Recipify

Project Documentation

1. Introduction:

Project Overview

The purpose of this project is creating a software application that allows you to collect all your personal recipes in one space, share them with others and collect the others'. Each user will be characterized by personal data and authentication data. The recipes will contain different information about their preparation, such as ingredients and instructions, and also about their classification, such as cuisine type or nutritional information. In a second step the application will expect to have a more sophisticated user type division, with elements like 'Chefs' and 'Dietitians'. Chef will be able to share their restaurant information with recipes of some of their plates served in their restaurant. Dietitians will be able to create some diets, made with all the shared recipes in the system, and share them to different users.

Document Structure

This document is structured to follow the Unified Process phases, elucidating the various stages of the project's development. The subsequent sections detail the requirements, analysis, design, development, and finalization phases, along with corresponding artifacts generated at each stage

---

2. Requirements Engineering:

Functional Requirements

★ **User Creation**
- **Requirements ID**: FR1
- **Description**: The system provides a registration process to let the user create a personal account with the possibility to choose between different User Types.

★ **User Authentication**
- **Requirements ID**: FR2
- **Description**: The system provides a login process to let the user authenticate himself and get access to all his saved information.

★ **Personal Recipe Management**
- **Requirements ID**: FR3
- **Description**: The system provides tools that let the user create, update, view or delete his recipes.

★ **Recipe Organization**
- **Requirements ID**: FR4
- **Description**: The system provides functionality to add tags to the recipes and for creating categories that helps the user organize his recipes.

★ **Sharing Recipe**
- **Requirements ID**: FR5
- **Description**: The system provides a way to set your recipe as public, letting other users the ability to read it.

★ **Review System**
- **Requirements ID**: FR6
- **Description**: The system provides a review system for the recipe, based on a 5-star rating, in which the user can provide feedback about: taste; ease of preparation; staleness; ease of reproducibility. The user will also be able to leave a comment associated with the rating.

★ **Personalize recipe**
- **Requirements ID**: FR7
- **Description**: The system allows the user to create variations of existing recipes shared by others.

★ **Search Functionality**
- **Requirements ID**: FR8
- **Description**: The system provides search functionality working through keywords, tags and nutritional attributes.

★ **‘Post your result' Functionality**
- **Requirements ID**: FR9
- **Description**: A functionality that allows users to post photos of their trial on others recipes.

★ **Chef Functionality**
- **Requirements ID**: FR10
- **Description**: The system provides Chef Users with the ability to share its restaurant information like the position and the menu made with the recipes he has provided. Also providing a Q&A section to help people create their plates.

★ **Nutritionist Functionality**
- **Requirements ID**: FR11
- **Description**: The system provides Nutritionist Users with the ability to create a period based diet, make them public and share a copy with their clients.

Non-Functional Requirements

★ **Back-end Implementation**
- **Requirements ID**: NFR1
- **Description**: The back-end system will be implemented in Java and MySQL.

★ **Front-end Implementation**
- **Requirements ID**: NFR2
- **Description**: The Front-end system will be implemented using Java and its package Swing.

★ **Security Requirements**
- **Requirements ID**: NFR3
- **Description**: The system will securely store user user data e specifically for the password will be used an encryption method.

★ **Accessibility Requirements**
- **Requirements ID**: NFR4
- **Description**: The system should consider a comprehensive list of all the nutritional conditions people might have.

★ **Domain Constraints**
- **Requirements ID**: NFR5
- **Description**: Each user can review only others' recipe and only once per recipe.

Potential Future Functional Requirements

★ **Grocery List**

- **Requirements ID**: FFR1
- **Description**: The system automatically creates a grocery list thanks to a "Try yourself" button on each recipe. Different views for the list based on components or plates.

★ **Cooking Help Toolkit**

- **Requirements ID**: FFR2
- **Description**: The Fe System provides a set of tools such as timer or reminder to help the user in the cooking process.

★ **Recipe Export**

- **Requirements ID**: FFR3
- **Description**: The system allows the users to export recipes in various formats (PDF/txt/..)

---

3. Requirements Analysis:

Use Case Descriptions

~~**OLD Use Case OUC1**: Create Account~~ (Now contained in UC1)

- **Scope**: Recipe Management System
- **Level**: User Goal
- **Primary Actor**: Standard User
- **Stakeholders and Interests**:
  - Standard User: Interests in creating an account to access the software.
  - Chef (Secondary Actor): Interests in creating an account to access the software.
  - Dietistic (Secondary Actor): Interests in creating an account to access the software.
- **Preconditions**: User has opened the software application.
- **Success Guarantee** (or **Postconditions**): System store user's data and logs-in the user.
- **Main Success Scenario** (or **Basic Flow**):
  1. User clicks "Create an Account".
  2. System prompts user to fill personal and user data, such as username, email, password.
  3. User fills in data and clicks "Continue".
  4. System checks for data integrity and no double accounts
  5. System asks for special user type (Chef/Dietistic).
  6. User select no.
  7. System submits data and log-in the user.
- **Extensions** (or **Alternative Flow**):
  - 5a. Chef Creation:
    - 2.b User selects "Chef" as a user type
    - 3.b System prompts user to fill additional data about Restaurant
    - 4.b User fills in data and clicks "Confirm"
    - 5.b System submits data and log-in the user.
  - 5b. Dietistic Creation:
    - 2.b User selects "Dietistic" as a user type
    - 3.b System prompts user to fill additional data about his studio
    - 4.b User fills in data and clicks "Confirm"
    - 5.b System submits data and log-in the user.

- **Special Requirements**:
  - System should support various ingredient formats and multimedia content (images, videos) for recipes.
- **Frequency of Occurrence**: Different times a day for active users.

~~**OLD Use Case OUC2**:User Authentication~~(Now contained in UC1)

- **Scope**: Recipe Management System
- **Level**: User Goal
- **Primary Actor**: Standard User
- **Preconditions**: User has opened the software application.
- **Success Guarantee** (or **Postconditions**): User can use the software application.
- **Main Success Scenario** (or **Basic Flow**):
  1. User clicks “Log-in”.
  2. System prompts the user to identify himself.
  3. User fills in data and clicks “Continue”.
  4. System authenticates the user and let him use it.
- **Extensions** (or **Alternative Flow**):
  - 4a. Authentication Failed:
    - 2.b System prompts an error message and let the user retry entering data.
    - 3.b User re-fills data and clicks “continue”.
    - 4.b System tries to authenticate the user.
- **Special Requirements**:
  - System should ask to reset the password if the user fails to enter the log-in details.
- **Frequency of Occurrence**: Sometimes a week.

~~**OLD Use Case OUC3**: Create Recipe~~ (Now contained in UC1)

- **Scope**: Recipe Management System
- **Level**: User Goal
- **Primary Actor**: Standard User
- **Stakeholders and Interests**:
  - Standard User: Interests in creating personalized recipes.
  - Chef (Secondary Actor): Interests in sharing unique recipes.
- **Preconditions**: User is logged into the system.
- **Success Guarantee** (or **Postconditions**): Recipe is successfully saved in the user's collection.
- **Main Success Scenario** (or **Basic Flow**):
  1. User selects "Create Recipe" from the menu.

2. System prompts user to enter recipe details: title, ingredients and servings/portions, instructions, preparation and cooking times, difficulty, cuisine type, meal type, dietary information, caloric information, tags, storage/leftover, notes.
3. User fills in the recipe details and confirms.
4. System validates and saves the recipe.

- **Extensions** (or **Alternative Flow**):
  - 3a. Cancel Creation:
    - 2.b User opts to cancel the creation process.
    - 3.b System discards entered data and returns to the previous screen.
- **Special Requirements**:
  - System should support various ingredient formats and multimedia content (images, videos) for recipes.
- **Frequency of Occurrence**: Multiple times a day for active users.

Use Case UC1: New User creates a New recipe

- **Scope**: Recipe Management System
- **Level**: User Goal
- **Primary Actor**: Standard User
- **Stakeholders and Interests**:
  - Standard User: Interests in creating an account to access the software and create a new recipe.
  - Chef (Secondary Actor): Interests in creating an account to access the software and create a new recipe.
  - Dietistic (Secondary Actor): Interests in creating an account to access the software and create a new recipe.
- **Preconditions**: User has opened the software application.
- **Success Guarantee** (or **Postconditions**): System stores user and recipes's data, authenticating the user in the meantime.
- **Main Success Scenario** (or **Basic Flow**):
  1. System renders WelcomePage with Login and Sign up forms.
  2. User fills the sign up form and clicks "Sign up".
  3. System validates data, creates the user and automatically authenticates him.
  4. System renders 'Personal recipe' for the user.
  5. User clicks on "New Recipe"
  6. System renders the 'Create Recipe' view, a form asking all the required information for the recipe.
  7. User fills in all the requested information and clicks "Save"
  8. System checks data and saves the recipe.

9. System renders the ‘Read Recipe’ View of the new recipe.

- **Extensions** (or **Alternative Flow**):
  - bis. User already register:
    - 2.b User fills the log-in form
    - 3.b System authenticates the user
    - 4.b (Back to 4th point of Basic Flow)
  - tris. User registers as chef:
    - 4.c System renders Restaurant - Create
    - 5.c Users Fills in information and confirms
    - 6.c System renders ‘Personal Recipe’ view (Back to 4th point)
- **Special Requirements**:
  - System should support a “personal information page” and also an “Update” version of it for collecting all the data not collected during the registration phase

Use Case UC2: User updates his personal information

- **Scope**: Recipe Management System
- **Level**: User Goal
- **Primary Actor**: Standard User
- **Stakeholders and Interests**:
  - Standard User: Interests in updating his personal information
  - Chef (Secondary Actor): Interests in updating his personal information and/or its restaurant information.
  - Dietistic (Secondary Actor): Interests in updating his personal information.
- **Preconditions**: User is logged in and the system renders the *Personal Recipe* View.
- **Success Guarantee** (or **Postconditions**): System stores updated user information.
- **Main Success Scenario** (or **Basic Flow**):
  1. User, from the Personal Recipe page, clicks on “Personal Page”
  2. System renders the ‘Read User’ view with all the related data.
  3. User clicks on “Update information”.
  4. System Renders “Update User” view, made of forms asking for updated data.
  5. User fills in the new data and clicks “Save updates”.
  6. System saves data and redirects back to “Personal Page”
- **Extensions** (or **Alternative Flow**):
  - 5bis. Cancel Update:
    - 2.b User opts to cancel the update process.

3.b System discards entered data and returns to the previous screen.

- **Special Requirements**:
  - To edit authentication information the current password is required.

Use Case UC3: Visit Chef's Restaurant Page

- **Actor**: Standard User
- **Basic Flow**:
  1. The user is browsing through recipes and selects one of them.
  2. System renders Recipe - Read
  3. User clicks on "View Restaurant" From the CookerInfo Panel
  4. System renders Restaurant - Read
- **Preconditions**: User is logged in.
- **Post-conditions**: A new photo of 'Results' and a new review are added to the recipe.

Use Case UC4: Try others' recipe

- **Actor**: Standard User
- **Basic Flow**:
  1. The user clicks on the 'search' tab
  2. The system provides a 'search form', in which the user can search by title or by tag
  3. The user writes a search key
  4. The system provides all the matches
  5. The user, browsing through the results, selects one of them
  6. The system provides all the recipe's details
  7. The user replicates the instructions, recreating the plate.
  8. The user clicks on 'Post your result' button
  9. The system prompts an upload view
  10. The user uploads a photo of his results
  11. The system asks the user if he want to review the recipe
  12. The user agrees
  13. The system prompts a form with multiple 5-star selectors and a comment section
  14. The user fills in the data and click 'submit'
  15. The system saves the data and redirects the user to the previous window.
- **Preconditions**: User is logged in.
- **Post-conditions**: A new photo of 'Results' and a new review are added to the recipe.

- **Special requirements**: Reviewing personal recipe: A user cannot review one of his personal recipes.

Use Case UC5: Update Recipe

- **Actor**: Standard User
- **Basic Flow**:
    1. The user clicks on ‘Update recipe’.
    2. The system displays an editable page of the recipe with all its information.
    3. The User updates the information and clicks ‘Save Updates’.
    4. The system verifies data integrity and confirms the process.
    5. The system presents the view page with the updated information.
- **Preconditions**: User is logged in and is being shown the details of a single recipe.
- **Post-conditions**: Modified version of the recipe is saved as a variation.
- **Extensions** (or **Alternative Flow**):
    - 3a. Cancel Update:
        3.a User opts to cancel the update process.
        4.a System discards entered data and returns to the previous screen.

Use Case UC6: Delete Recipe

- **Actor**: Standard User
- **Basic Flow**:
    1. The user clicks on the ‘Delete’ Button.
    2. The system prompts a confirmation window.
    3. The User selects “Confirm”.
    4. The system deletes the recipe data and presents the previous screen.
- **Preconditions**: User is logged in and is being shown the details of a single recipe.
- **Post-conditions**: The recipe is deleted.

Use Case UC7: Organize Recipes

- **Actor**: Standard User
- **Basic Flow**:
    1. The user goes in “Personal CookBook” section

2. The user is presented by the system with all its personal and saved recipes.
3. The user goes in the “Personal Categories” sub-section
4. The user clicks on ‘Create Category’
5. The system prompts a Category details form.
6. The user fills the form and clicks confirm button
7. The system presents the user’s personal recipes and ask to add some in the new category
8. The user selects some of the recipes and clicks confirm.
9. The system saves the updates associating the Tag to the selected recipes
10. The system presents the new category with all the associated recipes.

- **Preconditions**: User is logged in.
- **Post-conditions**: A new category is created and the related recipes are associated with it through a tag.
- **Extensions** (or **Alternative Flow**):
    - Associate to an existing recipe a tag of an existing category:
        - 3.a User select a recipe he wants to organize
        - 4.a System presents the recipe information
        - 5.a User click “Add Tag” from the Tag section
        - 6.a System prompts a list of existing tags
        - 7.a The user select all the tags he wants to associate with the recipes and clicks confirm
        - 8.a The system saves the updates and goes back to the recipe information page.
    - Associate Tag of existing category to different recipe:
        - 4.a Use select an existing category
        - 5.a System presents all the recipes already associated with that category.
        - 6.a User clicks on ‘Associate other recipes’
        - 7.a System presents all the personal recipes not associated with that category
        - 8.a The user selects all the recipe he wants to associate with that category
        - 9.a The system saves the new association and returns to the category view.

Use Case UC8: Create a Recipe Variation

- **Actor**: Standard User

- **Basic Flow**:
    1. The user is browsing through others' recipes and selects one of them
    2. In the view page, the user clicks on 'Create your Variation'
    3. The system provides the ingredients list, asking to remove or add some other ingredients.
    4. The user fills in those information and click on "continue"
    5. The system provides the step-by-step instructions and asks to remove or add some other step.
    6. The user fills in those informations and clicks on "continue".
    7. The system saves the variation and renders his view.
- **Preconditions**: User is logged in.
- **Post-conditions**: A new recipe variation is created.

Use Case UC9: Uploading Restaurant Information

- **Actor**: Chef User Type
- **Basic Flow**:
    1. The user goes into his personal section
    2. Clicks on "New Restaurant"
    3. The system prompts a form asking for restaurant information such as name, location and kitchen style.
    4. The user fills in the information and click continue
    5. The system asks if he wants to associate some of his personal recipe to the restaurant menu
    6. The user agrees
    7. The system show all the personal recipes
    8. The user selects the ones he wants and press continue
    9. The system saves the information and presents the restaurant page.
- **Preconditions**: User is logged in.

Use Case UC10: Creating a diet for a client

- **Actor**: Nutritionist User Type
- **Basic Flow**:
    1. The user goes into his diets section
    2. Clicks on 'New Diet'
    3. The system asks for a diet name
    4. The user gives a name
    5. The system asks for the description of the meal nutrition information and for some recipes
    6. The user inputs those information.

repeat 5-6 for every meal of the week

7. The user confirms the completion of the diet
8. The system saves the data and render the diet view
9. The user click the button “Share Recipe”
10. The system asks for an email
11. The user give the info and click confirm
12. The system links the diet to the new email

- **Preconditions**: Nutritionist user is logged in.

Actors and Scenarios

Identified Actors:

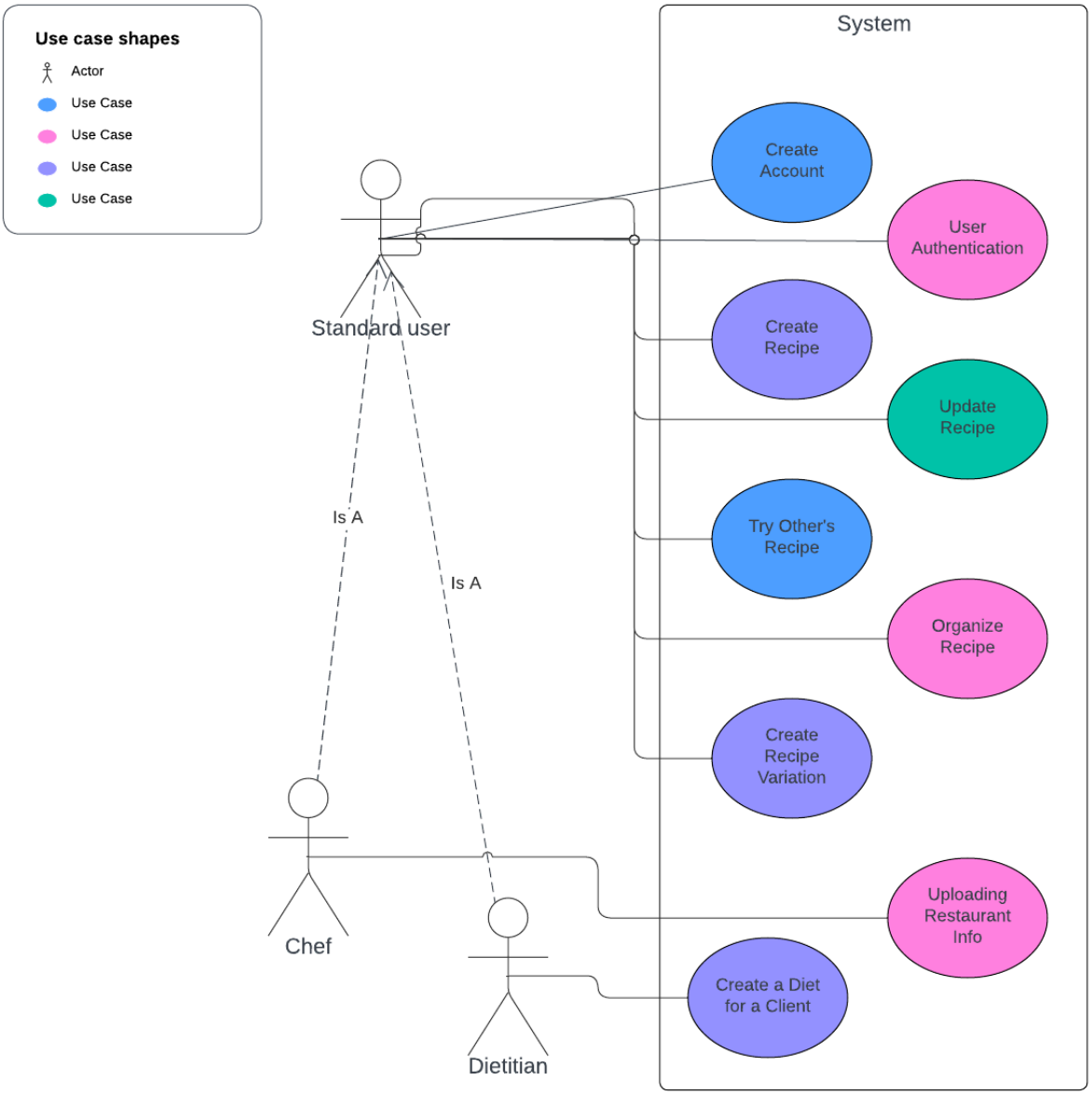
- **Standard User**: Interacts with various features like account creation, authentication, recipe management, etc.
- **Chef**: Specifically interacts with functionalities related to restaurant information and sharing recipes.
- **Dietitian**: Engages with functionalities for creating diets and sharing them with clients.

Identified Use Cases:

~~1. Create Account (OUC1)~~
~~2. User Authentication (OUC2)~~
~~3. Create Recipe (OUC3)~~

1. New User creates a New recipe (UC1)
2. User updates his personal information (UC2)
3. Visit Chef’s Restaurant Page (UC3)
4. Try Others’ Recipe (UC4)
5. Update Recipe (UC5)
6. Delete Recipe (UC6)
7. Organize Recipes (UC7)
8. Create a Recipe Variation (UC8)
9. Uploading Restaurant Information (UC9)
10. Creating a Diet for a Client (UC10)

Use Case Diagram



4. Domain Model:

Core Concepts:

Here are the core concepts, as entities and relationships, of the application domain obtained by the Requirements Analysis.

Entities:

- **System Entity:**
  - **Attributes**: UserConnected.
  - **Methods**: RegisterUser, UpdateProfile, AuthenticateUser, Get/Set Attributes, [...].
- **User**:
  - **Attributes**: UserID, Username, Email, Password, UserType, RecipeList, ReviewList, Restaurant, DietSharedList, [...].
  - **Methods**: CreateRecipe, UpdateRecipe, DeleteRecipe, CreateTag, UpdateTag, DeleteTag, GetAllRecipes, GetAllTags, ReadReceivedDietPlan, Get/Set Attributes, [...].
  - ➢ **Chef** (Specialization):
    - **Attributes**: RestaurantID
    - **Methods**: CreateRestaurant, UpdateRestaurant, DeleteRestaurant, GetRestaurantMenu, Get/Set Attributes, [...].
  - ➢ **Dietitian** (Specialization):
    - **Attributes**: CreatedDietList
    - **Methods**: CreateDiet, ShareDiet, GetDietPlan, Get/Set Attributes, [...].
- **Recipe**:
  - **Attributes**: RecipeID, UserID, Title, OriginalRecipeID, OriginalUserID Description, Ingredients, Instruction, PreparationTime, CookingTime, TotalTime, DifficultyLevel, CuisineType, MealType, CaloricInfo, Ratings, ReviewListCa [...].
  - **Methods**: GetRecipeDetails, ReviewRecipe, AddTag, FilterByAttribute (CuisineType, MealType, TotalTime,..), GetReviewList, Get/Set Attributes, [...].
- **Category/Tag**:
  - **Attributes**: TagID, UserID, CategoryName, Tag, RecipeList, [...].
  - **Methods**: AssociateRecipes, Get/Set Attributes, [...].
- **Review**:
  - **Attributes**: ReviewID, RecipeID, UserID, OverallRating, TasteRating, PreparationRating, ReproducibilityRating, StalenessRating, Comment, DatePosted, [...].

- **Methods**: CreateReview, Get/Set Attributes, [...].

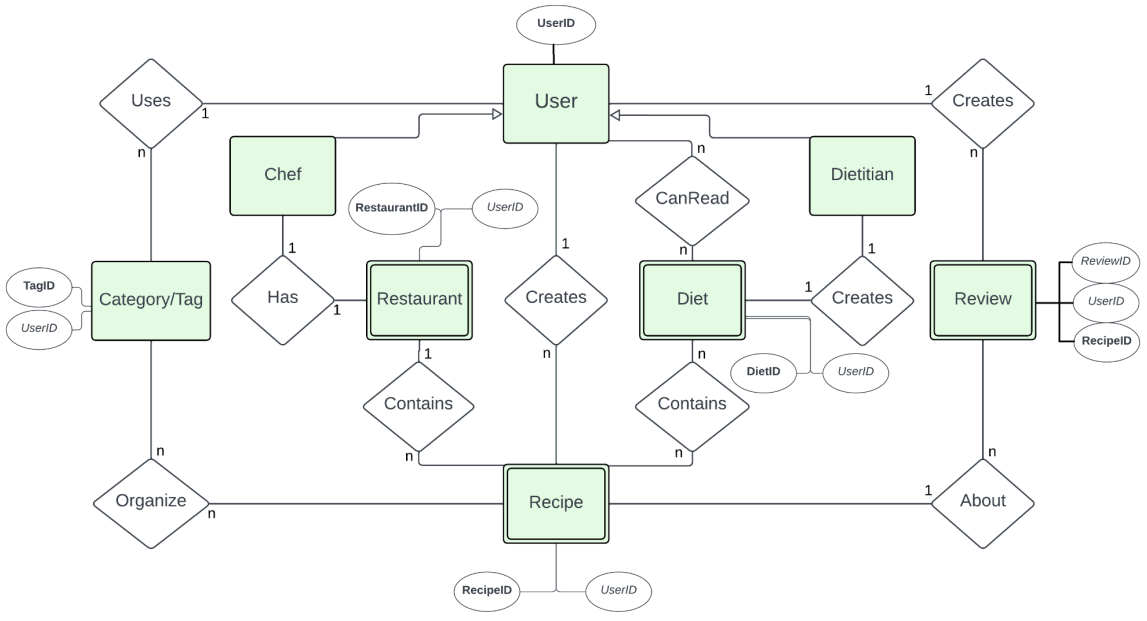
- **Restaurant**:
  - **Attributes**: RestaurantID, Nome, Indirizzo, Orari, Menu [...].
  - **Methods**: CreateMenu, UpdateMenu, UpdateInformation, Get/Set Attributes, [...].

- **Diet**:
  - **Attributes**: DietID, NutritionistID, DietName, SharedWith, NutritionInfo, MealPlan, [...].
  - **Methods**: CreateDietPlan, GetMeal, GetClients, Get/Set Attributes, [...].

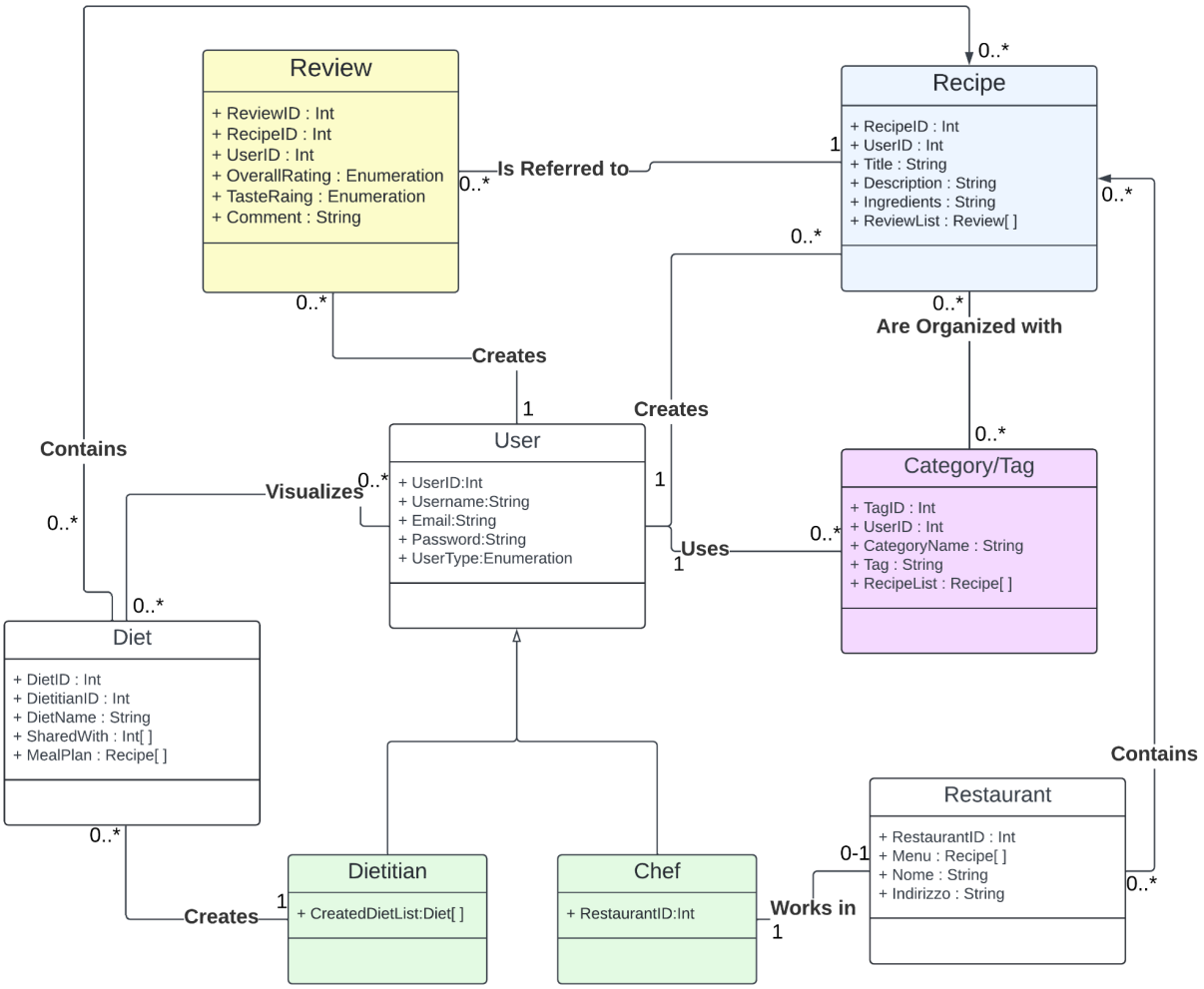
Relationships:

- **User - Recipe**: One-to-Many (A user can have multiple recipes, each belongs to one user).
- **User - Review**: One-to-Many (A user can have multiple review, each belongs to one user).
- **User (Chef) - Restaurant**: One-to-One (Each chef user manages one restaurant, a restaurant is managed by one Chef).
- **User - Diet Plan**: Many-to-Many (Each User can see multiple Diet Plans, each diet plan can be seen by multiple users).
- **User (Dietitian) - Diet Plan**: One-to-Many (Each Dietitian can create multiple Diet Plans, each Diet plan can be created by a single Dietitian.
- **Recipe - Category/Tag**: Many-to-Many (Each recipe can be associated with multiple tags, each tag can be associated with multiple recipes).
- **User - Category/Tag**: One-to-Many (Each User can have multiple tags, each tag can be associated with only one user).
- **Recipe - Restaurant**: Many-to-Many (Each recipe can be associated with multiple restaurants' menu, each restaurant's menu can have multiple recipes).

Entity-Relationship Diagram



Object Model Diagram



Comments

In both the ER Diagram and the Object Model Diagram the system entity is not represented. This choice was done considering the fact that its only functionality is the user creation, while it is more relevant to emphasize the other entities and their relationships.

5. Design

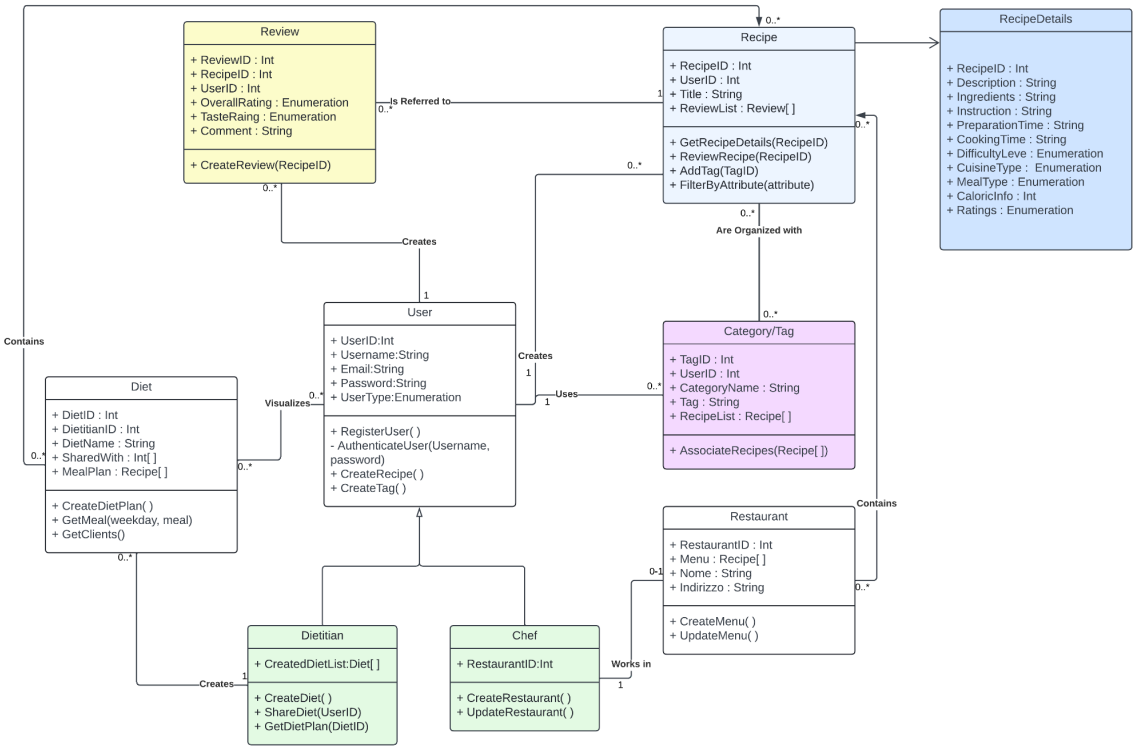
The possible system architectures were Layered Architecture and MVC. In the decision process for the system architecture the following considerations were taken:

- The *Layered Architecture* emphasizes the organization of the system based on functional layers, but our system provides a lot of simple functionality, all around the 4 main entities (User, Recipe, Review, Tag), so this choice would lead to a nonfunctional fragmentation.
- The application is primarily focused on showing data to the user, so there is no need for particular security or distance between the Presentation Layer and the Data Access Layer, which are delivered by the *Layered Architecture*.
- The *MVC architecture* has the same separation as the application conceptual structure: a part that interacts with the user, a part that contains the information and the last that connects these two.

Considering this, the MVC seems the best choice to obtain a Modular and Reusable application, ensuring a good separation of concerns.

Models Layer:

Starting from the Object Model Diagram of the Domain Model, we can retrieve with ease a Class Model, provided using UML. Given the fact that the Recipe Entity contains a lot of details, it has been taken the choice to create a new Weak Entity called RecipeClients. This is in line with the current idea of creating a specific View for the Recipe with all its details, and other views in which you have to show multiple recipes without all its information reducing noticeably the storage information exchanged during the queries. Furthermore, this choice should also improve clarity and readability of the code.



From this, we can identify the following Models to implement:

- **User Model**
- **Review Model**
- **Diet Model**
- **Restaurant Model**
- **Category Model**
- **Recipe Model**
- **RecipeDetail Model**
- **Recipefy Model:** System Entity, as said before note visualized but necessary.

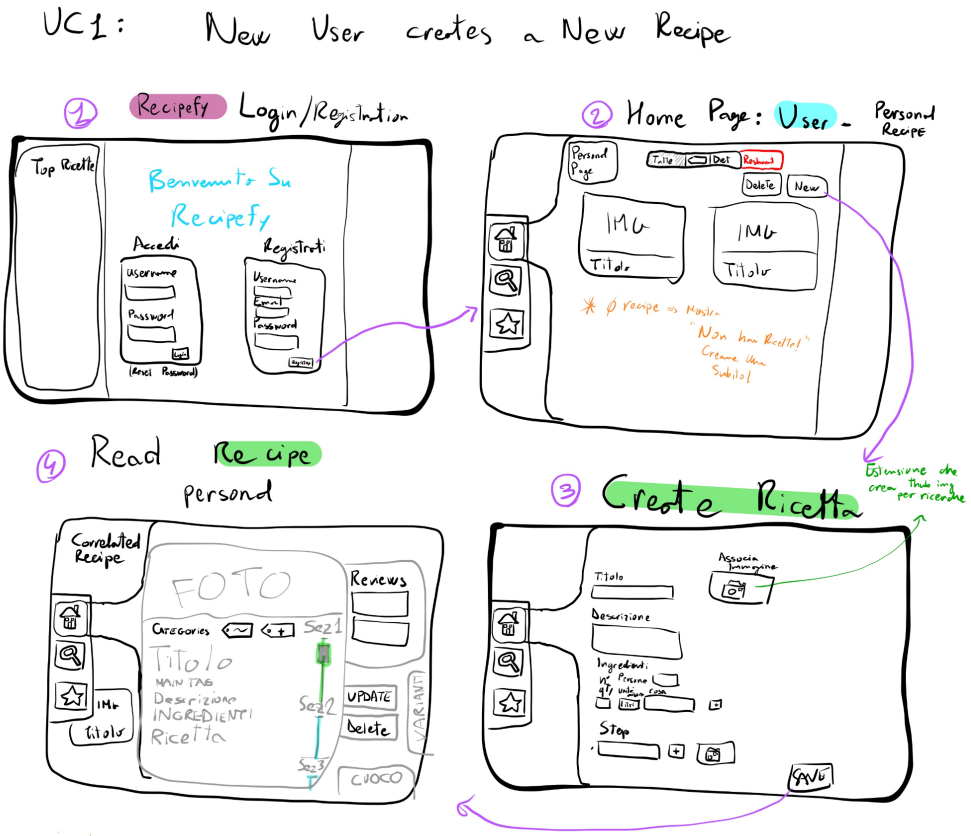
Views Layer:

To determine the presentation Layer of this application, the process has been:

- Creating Mockups of the views drawing them with an Ipad (not particularly different from using pen and paper), based on most important Use Cases.
- Associate each frame to a specific entity domain.
- Collect the functionality provided by that frame (this is helpful to the next phase: determining the functional domain of controllers).

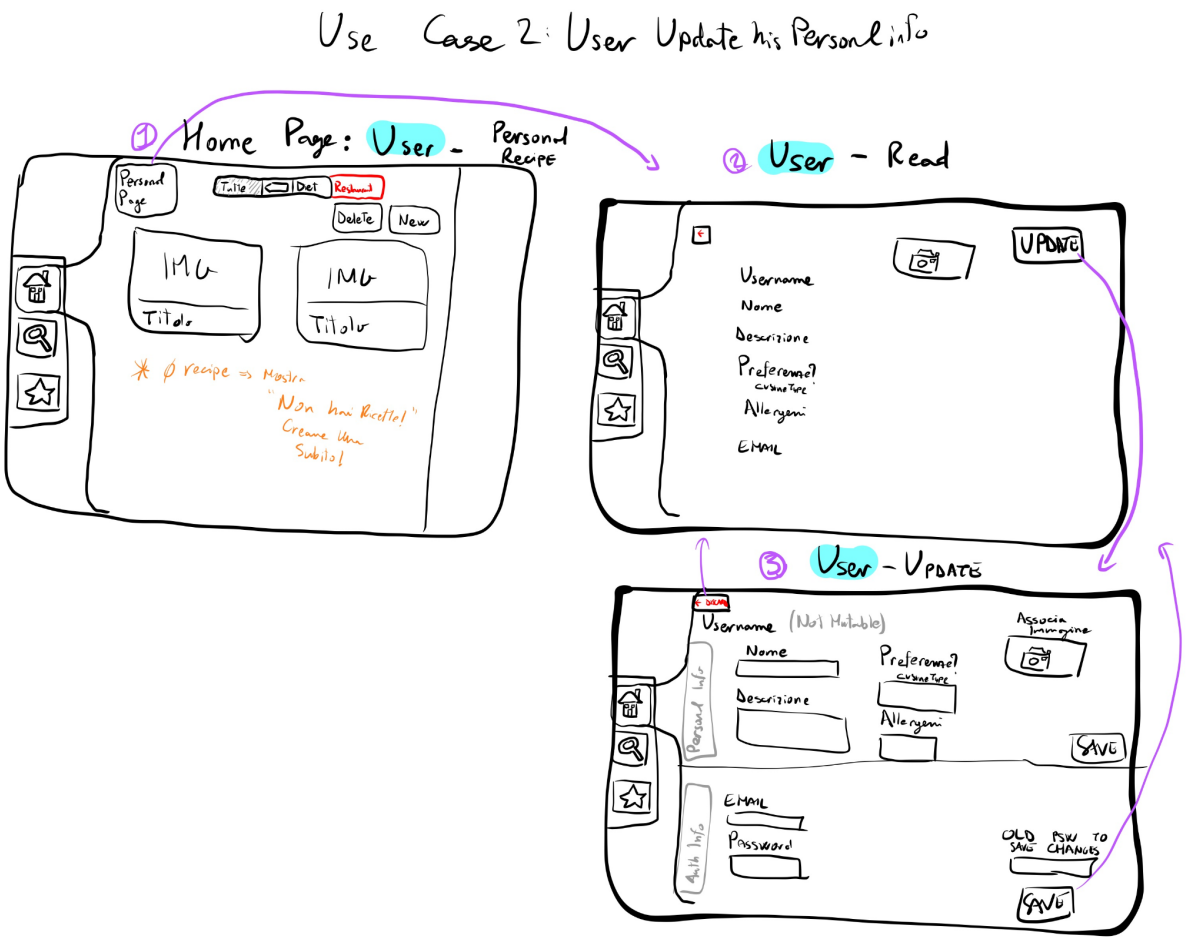
Use Case UC1: New user creates a New Recipe

This Use Case has been developed during this stage, grouping the olds *OUC1, OUC2, OUC3.* During the development we also noticed the need of the **UC2: User Updates his personal information**. Furthermore, we realized that associated with the Chef Type User

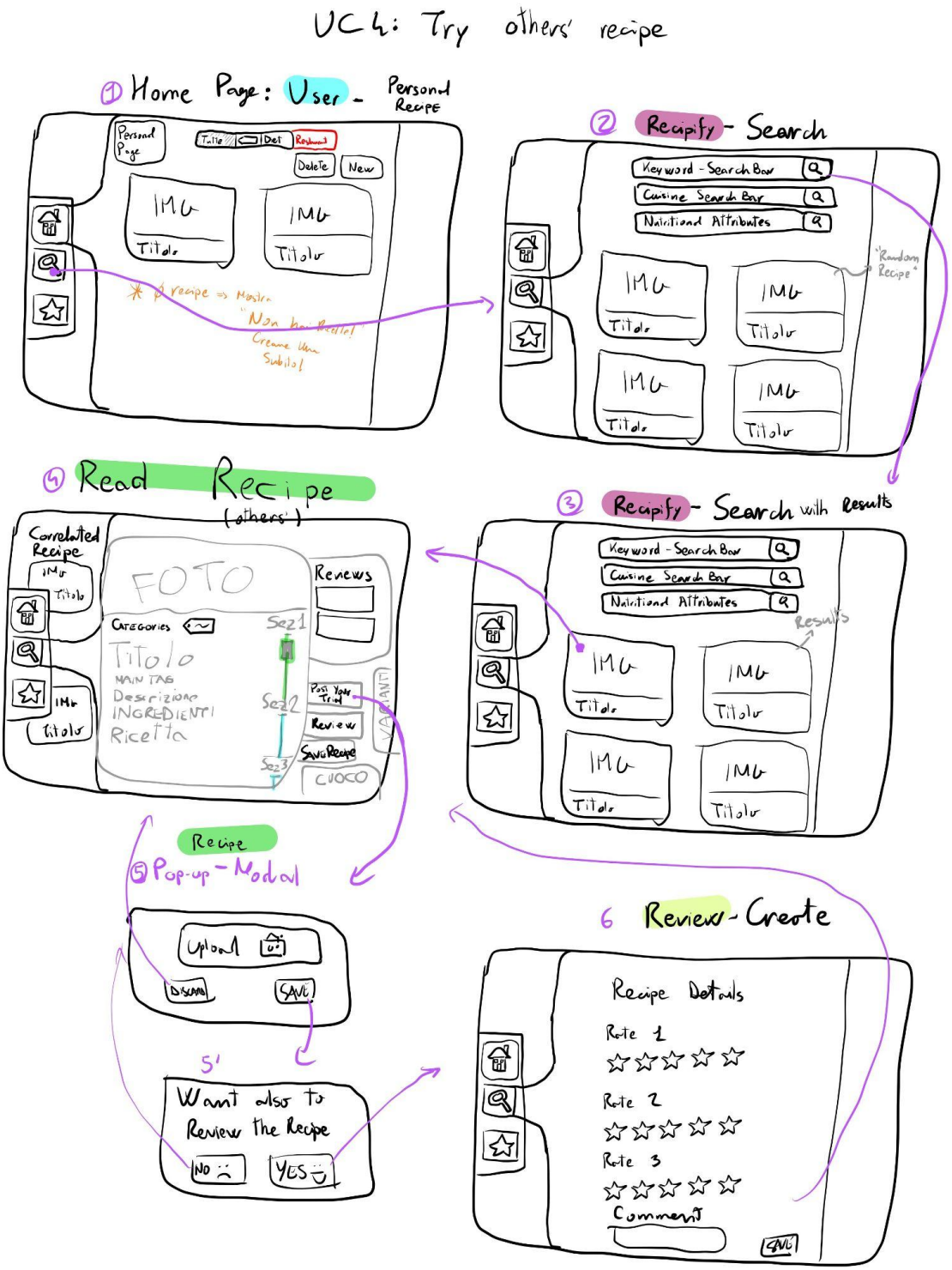


creation, we have to provide the Restaurant creation. So a new Extension has been added to this use case (4tris) and the View “Restaurant - Create” has been identified.

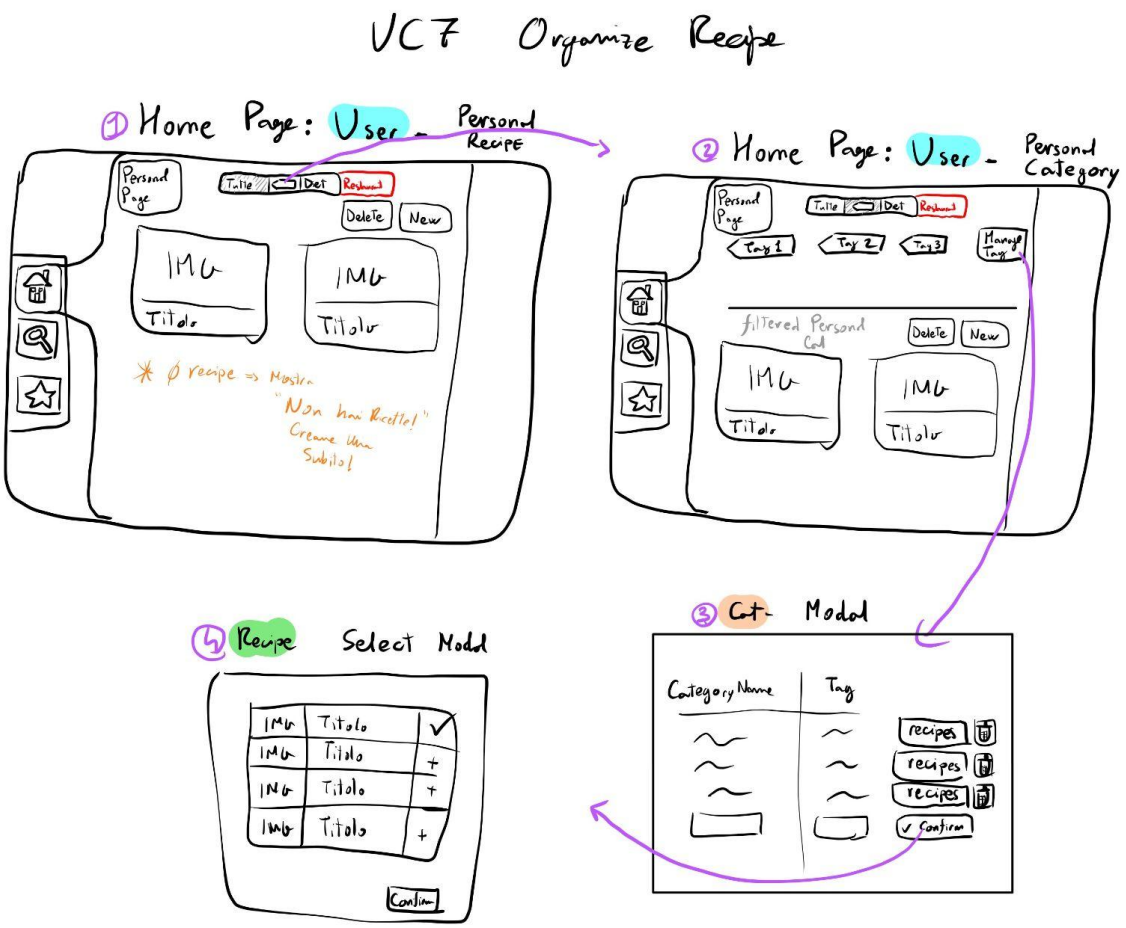
Use Case UC2: User Updates his personal information

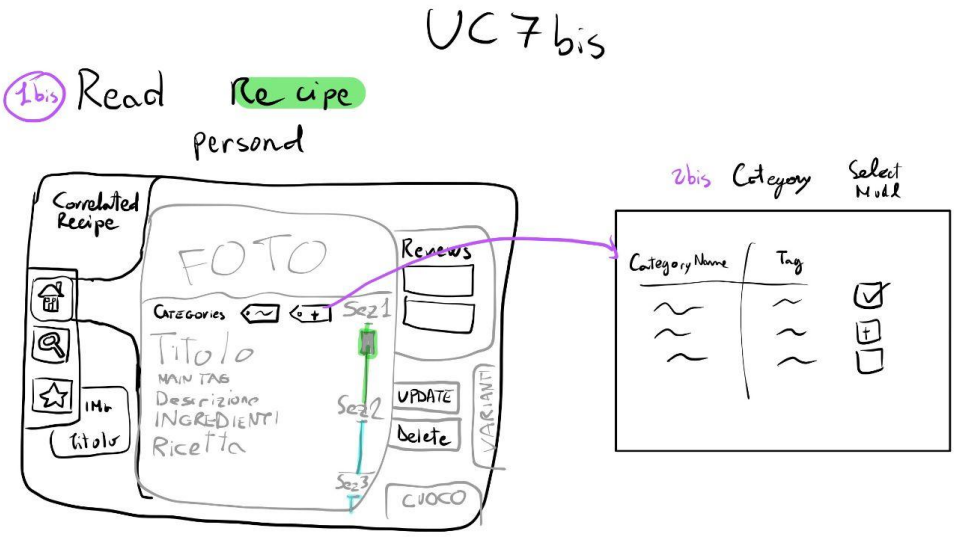


Use Case UC4: Try others' recipe

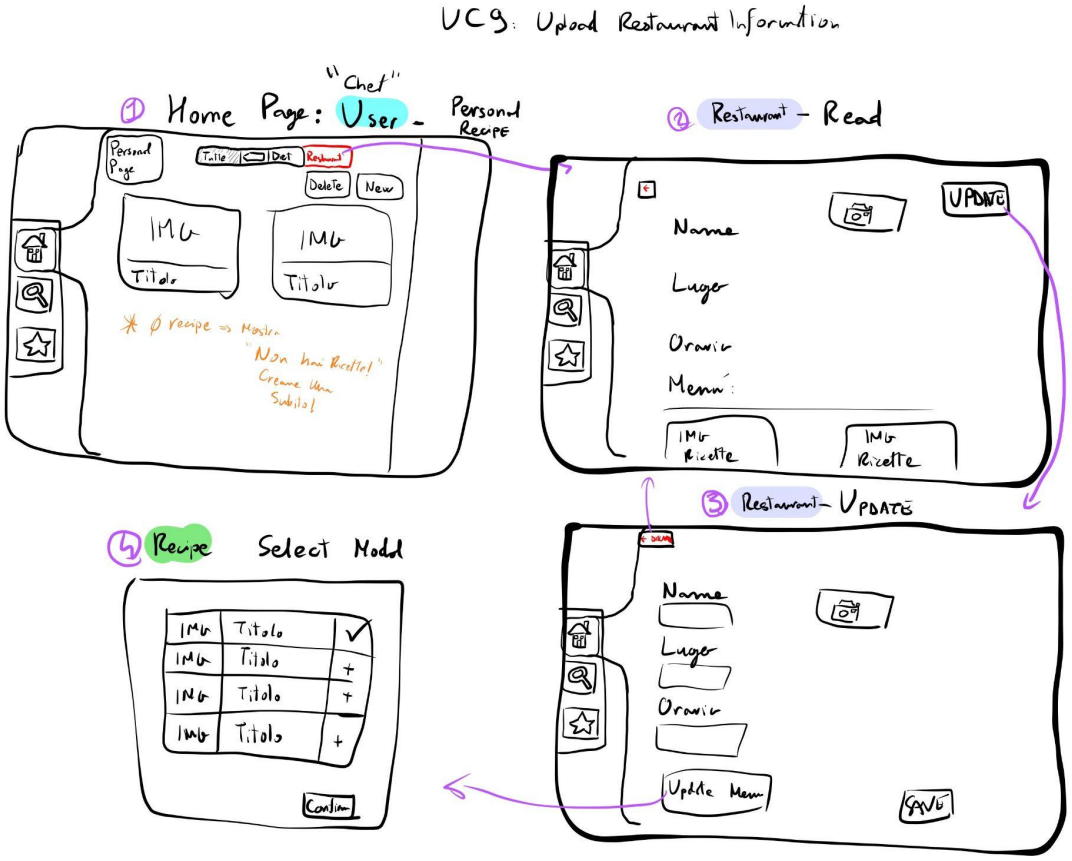


Use Case UC7: Organize Recipes

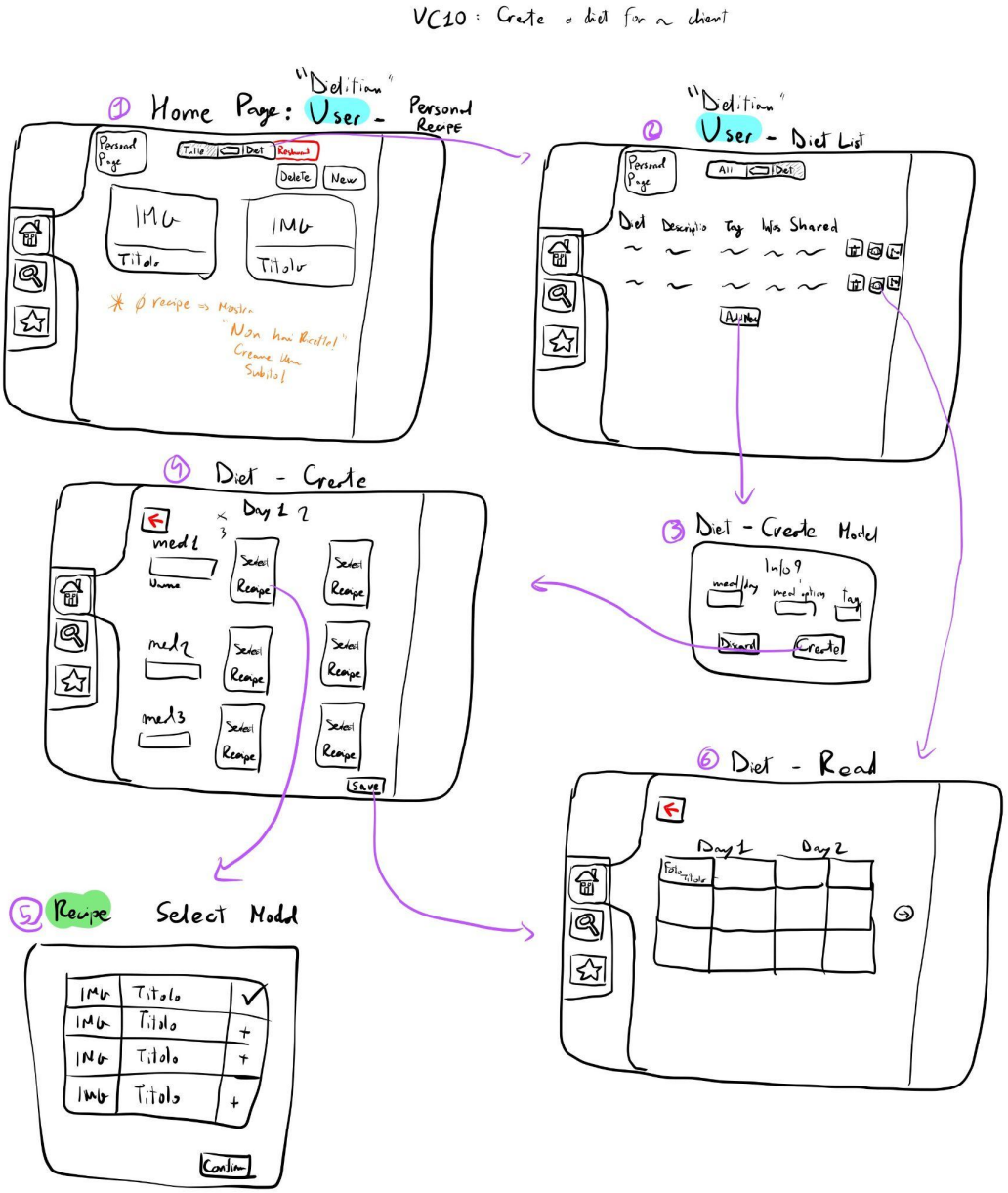




Use Case UC9: Uploading Restaurant Information



Use Case UC10: Creating a diet for a client



NOTE
Considering all the previous considerations, the following uc and requirements were added at this stage:
UC1: New user creates a New Recipe
UC2: User updates his personal information
UC3: Visit Chef's Restaurant Page

In this early iterations those functionalities haven't been developed yet:

- The "create your variation" is being postponed due to a need for further storage management and relationships analysis.

- There is a “Save Recipe” button but still isn’t provided a storage logic to it and how to read all those recipes.
- Review functionality to let the user view, update or delete a review are not provided yet.

The following Use Cases were not implemented as mockups for the following reasons:

- UC3: it wouldn’t provide new views.
- UC5: it would provide only one new view and without any new functionality.
- UC6: it’s implemented with a button, providing no new views.
- UC8: further analysis about storage and logic needed.

View Identified

Analyzing the mockups created, we can identify those View, grouped by entity:

- **Recipefy:**
  - Login/Registration (UC1)
- **User:**
  - HomePage/Personal CookBook (UC1,UC2,UC4, UC7,UC9,UC10)
  - Read (UC2)
  - Update (UC2)
  - PersonalCategory (UC7)
  - DietList (UC10)
- **Recipe:**
  - Create (UC1)
  - Read (UC1,UC4,UC7,UC7bis) [Personal-Others']
  - _UploadTrial - Modal (UC4)
  - _Ask for Review - Modal (UC4)
  - _Select - Modal (UC4,UC9,UC10)
  - Search (UC4)
  - Update (UC5)
- **Review:**
  - Create (UC4)
  - ____MISSING____ Update
  - ____MISSING____ Read
  - ____MISSING____ Delete
- **Category:**
  - _Manage - Modal (UC7)
  - _Select - Modal (UC7bis)
- **Restaurant:**
  - Read (UC9)
  - Update (UC9)
  - Create (UC1)
- **Diet:**
  - _Create - Modal (UC10)
  - Create (UC10)
  - Read (UC10)
  - ____MISSING____ Update
- **Components**:
  - NavBar
  - RecipeShower
  - FormPanel
  - ShowAttributePanel
  - UserNavBar

For the UI a '**Components**' package is provided, to maximize the reusability of Interface Components.

Controller Layer:

The controllers as well are grouped by entities. To collect all the functionality that a controller must provide, all the views belonging to the same entity are analyzed.

- RecipefyController:
  - [Login] UserAuth(username, password) : **Creator Pattern**
  - [Login] UserCreate(username, email, password) : **Creator Pattern**
  - [Login] ResetPassword(email) : **Creator Pattern**
- User:
  - [HomePage] PersonalInfoRedirecter()
  - [HomePage] ShowRecipes(recipes[]) :**Controller and high Coesion**
  - [HomePage] CreateRecipe() : **Creator Pattern**
  - [HomePage] MultiNavBarRedirector()
  - [Read] ShowUserInfo()
  - [Read] UpdateButton()
  - [Read] BackButton()
  - [Update] CollectData()
  - [Update] PasswordCheck()
  - [Update] SaveButton()
  - [Update] DiscardButton()
  - [CatPage] PersonalInfoRedirector()
  - [CatPage] ShowCategory()
  - [CatPage] ManageCategoryButton()
  - [CatPage] MultiNavBarRedirector()
  - [CatPage] ShowRecipes(recipe[])
  - [DietList] PersonalInfoRedirector()
  - [DietList] MultiNavBarRedirector()
  - [DietList] ShowDietList(diet[])
  - [DietList] NewDiet()
- Recipe:
  - [Create] CollectData()
  - [Create] SaveButton()
  - [Create] DiscardButton()
  - [Read] ShowRecipeDetails(recipe)
  - [Read] ShowReviewList(recipe)
  - [Read] ShowCookerInfo(userID)
  - [Read-Personal] Cats elector()
  - [Read-personal] UpdateButton(recipe)
  - [Read-personal] DeleteButton(recipe)
  - [Read-Others’] SaveButton(recipe, userID)
  - [Read-Others’] ReviewButton(recipe, userID)
  - [Read-Others’] TrialButton(recipe, userID)
  - [_UploadTrial] UploadPhoto()
  - [_UploadTrial] ConfirmButton()
  - [_UploadTrial] DiscardButton()

    - [_Ask for Review] ReviewRedirect()
    - [_Ask for Review] DiscardRedirect()
    - [_Select] ShowRecipesSelectable()
    - [_Select] ConfirmButton()
    - [Search] Searchbar(keys) : ***Strategy Pattern***
    - [Search] ShowRecipes(recipe[]) : ***Composite Pattern***
- Review:
    - [Create] DataCollect()
    - [Create] ConfirmButton()
- Category:
    - [_Manage] ShowCategory(userid)
    - [_Manage] AddRecipes()
    - [_Manage] DeleteCategory(catId)
    - [_Select] ShowCategorySelectable(Userid)
- Restaurant:
    - [Read] ShowRestaurantInfo()
    - [Read] ShowRecipe(menuid)
    - [Read] UpdateButton()
    - [Update] DataCollectUpdate()
    - [Update] SaveButton()
    - [Update] DiscardButton()
- Diet:
    - [_Create] InfoCollect()
    - [_Create] ConfirmButton()
    - [_Create] DiscardButton()
    - [Create] MealInfoCollecter()
    - [Create] SelectSingleRecipe()
    - [Create] SaveButton()
    - [Create] DiscardButton()
    - [Read] BackButton()
    - [Read] DayShower()
    - [Read] DaySwitcher()

Comments about Responsibility Assignment

General Comments:

In this phase I noticed that I wasn't providing a way to let other Users visit the Chef's Restaurant Page. **Use Case 3** has been developed but not implemented as mockups because it wouldn't provide new views. The only new thing is the link between the Recipe Page (*Recepe - Read*) and the Restaurant Page (*Restaurant - Page*). In particular, the CookerInformation Panel as a redirector to the restaurant page.

- Spiegare l'idea di SaveRecipe
- Share recipe still not implemented

Architecture hypothesis

After a further analysis, the final architecture developed in this phase is the following:

- Views:
  - **Recipefy:**
    - ➢ Login/Registration
  - **User:**
    - ➢ HomePage/Personal CookBook
    - ➢ Read
    - ➢ Update
    - ➢ PersonalCategory
    - ➢ DietList
  - **Recipe:**
    - ➢ Create
    - ➢ Read [Personal-Others']
    - ➢ _UploadTrial - Modal
    - ➢ _Ask for Review - Modal
    - ➢ _Select - Modal
    - ➢ Search
    - ➢ Update
  - **Review:**
    - ➢ Create
  - **Category:**
    - ➢ _Manage - Modal
    - ➢ _Select - Modal
  - **Restaurant:**
    - ➢ Read
    - ➢ Update
    - ➢ Create
  - **Diet:**
    - ➢ _Create - Modal
    - ➢ Create
    - ➢ Read
- Controllers:
  - RecipefyController
  - UserController
  - RecipeController
  - ReviewController
  - CategoryController
  - RestaurantController
  - DietController
- Models:
  - User Model
  - Review Model
  - Diet Model
  - Restaurant Model
  - Category Model
  - Recipe Model
  - RecipeDetail Model
  - Recipefy Model:

- **UI Components**:
    - NavBar
    - Buttons
    - RecipeShower
    - Modal Dialog
    - Form Panel
    - Search Bar
    - NavigationHeader
- **Utility**:
    - Data Collector

— — — — —

Idee:

- Specify the functionality to Save others' recipe
- ~~Separate Le informazioni riguardanti la Ricetta e la Ricetta stessa (In accordo con la separazione tra la view in cui puoi vedere tante ricette, e la view in cui vedi in dettaglio la ricetta)~~
- Il filtro per gli attributi può essere un metodo di sistema, dato che è ciò che contiene la cosa.
    - Verificare con i pattern GRASP.
    - Fare il Check per altri metodi che restituiscono un array di elementi, soprattutto se è un arrey di oggetti appartenenti alla classe di quel metodo. (Recipe -> RecipeID[ ] FilterByAttribute)
- Implementare "PostYourResults" in iterazione successiva
- Creare DietPlan come suddivisione gerarchica in DietDay (Composizione di DietMeal) e DietMeal (Composizione di ricette). Salvarle in database come vettore, matrice e vettore di matrici (corrispondenza contraria). Utilizzare un parser per convertire dato in oggetto.
- Creare Categorie generali (non associate ad utenti) tipo CuisineType e Nutritional Attributes, e Categorie personali. Una tabella, con una colonna CategoryTipe
- Aggiungere "View Others' Trial" (Notato durante Studio UC4)
- Aggiungere "Review - Read/Update/Delete" (Notato durante Studio UC4)
- Devi poter vedere il ristorante degli chef (Notato durante Studio di UC9)
- Handle exception throws back-propagation for Login and Register
- Implement a Database Table for Constant through the system
- Create Chefs and Dietitians through registration.
- Form validation
- Implement difficulty level as enum
- login/registration set attributes as private and create get methods.

— — — — —