity: | decision points | +1 = 4 + 1 = 5

Basis Paths

1. Baseline path: 1, 2, 4, 6, 8, 10, 11
2. Basis Path 2: 1, 2, 3, 4, 6, 8, 10, 11
3. Basis Path 3: 1, 2, 4, 5, 6, 8, 10, 11
4. Basis Path 4: 1, 2, 4, 6, 7, 8, 10, 11
5. Basis Path 5: 1, 2, 4, 6, 8, 9, 10, 11

Testing: apply_filters()

Basis Path	Cuisine	Cost	Estimated Preparation Time	Meal Type	Expected Output	Actual Output
1	(blank)	(blank)	(blank)	(blank)	System displays all recipes.	System displays all recipes.
2	American	(blank)	(blank)	(blank)	System displays the recipes that match the 'Cuisine' filter.	System displays the recipes that match the 'Cuisine' filter.
3	(blank)	Min $: 0 Max $: 5	(blank)	(blank)	System displays the recipes that match the 'Cost' filter.	System displays the recipes that match the 'Cost' filter.
4	(blank)	(blank)	Min(minutes): 0 Max(minutes): 50	(blank)	System displays the recipes that match the 'Estimated Preparation Time' filter.	System displays the recipes that match the 'Estimated Preparation Time' filter.
5	(blank)	(blank)	(blank)	Lunch	System displays the recipes that match the 'Meal Type' filter.	System displays the recipes that match the 'Meal Type' filter.

Thank You!