POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

AUTOMATION AND CONTROL ENGINEERING
SOFTWARE ENGINEERING PROJECT

DIFFEREAT

Always unique meal

Serra Nur AKMESE

Arda BOZKURT

Steve FOFOU LONTCHI

Academic Year: 2022-23

Contents

Abstract

The "Differeat" project aims to create an innovative software application that addresses the common challenges students face when it comes to home cooking. This project centers on developing a user-friendly platform that enables users to effortlessly input the ingredients they have at their disposal. In return, the application will provide personalized recipe recommendations, empowering users to prepare delicious meals with the ingredients they have on hand.

One of the core features of "Differeat" is its ability to curate and present the top 5 matching dishes based on the user's input. This not only streamlines the cooking process but also serves as a source of culinary inspiration. Furthermore, the application will accommodate users with diverse culinary preferences by allowing them to select a specific cuisine type, such as Turkish, Chinese, or Italian, for their recipe recommendations. This versatility ensures that "Differeat" caters to a wide range of tastes and preferences, making it an ideal companion for students seeking diverse and delightful meal options.

Overall, the "Differeat" project embodies a commitment to enhancing the cooking experience for students. By simplifying the process of finding and preparing meals using available ingredients, this software application aims to promote culinary exploration, healthy eating habits, and a balanced lifestyle among its users. Through a user-centric design and a focus on practicality, "Differeat" aspires to be the go-to solution for students looking to savor homemade, budget-friendly, and delectable dishes in their everyday lives.

1. Feasibility study

1.1. Project description and competitor analysis

The problem

Students often face multiple challenges when it comes to cooking at home. One of the main issues is the lack of knowledge and experience in preparing meals, as many students have never learned basic cooking techniques or recipes.
Additionally, many students have busy schedules, which limits the time available for cooking and meal preparation. Furthermore, students may not have access to a wide range of ingredients due to limited budgets or living arrangements, which can make meal planning and preparation difficult.
Despite these challenges, students still desire to enjoy delicious, healthy, and homemade food as part of a balanced lifestyle.

The solution

Our solution is to address the problems faced by students who have difficulty cooking with a software application called "Differeat".
The application will provide a simple and user-friendly interface that allows the user to input the ingredients they have available, select the desired cuisine, and receive a list of the top 5 recipes that can be made with those ingredients. The recipes will be curated from reliable sources and will be easy to follow, even for novice cooks.
To address the time constraint issue, the application will prioritize recipes that can be made quickly, such as those that can be prepared within 30 minutes or less. In addition, the app will suggest recipes that require minimal preparation and clean-up time.
To overcome the problem of limited ingredient availability, the application will be designed to suggest recipes that require only a few basic ingredients, along with a few that the user may not have but are easily obtainable. The recipes will also be flexible and will allow the user to substitute ingredients if necessary.
To further enhance the user experience, the application may also allow users to save their favourite recipes, create shopping lists, and share recipes with friends and family through social media.

The competitor's analysis

In order to assess the market for the proposed "Differeat" application, a competitor analysis was conducted. The analysis revealed several competitors and alternative solutions that may pose a challenge to the success of the project:

- **ChatGPT and other generative AI:** These AI-based language models have the capability to generate recipes based on ingredients, but they may lack the human touch and practicality in terms of real-world cooking experience. They are primarily text-based and may not provide a user-friendly interface or additional features.
- **Classic cooking websites:** Traditional cooking websites such as Food Network, Allrecipes, and BBC Good Food offer a vast collection of recipes, but users may need to search and filter through a large database to find recipes that match their available ingredients.
- **Social media influencers:** Influencers on platforms like YouTube and Instagram often share their own recipes, but these may not be tailored to the specific needs and limitations of students, such as budget-friendly options or limited ingredient availability.

- **Other recipe apps like SuperCook, Dinner Spinner, Epicurious, and Cookpad:** These apps allow users to search for recipes based on available ingredients, but they may not cater specifically to students and their unique challenges in terms of time constraints, limited ingredients, and cooking skills.

Differeat could stand out by being created by students for students. It could offer budget-friendly recipes, easy-to-make options, and ingredient substitution flexibility, all tailored to students' specific needs and limitations. The user interface could be simple and intuitive, with the potential to add meal planning, grocery list creation, and social sharing options later on.

1.2. Project scope

The prototype should be able to:

- **User Input:** Allow users to input the ingredients they have available for cooking through a simple user interface that allows users to select from a predefined list of ingredients in a database
- **Recipe Matching:** Implement a matching algorithm that takes the user-inputted ingredients and matches them with recipes in a database.
- **Recipe Display:** Display the top 5 matching recipes to the user, along with relevant details such as recipe name, ingredients, and cooking instructions.
- **Cuisine Selection:** Allow users to choose a specific cuisine (e.g., Turkish, Chinese, Italian) for the recipe recommendations.
- **Basic User Management:** Implement basic user management functionality, such as user registration and login

The overall app could further include the following features:

- **Favourite Recipes:** Allow users to save their favourite recipes to their account for easy access and future reference.
- **Search History:** Provide users with the ability to view their search history, allowing them to revisit previous recipes and ingredient combinations.
- **Personalization:** Allow users to set their dietary preferences and restrictions, such as vegetarian, vegan, gluten-free, etc. The system can use this information to generate personalized recipe recommendations.
- **Notifications:** Implement a notification system to alert users about new recipes, personalized recommendations, or updates related to their favourite recipes.
- **Nutritional Information:** Display nutritional information for each recipe, including calories, macronutrients, and other relevant details.
- **Shopping List:** Generate and manage a shopping list based on the selected recipes, allowing users to easily gather the necessary ingredients for their cooking.
- **Quantity Management:** Provide a feature to add the specified quantity of each ingredient to the system. This will enable users to track their ingredient stock and receive reminders when certain ingredients are running low.
- **Admin Management Section:** Provide an admin management section for administrative users to manage user accounts, ingredient and recipe database (add, edit, delete), and perform other administrative tasks.

1.3. Technical feasibility and risks and challenges

- **Development Framework and Language**: Leveraging Flutter as our development framework and Dart as the programming language, both of which are beginner-friendly, enhances the project's feasibility.

- **Standardized Database**: To build the database of ingredients and recipes, manual data entry will be required.
- **Technical Resources**: Our project does not demand specialized hardware or software. Standard development tools and personal computers suffice.
- **Engineering Team Capabilities**: As students embarking on this school project, our engineering team has the requisite knowledge to work with Flutter and Dart effectively.

Risks and Challenges

- **Database Creation**: Manually creating the ingredients and recipes database can be time-consuming. However, it presents a valuable learning experience.
- **Cross-Platform Compatibility**: Ensuring seamless functionality on both Android and iOS platforms might pose challenges, but Flutter's cross-platform capabilities mitigate this risk.
- **Scalability**: Scaling the application as it gains popularity may require additional resources, but our iterative development approach allows us to address scalability progressively.

1.4. Schedule

A possible schedule for the development of the Differeat application, considering the project scope and the technical feasibility, could be as follows:

Month 1:

Week 1-2: Define project scope, gather requirements, create a high-level design document.

Week 3-4: Low level design.

Month 2:

Week 1-2: Coding&Unit test

Week 3-4: Coding&Unit test

Month 3:

Week 1-2: Perform system integration, testing and debugging of the application.

Week 3-4: Prepare for project presentation and submission.

2. Requirements Analysis and Specification

2.1. Scenarios

Scenario 1: Maria plans to prepare dinner once she returns home after a day of classes. Upon reaching home, she decides to download and install the Differeat application. After creating an account, Maria logs into the app. To find a suitable recipe, she enters the ingredients available in her kitchen and may optionally specify a preferred type of cuisine. The application considers the entered ingredients and cuisine preference and presents her with five recipe suggestions. After carefully considering the options, she selects one and follows the provided cooking instructions to prepare her dinner. Finally, she savours the meal and thoroughly enjoys it.

Scenario 2: *Jason wants to plan his meals for the week and ensure he has all the necessary ingredients. He opens the Differeat application and logs into his account. He accesses the "Shopping List" feature and reviews the ingredients needed for the recipes he has selected. He adds any missing ingredients to his shopping list and specifies the quantities needed. The application generates the complete shopping list based on the selected recipes and quantities. Jason goes to the grocery store, checks off the items from his shopping list as he purchases them, and ensures he has everything he needs for the upcoming meals.*

Scenario 3: *Paolo, a cooking enthusiast, uses the Differeat application to enhance his culinary experience. He saves favourite recipes and manages them in the "Favourites" section. Paolo receives notifications for new recipes and personalized recommendations, which he can manage in his notification preferences. He can also access detailed nutritional information for each recipe, including calories, macronutrients, vitamins, and minerals. These features enhance Paolo's cooking experience and encourage him to explore more recipes.*

Scenario 4: *Greta, a student with specific dietary restrictions, wants to find suitable recipes. She logs into the Differeat application and accesses her account settings. In the "Dietary Preferences" section, she indicates her dietary restrictions, such as being gluten-free and lactose intolerant. The application saves her preferences. Greta then proceeds to search for recipes by entering the available ingredients she has at home. The application matches the ingredients with recipes that comply with her dietary restrictions and presents the top 5 suitable recipes. Greta selects a recipe and follows the provided cooking instructions, ensuring it aligns with her dietary needs. She successfully prepares a delicious meal that meets her dietary requirements and enjoys the satisfaction of finding a recipe tailored to her needs.*

*the scenarios 2,3,4 are not developed in the prototype.

2.2. Use cases

From the scenario 1, we retrieve these use cases:

- Sign Up User
- Login
- Input Ingredients
- Select a cuisine
- Choose a Recipe

UC1	**Sign Up User**
Actors	User
Entry Condition	The user has installed the application on his/her device
Flow of Events	1. Click on “Sign up” button 2. Fill all the necessary information 3. Click on “Confirm” button 4. The system saves the data
Exit Condition	The user has successfully registered and can use the application
Exceptions	1. The user is already signed up 2. The user didn’t fill all the mandatory fields with valid data 3. The username is already taken

	4. The e-mail is already registered 5. All the exceptions are handled by notifying the user and taking him back to the sign-up activity.

UC2	**Login**
Actors	User
Entry Condition	The user is previously successfully signed up
Flow of Events	1. The user opens the application on his device 2. He enters his credentials in the “Username” and “Password” fields of the home page 3. The user clicks on the “Log in” button 4. The user is successfully logged in his session
Exit Condition	The user is successfully redirected to his/her session
Exceptions	1. The user enters invalid Username 2. The user enters invalid Password 3. All the exceptions are handled by notifying the user and taking him back to the login activity

UC3	**Input Ingredients**
Actors	User
Entry Condition	User has successfully logged in
Flow of Events	1. User opens the application. 2. User selects the "Input Ingredients" option. 3. Application displays a list of ingredients from the database. 4. User selects the ingredients they have available. 5. User submits the selected ingredients to the application.
Exit Condition	The application receives the user's inputted ingredients and proceeds to find matching recipes
Exceptions	If the user submits none, invalid or non-existent ingredients, the application displays an error message and prompts the user to input valid ingredients

UC4	**Select a cuisine**
Actors	User
Entry Condition	The user has already inputted ingredients
Flow of Events	1. User select none, one or more type of cuisine 2. User submits his selection to the application
Exit Condition	The application receives the user's inputted cuisine and filters matching recipes

Exceptions	If the user selects an invalid or non-existent cuisine, the application displays an error message and prompts the user to select a valid cuisine

UC5	**Choose a Recipe**
Actors	User
Entry Condition	The user has already inputted ingredients and specified or not a type of cuisine
Flow of Events	1. The application displays a list of maximum five recipes 2. User selects one recipe 3. User submit his selection to the application
Exit Condition	The application displays the recipe name, ingredients, cooking instructions, and cooking time
Exceptions	If the user selects none, invalid or non-existent recipe, the application displays an error message and prompts the user to select a valid recipe

From the scenario 2:

UC6	**Create Meal Plan**
Actors	User
Entry Condition	The user has logged into their Differeat account.
Flow of Events	1. The user accesses the "Meal Plan" feature in the Differeat application. 2. The user selects the desired recipes for their meal plan, specifying the meals they want to prepare for the week. 3. The user reviews the selected recipes and confirms their meal plan. 4. The application generates a weekly meal plan based on the selected recipes.
Exit Condition	The application displays the complete meal plan for the week, including the chosen recipes for each meal.
Exceptions	If the user does not select any recipes or provides incomplete information, the application displays an error message and prompts the user to select valid recipes and specify all required details.

UC7	**Generate Shopping List**
Actors	User
Entry Condition	The user has a confirmed meal plan for the week.

Flow of Events	1. The user navigates to the "Shopping List" feature in the Differeat application. 2. The application retrieves the ingredients needed for the selected recipes in the user's meal plan. 3. The user reviews the generated shopping list and has the option to modify quantities or remove specific items if necessary. 4. The user finalizes the shopping list.
Exit Condition	The application presents the user with a complete shopping list that includes all the necessary ingredients for their chosen recipes.
Exceptions	I If there are no ingredients required for the selected recipes, the application displays a message indicating that no shopping list is available.

UC8	**Shopping List Management**
Actors	User
Entry Condition	The user has a generated shopping list.
Flow of Events	1. The user opens the shopping list in the Differeat application. 2. The user goes to the grocery store and checks off the items from their shopping list as they purchase them. 3. If the user cannot find a specific item or decides not to purchase it, they have the option to remove it from the shopping list. 4. After completing their shopping, the user confirms that they have obtained all the necessary items.
Exit Condition	The user's shopping list is updated to reflect the items they have purchased or removed.
Exceptions	If the user encounters any issues or discrepancies with the shopping list, they can report them through the application's support or feedback channels.

From scenario 3:

UC9	**Save Recipe as Favourite**
Actors	User
Entry Condition	The user is logged into his account and viewing a recipe.
Flow of Events	1. The user selects the "Save as Favourite" option for a recipe. 2. The application associates the selected recipe with the user's account and adds it to his favourites.

	3. The application confirms the successful addition of the recipe to the user's favourites.
Exit Condition	The selected recipe is successfully saved as a favourite for the user.
Exceptions	If the recipe is already saved as a favourite, the application displays a message indicating that the recipe is already in the user's favourites.

UC10	**Manage Favourite Recipes**
Actors	User
Entry Condition	The user is logged into their account and accessing the "Favourites" section.
Flow of Events	1. The user navigates to the "Favourites" section in the application. 2. The application displays a list of the user's favourite recipes. 3. The user can perform actions such as removing a recipe from their favourites, editing recipe details, or organizing the favourite recipes. 4. The application updates the user's favourites list according to the performed actions.
Exit Condition	The user successfully manages their favourite recipes, including removing recipes, editing details, or organizing the list.
Exceptions	If the user has no favourite recipes, the application displays a message indicating that there are no recipes saved as favourites.

UC11	**View Nutritional Information**
Actors	User
Entry Condition	The user is viewing a recipe.
Flow of Events	1. The user selects the "View Nutritional Information" option for a recipe. 2. The application retrieves the nutritional information for the selected recipe based on the ingredients and quantities used. 3. The application displays the nutritional information, including details such as calories, macronutrients (carbohydrates, proteins, fats), vitamins, and minerals.
Exit Condition	The user successfully views the nutritional information for the selected recipe.
Exceptions	If the nutritional information is not available for a recipe, the application displays a message indicating that the information is not provided.

From scenario 4:

UC12	Set Dietary Preferences
Actors	User
Entry Condition	The user has logged into their Differeat account.
Flow of Events	1. The user navigates to the "Account Settings" section in the Differeat application. 2. The user selects the "Dietary Preferences" option. 3. The user indicates their specific dietary restrictions, such as being gluten-free and lactose intolerant. 4. The application saves the user's dietary preferences.
Exit Condition	The application saves the user's dietary preferences for future reference.
Exceptions	If the user encounters any issues while setting their dietary preferences, such as invalid selections or technical errors, the application displays an error message and prompts the user to provide valid information.

2.3. Functional requirements

From the prototype scope define on the feasibility study, we refine functionality we need to achieve that:

S1 - User Input:

- R1 - The application should allow users to select ingredients they have from a predefined list of ingredients in a database.

S2 - Recipe Matching:

- R2 - The application should have a matching algorithm that takes the user-inputted ingredients and matches them with recipes in a database.
- R3 - The matching algorithm should prioritize recipes that have a higher number of matching ingredients.
- R4 - The matching algorithm should consider the cuisine preference selected by the user, if any.

S3 - Recipe Display:

- R5 - The application should display the top 5 matching recipes to the user.
- R6 - The application should display the recipe name, ingredients, cooking instructions, and cooking time for each recipe.

S4 - Cuisine Selection:

- R7 - The application should allow users to choose a specific cuisine (e.g., Turkish, Chinese, Italian) for the recipe recommendations.
- R8 - The application should have a database of recipes for each cuisine available.

S5 - Basic User Management:

- R9 - The application should allow users to create a new account and login.
- R10 - The application should allow users to update their account information (e.g., name, email address, password).

For further development these functionalities will be covered by these other requirements:

S6 - Favourite Recipes:

- *R11 - The application should allow users to save recipes as their favourites, associating them with their user account.*
- *R12 - The application should provide a "Favourites" section where users can view and manage their saved favourite recipes.*
- *R13 - The application should allow users to remove recipes from their favourites list if desired.*

S7 - Search History:

- *R14 - The application should provide a search history feature that records and displays the user's previous search queries.*
- *R15 - The search history feature should include information such as the search keywords, date and time of the search, and the matching recipes.*
- *R16 - The application should allow users to clear their search history if desired.*

S8 - Personalization

- *R17 - The application should provide an interface for users to set their dietary preferences and restrictions, such as vegetarian, vegan, gluten-free, and others.*
- *R18 - The system should use the user's dietary preferences and restrictions to generate personalized recipe recommendations that align with their chosen preferences.*

S8 - Notifications

- *R19 - The application should send notifications to users for updates on new recipes, personalized recommendations, and alerts related to their favourite recipes.*
- *R20 - Users should be able to manage their notification preferences, such as enabling or disabling specific types of notifications.*

S9 - Nutritional Information:

- *R21 - The application should display nutritional information for each recipe, including details such as calories, macronutrients (carbohydrates, proteins, fats), vitamins, and minerals.*
- *R22 - The nutritional information should be based on the ingredients and quantities used in the recipe.*

S10 – Shopping list

- *R23 - The application should provide a shopping list feature that generates a list of ingredients required for the selected recipes.*
- *R24 - Users should be able to view and manage their shopping list, including adding, removing, and updating quantities of ingredients.*

S11 – Quantity management

- *R25 - The application should provide a quantity management feature that allows users to specify the quantity of each ingredient they have and adjust it as needed.*
- *R26 - Users should receive reminders or notifications when the quantity of certain ingredients in their inventory falls below a specified threshold.*
- *R27 - The admin management section should allow administrative users to manage user accounts, including creating, editing, and deleting accounts.*

S12 – Admin Management Section

- *R28 - The admin management section should provide functionalities to manage the ingredient and recipe database, including adding new ingredients, updating existing ingredients, and removing ingredients that are no longer needed.*
- *R29 - Administrative users should be able to add, edit, and delete recipes from the database, ensuring the availability of up-to-date recipes for users.*
- *R30 - The admin management section should have proper authentication and access controls to restrict access to authorized administrative users only.*

* Sn for scope number n ** Rm for requirement number m

2.4. Non-Functional requirements:

Usability
The "Differeat" application prioritizes usability and user experience to create an intuitive and enjoyable cooking experience for students. With a user-friendly interface, clear input processes, personalization options, and responsive design, the app offers a seamless and tailored experience. Efficient performance, visual appeal, contextual help, and social media integration further enhance user engagement. By continuously gathering user feedback and implementing improvements, "Differeat" strives to deliver a user-centric app that empowers students to cook delicious, healthy meals with ease and satisfaction.

Performance
The "Differeat" application places a strong emphasis on performance, aiming to provide users with a seamless and efficient cooking experience. Through responsiveness, minimized loading times, swift search and filtering capabilities, and seamless data synchronization, the app ensures a smooth user journey. Scalability, network efficiency, error handling, and continuous optimization contribute to a reliable and high-performing application. By prioritizing performance optimization, "Differeat" strives to deliver an exceptional user experience, empowering students to access recipes and navigate the app effortlessly and without frustration.

Reliability
Reliability is a cornerstone of the "Differeat" application, establishing trust and dependability for users. Through high uptime, effective error handling, crash recovery, data integrity measures, fault tolerance, and performance monitoring, the app ensures a consistent and stable user experience. The implementation of version control, disaster recovery planning, continuous testing, and user feedback mechanisms further enhance reliability. By prioritizing reliability, "Differeat" fosters user confidence, ensuring uninterrupted access to recipes, protecting user data, and delivering a trustworthy cooking companion that students can rely on.

Supportability
Supportability lies at the core of the "Differeat" application, providing a solid framework for maintenance, user support, and future growth. With **a modular and maintainable design, version control, comprehensive documentation, and effective error reporting, "Differeat" ensures ease of maintenance and updates**. Bug tracking, training materials, and a responsive support system enhance user assistance. Scalability, continuous improvement, and a vibrant user community further contribute to the application's supportability, empowering users with seamless support and fostering an environment of ongoing development and user satisfaction.

Implementation
The implementation phase is crucial for the success of the "Differeat" application, laying the groundwork for a seamless cooking experience. By carefully considering the technology stack, architecture, security measures, and database optimization, the application is designed to meet scalability, performance, and data integrity requirements. Adhering to coding standards, conducting thorough quality assurance, and integrating external services efficiently ensures a solid and reliable codebase. Leveraging performance optimization techniques, cloud deployment, and a continuous integration and deployment pipeline, the application is fine-tuned for optimal performance and ease of updates. Comprehensive technical documentation

and knowledge transfer enable efficient maintenance and support. With a focus on implementation, "Differeat" emerges as a robust application, providing users with a trustworthy and effortless cooking companion.

Interface
The interface of the Differeat application takes centre stage, offering a user-focused experience that facilitates effortless cooking. With an intuitive and user-friendly design, responsive layout, and consistent visual aesthetics, Differeat ensures users can navigate the app with ease. The interface prioritizes accessibility, adhering to WCAG 2.1 guidelines, making it inclusive for users with disabilities. Clear and readable text, along with multimedia integration, enhances the overall user experience. Proper error handling and feedback mechanisms help users understand and resolve issues effectively. The interface supports localization and internationalization, catering to a global audience. Usability testing and user feedback play a vital role in continuous improvement, ensuring the interface evolves with user needs. By emphasizing these interface requirements, Differeat creates a seamless and engaging interface that enhances the joy of cooking for all users.

Security And Privacy
Security and privacy are paramount in the design of the Differeat application, aiming to protect user data and foster trust. By implementing robust encryption, secure authentication, and access control mechanisms, Differeat ensures the confidentiality and integrity of user information. Secure coding practices and API protection mitigate common vulnerabilities and safeguard against unauthorized access. Payment processing adheres to industry standards, ensuring the security of financial transactions. Transparent privacy practices, consent management, and data protection measures establish user confidence. Anonymization and aggregation of data protect user privacy while enabling valuable analytics. **Compliance with data protection regulations such as GDPR and CCPA ensures adherence to legal requirements.** Regular security audits and updates maintain a robust security posture. By addressing these security and privacy requirements, Differeat cultivates a safe and private environment, allowing users to confidently engage with the application and focus on their cooking experience.

Packaging
The packaging of the Differeat application plays a vital role in ensuring a seamless deployment process and enhancing the user experience. By addressing various packaging requirements, the application can be efficiently distributed, installed, and updated on different platforms. Compatibility with target platforms, maintaining file integrity, and optimizing package size are essential considerations. The installation process should be user-friendly, providing clear instructions and progress indicators. Versioning and updates enable continuous improvements and bug fixes. Implementing digital signatures enhances security and authenticity. Including licensing and copyright information ensures compliance and protection of intellectual property rights. Packaging should align with the requirements of distribution channels, such as app stores, and facilitate the uninstallation process. Documentation and support resources assist users in installing and using the application. By focusing on these packaging requirements, the Differeat application can provide a smooth and hassle-free deployment experience, ensuring user satisfaction and adoption.

2.5. System model

Class diagrams are part of the Unified Modeling Language (UML) and are used to visualize the structure of a system by showing the classes, their attributes, methods, and relationships.

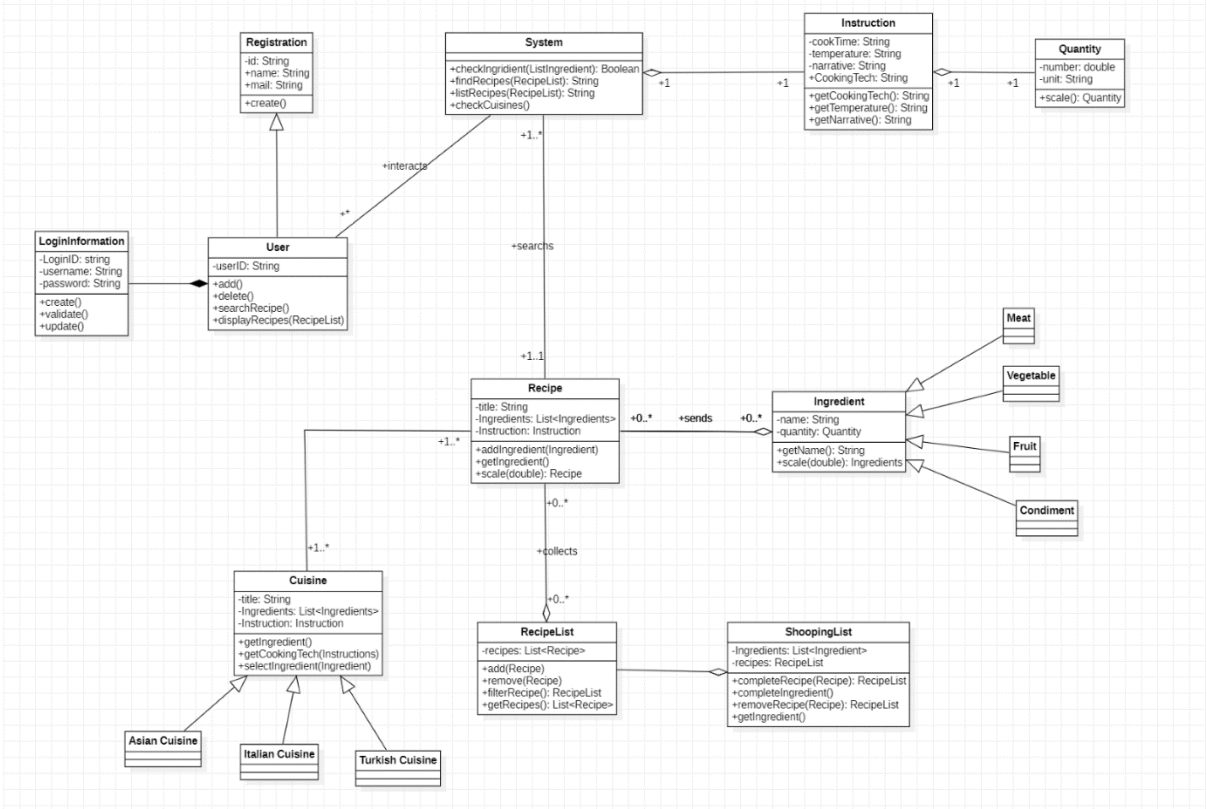
They include classes, associations, inheritance, and multiplicity. Class diagrams help in designing and understanding the static structure of a system, including the relationships between classes.

Sequence diagrams in UML are used to model the dynamic behavior of a system by illustrating how objects interact over time.They include lifelines (representing objects), messages (communication between objects), activation bars (duration of an object's activity), and constraints.Sequence diagrams are valuable for visualizing the flow of messages and the order of interactions between objects during the execution of a use case or scenario.

UML can be used to represent the overall architecture of a system, including its high-level components, their interactions, and dependencies. UML architectural diagrams may include component diagrams (showing system components and their relationships), deployment diagrams (depicting the physical deployment of components), and package diagrams (organizing and grouping related classes and components). UML architecture diagrams are essential for understanding and communicating the system's structure and how various parts of the system work together.

Class Diagram

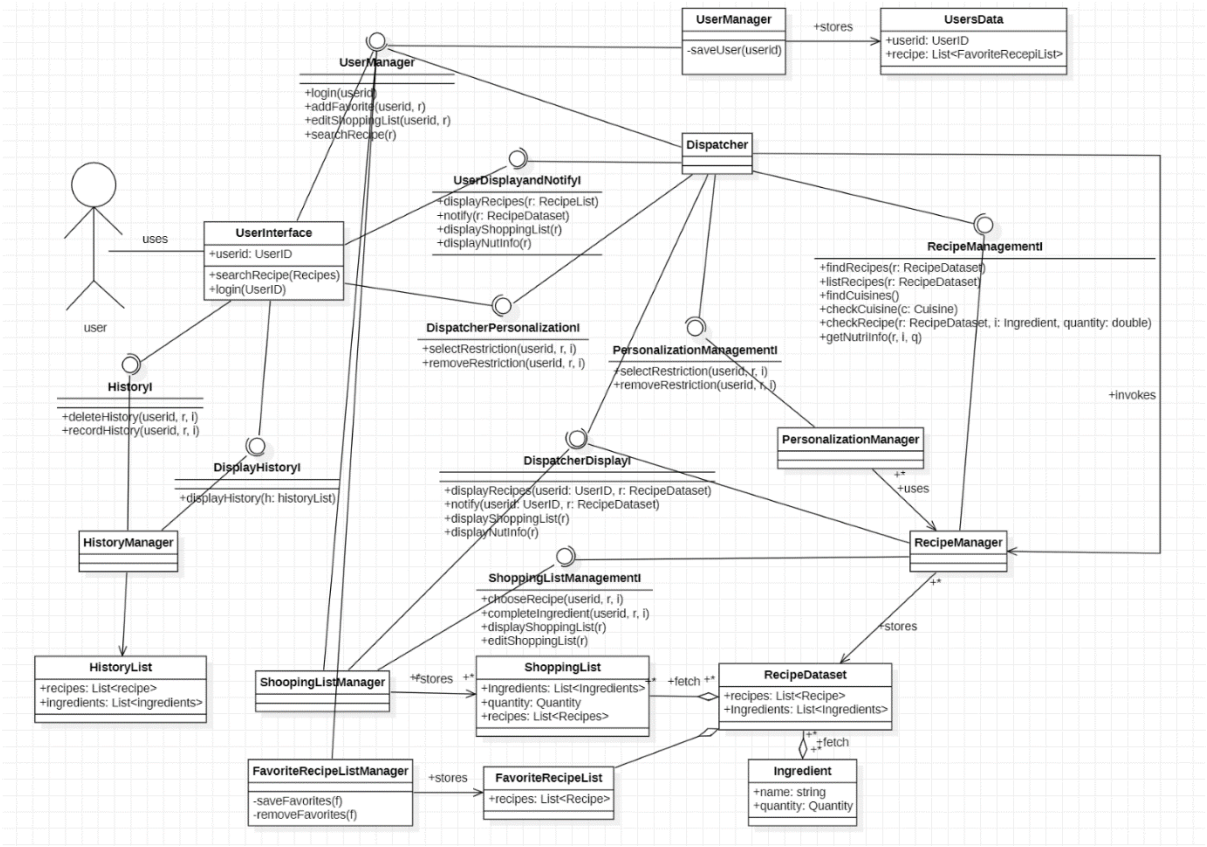
It represents all the classes and subclasses that needed and interacts with each other such as Recipe, Ingredient, Cuisine etc.



UML Architecture

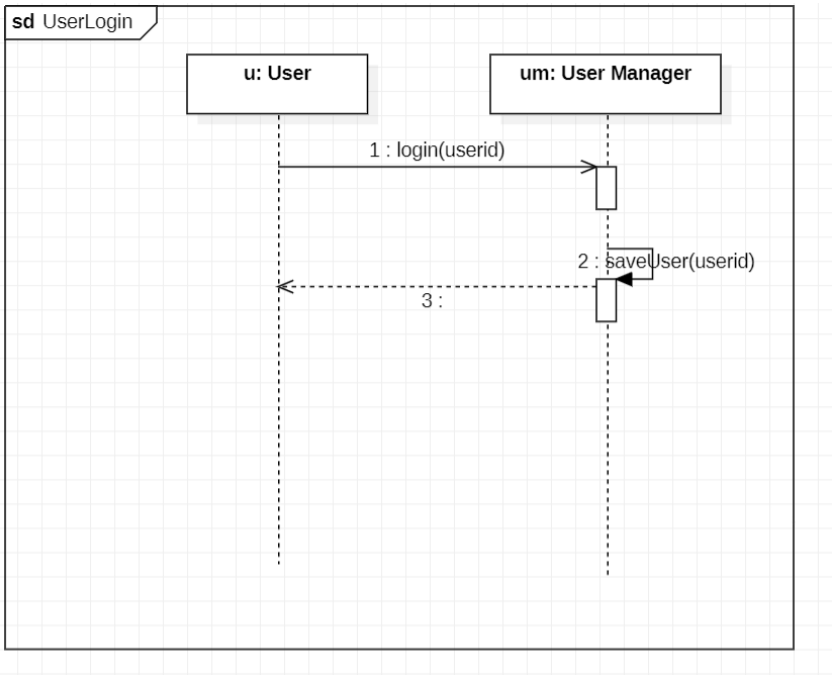
The requriements are seen in this architecture from S1 to S10. S11 and S12 are not represented for the sake of simplicity.In this case, “User Manager Interface” interacts with Shopping List, Favorite Recipe List and Dispatcher. The notify operations stand for notification

requirements. “Dispatcher Display Interface” evokes by both Shopping List and Recipe Manager.

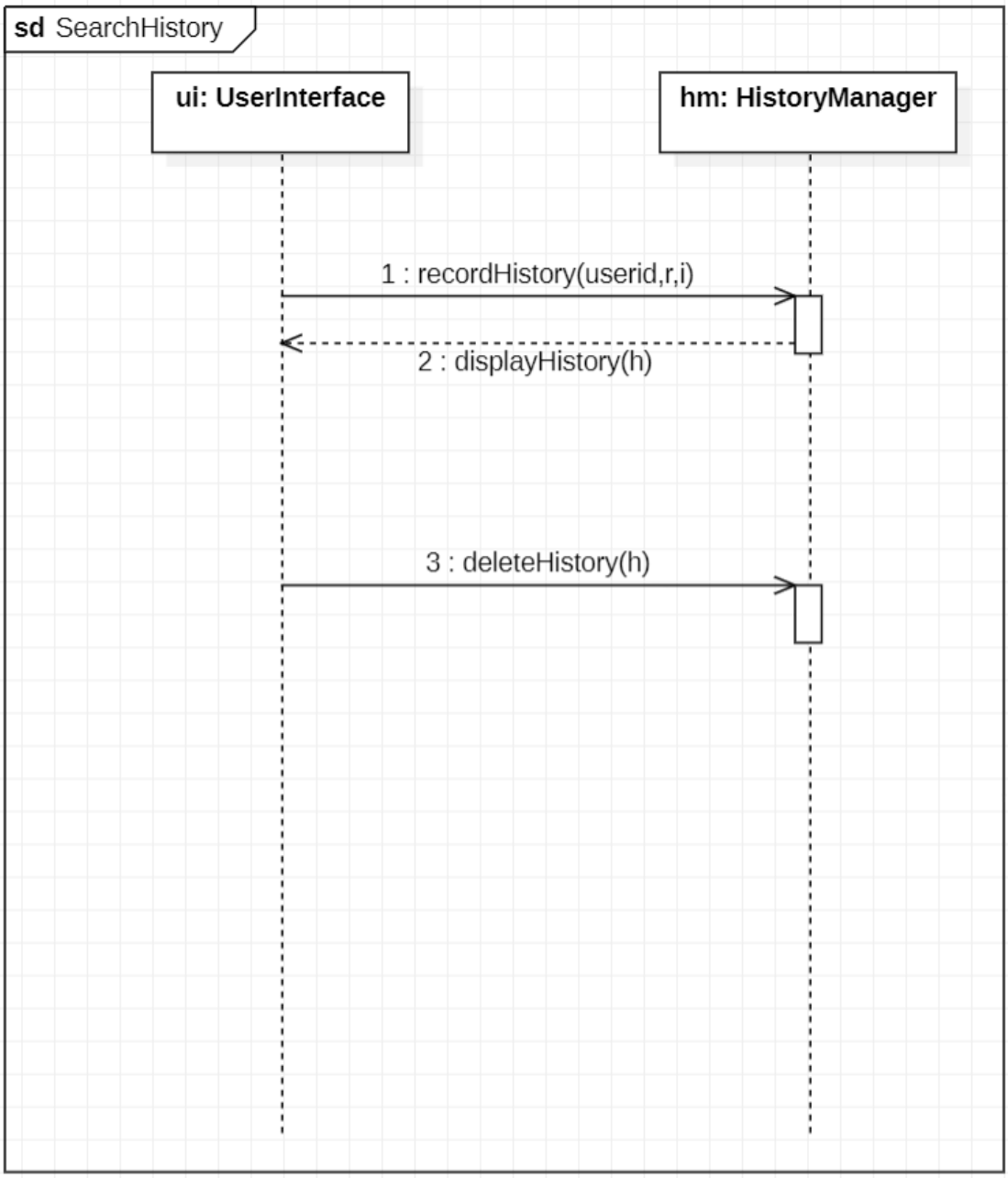


Sequence Diagram

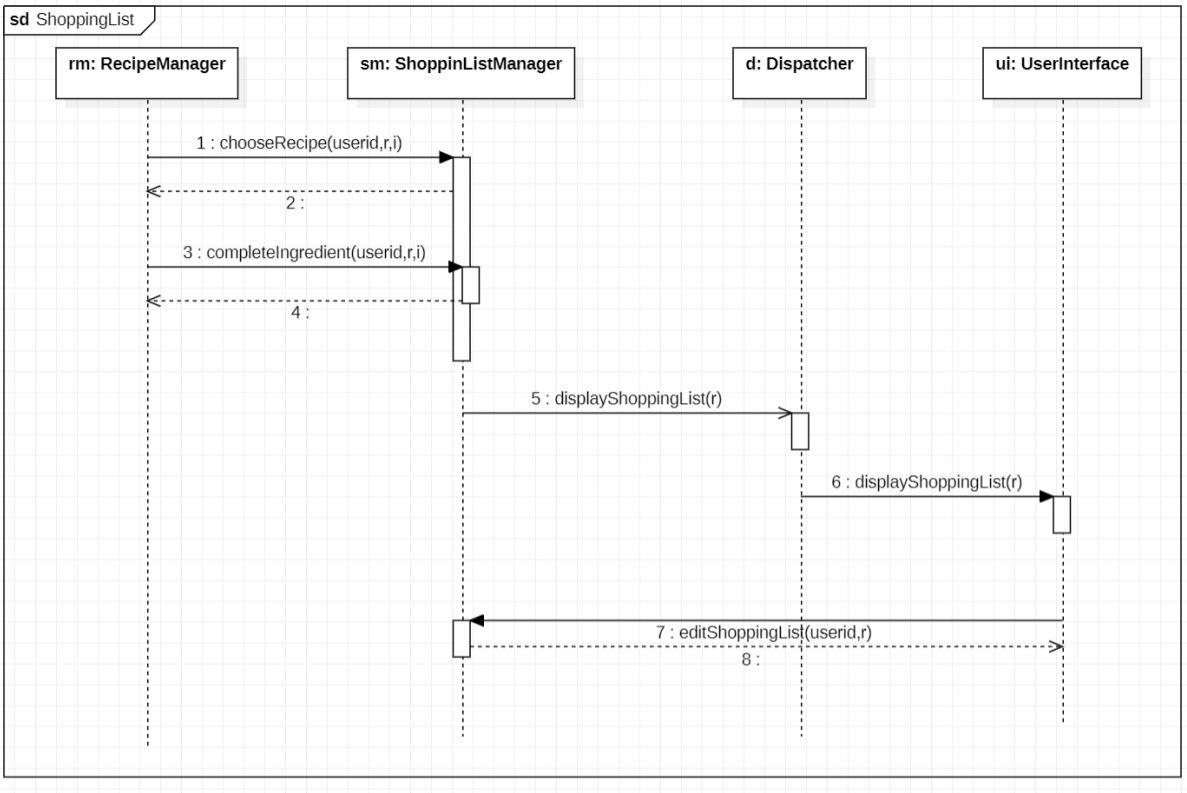
- **S5;**



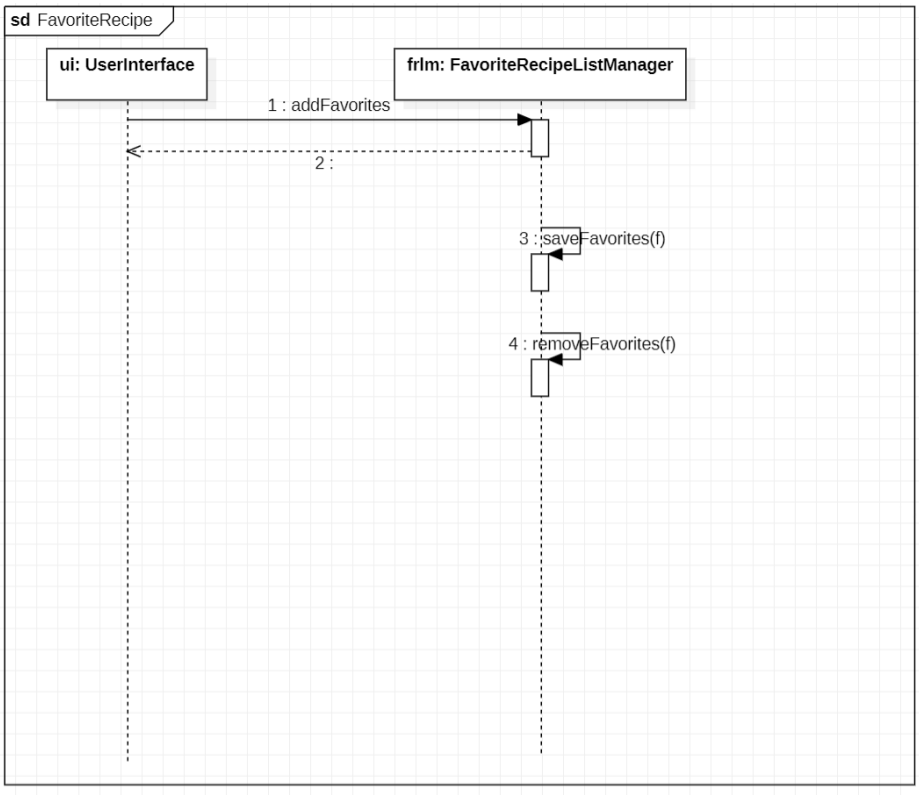
- **S7;**


sd SearchHistory
ui: UserInterface
hm: HistoryManager
1 : recordHistory(userid,r,i)
2 : displayHistory(h)
3 : deleteHistory(h)

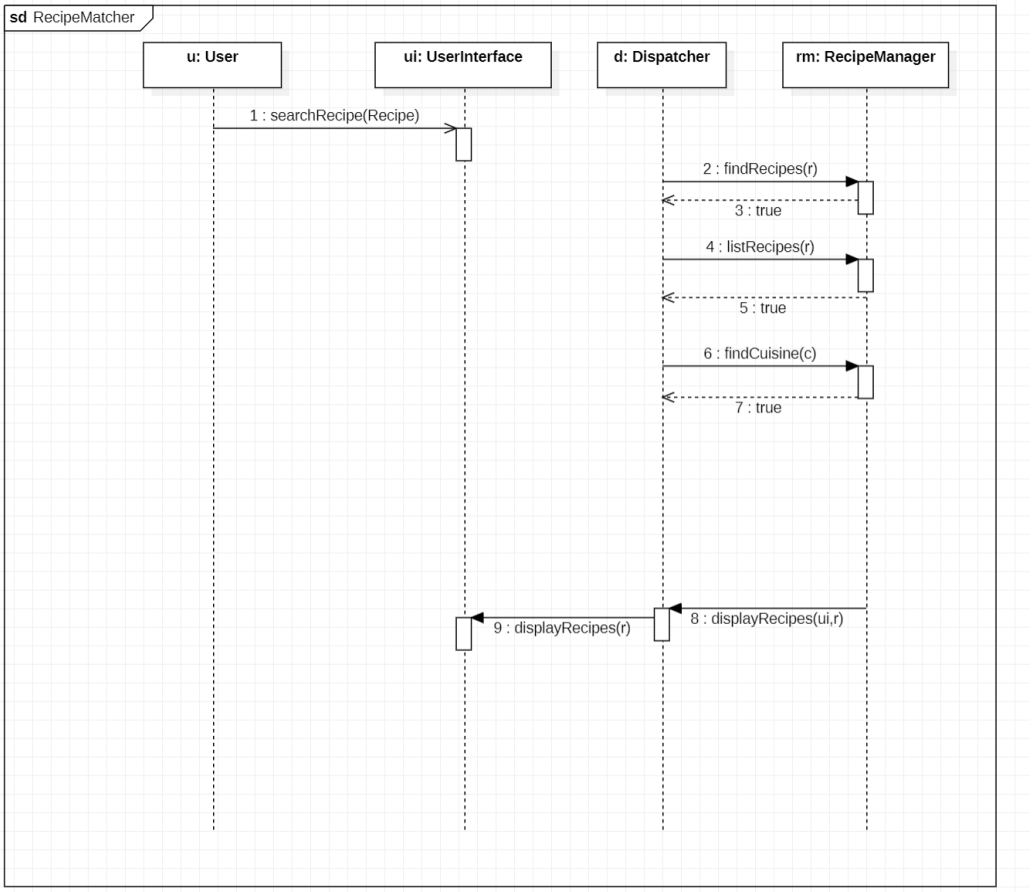
- **S10;**


sd ShoppingList
rm: RecipeManager
sm: ShoppinListManager
d: Dispatcher
ui: UserInterface
1 : chooseRecipe(userid,r,i)
2 :
3 : completeIngredient(userid,r,i)
4 :
5 : displayShoppingList(r)
6 : displayShoppingList(r)
7 : editShoppingList(userid,r)
8 :

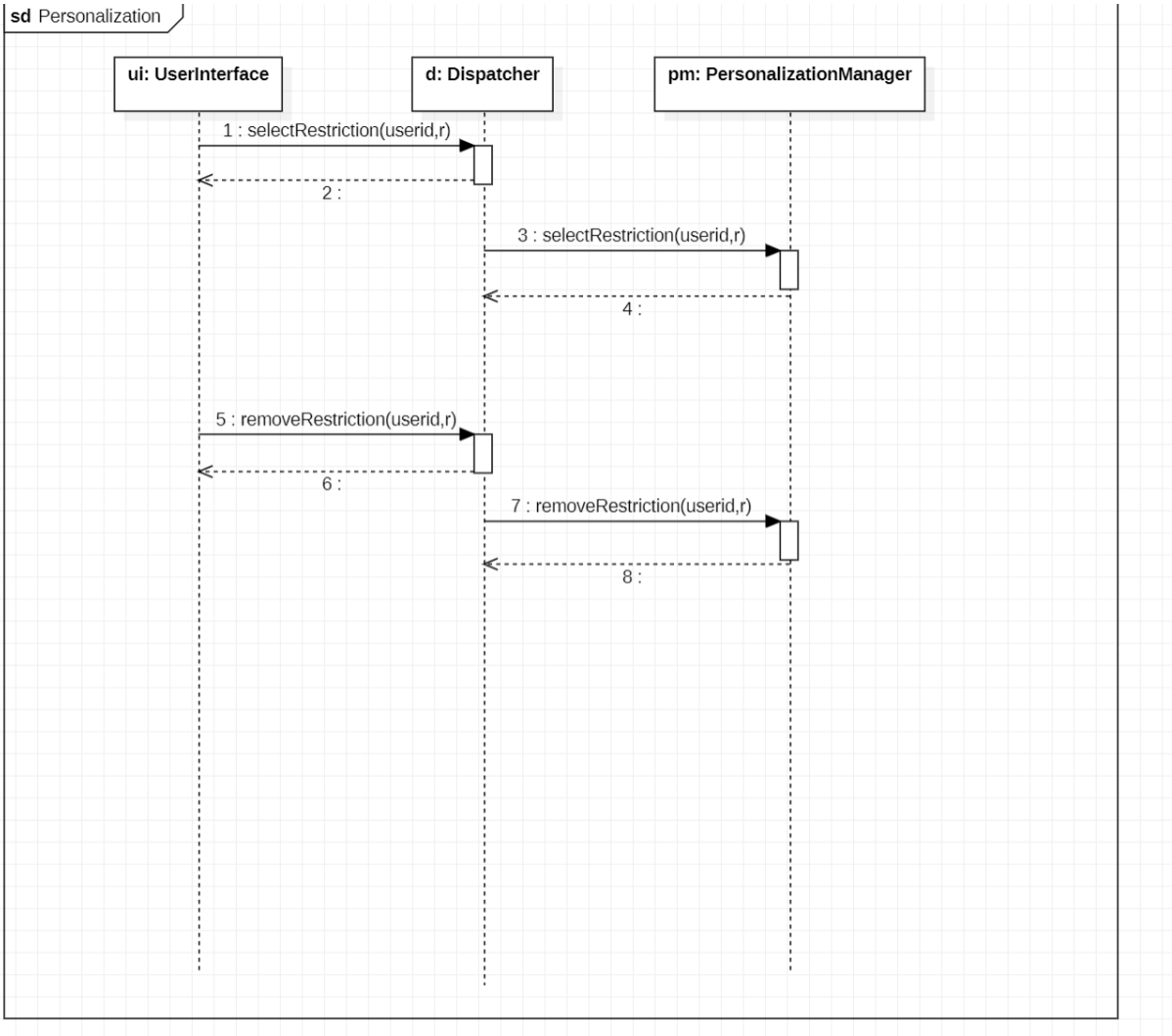
S6;


sd FavoriteRecipe
ui: UserInterface
frlm: FavoriteRecipeListManager
1 : addFavorites
2 :
3 : saveFavorites(f)
4 : removeFavorites(f)

- **S1,S2,S3,S4,S9 and S10;**



- **S8;**



3. Implementation

3.1. Framework and Language

Initially, Java was used at the beginning of the project. However, due to Flutter's ability to create visually appealing and user-friendly applications, it became evident that Flutter was a more accessible choice compared to Java.
Java has some disadvantages compared to Flutter for mobile app development:

1. Platform Specific: Java is primarily for Android, requiring a separate technology for iOS. Flutter offers cross-platform development.
2. UI Development: Java can be more code-intensive for UI, while Flutter offers a more efficient and flexible approach.
3. Performance: Flutter's compiled code can be more efficient than Java in some cases.
4. Learning Curve: Dart (Flutter's language) is often considered easier to learn than Java for mobile development.

5. Community and Ecosystem: Flutter has a growing and active community with numerous packages, simplifying development. Java's ecosystem is also robust but more fragmented.
6. Development Speed: Flutter's hot reload speeds up development, while Java often involves longer build times.
7. Maintenance: Java-based apps may require more maintenance with new Android versions and devices. Flutter simplifies maintenance with a single codebase.

1. Flutter Framework

Flutter is an open-source user interface (UI) framework created by Google for building natively compiled applications for mobile, web, and desktop from a single codebase. It allows developers to create high-quality, visually appealing apps with a consistent user experience across different platforms using the Dart programming language.

Advantages of Flutter:

1. **Single Codebase, Multiple Platforms:** With Flutter, we can simultaneously build for the Android and iOS platforms thanks to its single codebase strategy. This quickens development, lowers maintenance requirements, and guarantees a consistent user experience across all platforms.
2. **Native Performance:** Flutter apps are created using native ARM code, resulting in fast animations and good performance. This is essential for creating an engaging and responsive user interface, especially when users interact with ingredient lists and recipe recommendations.
3. **Wide Range of Widgets:** Flutter has a wide library of widgets. With the help of these widgets, we can create complex and captivating UI elements like ingredient cards, recipe previews, and interactive buttons while still maintaining a consistent design.
4. **Hot Reload for Iterative Development:** The hot reload function in Flutter allows us to see code changes in real-time, accelerating experimentation and development. When perfecting the app's design, behavior, and user flow, this is especially helpful.
5. **UI Consistency**: Flutter guarantees a uniform appearance and feel on several platforms, reducing the need for platform-specific adjustments. This benefits our target audience of students who use a variety of gadgets.

2. Dart Programming Language

Dart is a programming language developed by Google, known for its use in the Flutter framework for building cross-platform mobile, web, and desktop applications. It is designed for ease of development, high performance, and scalability, making it suitable for a wide range of application types.

1. **Object-Oriented Approach:** Dart's object-oriented design complements the way the data and logic in our application are organized. By modeling ingredients,

recipes, and user profiles as objects, we may improve the organization and maintainability of our code.

2. **Strongly Typed for Safety:** The static type system of Dart helps developers identify problems early on, lowering the risk of runtime crashes or unexpected behaviour. This makes the application more dependable and stable.

3. **Async Programming with Ease:** The async and await keywords in Dart make it simple to handle asynchronous operations. This is important when obtaining information from databases or outside APIs, like in the case of ingredient searches and recipe suggestions.

4. **Expressive and Readable Code:** Dart's clear and modern grammar makes coding easier and encourages readability. This is helpful when working with a team or going back to the codebase for upkeep and updates.

5. **Growing Community and Google Support:** Google supports Dart, which is growing popularity among developers. Google's ongoing development and support guarantee that we'll have access to resources and upgrades for many years to come.

It is setting up "Differeat" application for success by taking advantage of Flutter's skills for providing cross-platform, performance-driven UIs and Dart's abilities in object-oriented programming, asynchronous operations, and readability. This technological stack will enable us to effectively construct an app that serves the special demands of students looking for simple and delicious meal alternatives using the products they have on hand.

3. Included Requirements

It's common in software development to start with a subset of features and gradually add more functionality as the project progresses. It seems that during the development of our recipe application, we initially implemented the following scopes and their associated requirements:

- User Input (Scope S1)
- Recipe Matching (Scope S2)
- Recipe Display (Scope S3)
- Cuisine Selection (Scope S4)
- Favourite Recipes (Scope S6)

This approach, often referred to as iterative development, allows us to release a functional version of our application more quickly and gather user feedback. Based on user feedback and evolving project priorities, we can then decide which additional features to implement in subsequent iterations.

If we plan to add the remaining scopes and requirements in the future, it's essential to maintain good documentation and a roadmap for our application's development. This will help ensure that we have a clear plan for implementing the remaining features and that we can efficiently track and manage the development process.

Remember to gather feedback from our users as we introduce new features, as this can help us make informed decisions about the direction of our application and prioritize what's most valuable to our user base.

3.2. Coding Part

1.1 Structure of Code

2.1.1 App_state Class:

- Represents a state management class for the application.
- Manages various application states and user interactions.
- Extends ChangeNotifier to notify listeners about state changes.
- Contains properties and methods such as loadingSearch, searchTextFocusNode, searchTextController, selectedBottomNavbarIndex, selectedCuisines, selectedIngredients, results, favorites, etc.
- Provides methods for managing user selections, updating state, and handling favorites.

2.1.2 Recipe_model Class:

- Represents the RecipeModel class for storing recipe data.
- Contains properties like title, ingredients, directions, link, source, ner, cuisine, etc.
- Provides methods for converting between JSON data and class objects (toMap, fromJson, toJson).
- Implements == and hashCode methods for object comparison.

2.1.3 Recipe_Manager Class:

- Represents a class to manage recipe data.
- Contains methods for reading and searching recipes.
- Utilizes readData to read and parse JSON recipe data.
- Utilizes searchRecipe to filter recipes based on ingredients and cuisine types.

2.1.4 Favorite_Manager Class:

- Represents a class for managing favorite recipes.
- Utilizes SharedPreferences to store and retrieve favorite recipes.
- Contains static methods like getFavoriteRecipes, addFavorite, removeFavorite.

2.1.5 Screens (Search, Recipe, Home, Favorite) Classes:

Search Class:

- Represents a search screen for finding recipes.
- Users can search by ingredients and cuisine types.
- Displays search results and provides interactions.

Recipe Class:

- Represents a screen showing recipe details.

- Displays recipe title, ingredients, directions, and link.

Home Class:

- Represents the main application screen.
- Contains navigation bars and content areas.

Favorite Class:

- Represents a screen for displaying favorite recipes.

2.1.6 Main Class:

- Represents the entry point of the Flutter application.
- Defines the MyApp class.
- Utilizes ChangeNotifierProvider to manage application state.
- Uses MaterialApp to manage theme and navigation.
- Sets up the home screen as the Home widget.

This structured outline gives a clear overview of the components and classes within the application. Each class has a distinct role and contributes to the functionality and user experience of "Differeat" recipe application. State management, data models, recipe management, and UI representation are all carefully organized to create a seamless and user-friendly experience for the application users.

2.2 Overview of Code

2.2.1 App_state of Class

This code represents a "state management" class that can be used in Flutter. State management is a method for maintaining and updating application state by monitoring data changes in an application and notifying all application parts of these changes.

The above code defines a class called `AppState`. This class extends the `ChangeNotifier` class so that when the data in it changes, it notifies listeners (for example Flutter widgets) by calling the `notifyListeners` method. This way, application screens can be updated in response to status changes.

The properties and methods in the `AppState` class are as follows:

1. `loadingSearch`: A flag (boolean value) used while performing a search, waiting for search results.

2. `searchTextFocusNode`: Focus button for a text input field.

3. `searchTextController`: Text controller for a text input field.

4. `selectedBottomNavbarIndex`: The index of the selected item in the footer navigation bar.

5. `selectedCuisines`: List of cuisine types selected by the user.

6. `selectedIngredients`: List of materials added by the user.

7. `results`: List of recipes found by the app.

8. `favorites`: List of recipes that the user has marked as favourites.

9. The `AppState` class has a constructor method. This method takes the user's favorite recipes when the application is started and loads them into the `favorites` list.

10. `addCuisine`, `removeCuisine`: Methods for adding and removing selected cuisine types.

11. `addFavorite`, `removeFavorite`: Methods to add and remove favorite recipes.

12. `updateLoading`: Method to update `loadingSearch` value.

13. `updateResults`: Method to update the `results` list.

14. `clearResult`: Method to clear the `results` list.

15. `updateSelectedNavbarIndex`: Method to update the selected footer navigation bar item.

16. `addIngredients`, `removeIngredients`, `clearIngredients`: Methods for adding, removing and cleaning selected materials.

This piece of code contains the core functionality used to manage states in the application. State management becomes an essential component for efficiently and regularly managing application state in large and complex applications. This piece of code becomes better understandable and valuable when used in a wider context in an application.

2.2.2 Recipe_model of Class

This code represents a `RecipeModel` class that can be used in Flutter applications. This class is used to represent recipe data.

Inside the `RecipeModel` class are the following properties:

1. `title`: The title (String) of the recipe.

2. `ingredients`: The list of ingredients used in the recipe (List<String>).

3. `directions`: The preparation method or instructions of the recipe (List<String>).

4. `link`: The link or source (String) of the recipe.

5. `source`: The source (String) of the recipe.

6. `ner`: List of Named Entity Recognition of the recipe (List<String>).

7. `cuisine`: The culinary type (String) of the recipe.

The class's constructor method (`RecipeModel`) provides access to these properties and creates a recipe instance. Also, the class has the ability to convert between JSON data and the class object. There are `toMap` and `fromJson` methods for this.

- `toMap`: Converts class properties to a `Map` data.

- `fromJson`: Takes JSON data and parses it as `Map`, then creates a `RecipeModel` object using this `Map` data.

Also, the `toJson` method converts the `toMap` result to JSON format.

The class also implements the `==` and `hashCode` methods. The `==` method is used when comparing the contents of two `RecipeModel` objects, while the `hashCode` method is used to get a unique hash value of the object.

This `RecipeModel` class can represent data for recipes in an orderly fashion and can help manage recipes within the application. JSON conversion makes this class easy to use in applications such as storing data or exchanging data from the server.

2.2.3 Recipe_Manager of Class

This code represents the RecipeManager class, which is a recipe data manager. This class is used to read and search recipes data in the app. Recipes data is stored in a file in JSON format and this class reads and processes data from this file.

Methods inside the RecipeManager class:

readData: It is an asynchronous method that reads recipe data in JSON format. This method takes the dataset from "assets/data/dataset.json", parses the JSON data and converts it into RecipeModel objects. As a result, all recipes are returned as a list.

searchRecipe: It is an asynchronous method that filters recipes by given ingredients (ingredients) and selected cuisine types (selectedCuisines). This method gets all recipes using the readData method. Then it filters the recipes according to the given ingredients and filters once again according to the selected cuisine types. As a result, a list of recipes that meet the appropriate criteria is returned.

This piece of code can be used to provide data management for recipes in the app. Along with the RecipeModel class, it can be used to visualize recipes in the app and allow the user to search for recipes by selecting specific ingredients and cuisine types. Storing the data in a file in JSON format makes it easy to modify and update the dataset. The RecipeManager class is a useful class for managing operations such as retrieving and filtering a dataset.

2.2.4 Favorite_Manager of Class

This code represents a class called FavoriteManager. This class contains static methods used to manage favorite recipes. SharedPreferences are used to keep and store favorite recipes within the app.

Static methods in FavoriteManager class:

favoriteKey: Represents the key value to be used to store favorite recipes. This value is used in the area where SharedPreferences and favorite recipes are stored.

getFavoriteRecipes: It is a static asynchronous method that takes data from the storage area of favorite recipes and returns it as a RecipeModel list. Using SharedPreferences, it takes JSON data of favorite recipes, analyzes this data and converts it into RecipeModel objects.

addFavorite: Static asynchronous method used to add a new favorite recipe. This method first gets the current favorite recipes using the getFavoriteRecipes method, adds the new recipe to this list, and then saves the updated favorite recipe list to SharedPreferences in JSON format.

removeFavorite: Static asynchronous method used to delete favorite recipes. This method first retrieves the current favorite recipes using the getFavoriteRecipes method, removes the

specified recipe from the list and saves the updated favorite recipe list back to SharedPreferences in JSON format.

This code is used within the app to allow users to add, remove and manage their favorite recipes. SharedPreferences are an easy and common method for persistently storing small amounts of data (key-value pairs). Therefore, it is preferred as a convenient method to save the user's favorite recipes on their device.

2.2.5 Screens of Class

`Search` Class:

This class represents a search screen where the user can search for recipes by ingredients and cuisine types. Users can add ingredients in the search box, see ingredients with chips, and filter recipes based on selected ingredients and cuisine types. It also displays cards listing search results and provides interaction to add or remove favorite recipes.

`Recipe` (Recipe) Class:

This class represents a screen that shows the details of a selected recipe. The recipe's title, ingredients, instructions, and recipe link are visualized. It also provides functionality to copy the link of the recipe to the clipboard.

`Home` Class:

This class represents the main screen of the application. Creates a widget with top navigation bar, bottom navigation bar and content areas. Via the bottom navigation bar, users can navigate between the "Search" and "Favorite" screens. At the same time, the `Consumer` widget, which monitors the application status, listens for status changes and makes instant updates on the screen.

`Favorite` Class:

This class represents a screen where the user can view their favorite recipes. It visualizes the list of favorite recipes and users can interact on the recipes. It also represents recipe data, used in conjunction with the `RecipeModel` class.

These four classes together provide the basic features of a recipe app. Users can search for recipes within the application, see their details, mark favorite recipes and view their favorite recipes. This way, users can easily find and manage recipes in the app.

2.2.6 Main of Class

This snippet exposes the `MyApp` class, which represents the main entry point of a Flutter application. The `MyApp` class makes the application launch and display the home screen (`Home`). It also sets the theme and state management for the application.

Important elements found inside the `MyApp` class:

1. `ChangeNotifierProvider`: A `ChangeNotifierProvider` created with the `AppState` class is used to manage application state and track state changes across the entire widget tree. As part of the `ChangeNotifierProvider`, the `AppState` object represents the state of the application and passes this state to all child widgets.

2. `MaterialApp`: This widget is the core widget of a Flutter application and manages the theme and navigation throughout the application. `MaterialApp` specifies the application title (`title`), debug mode (`debugShowCheckedModeBanner`), theme settings and home screen (`home`).

3. `theme`: Sets the general theme settings for the application. In this code snippet, a light theme is used. `primaryColor` specifies the color `Colors.orange` as the primary color. Also, the `AppBar` theme is customized and set to `Colors.orange`.

4. `home: const Home()`: Specifies the `Home` widget as the home screen of the application. The `Home` widget is a widget through which the user can navigate and search on the home screen of the application, representing the `Home` class.

This code starts the Flutter application starting with the `MyApp` class, manages the application state and sets the general theme settings for the application. Via `ChangeNotifierProvider` it is ensured that changes in application state are reflected to all child widgets, thus keeping the data within the application up to date.

3.3. User Manual

1. Main Page of Application

It is the first page that appears in front of the user when the application is opened. On this page, the user can first select from which cuisine the recipes can be. In addition, the user can enter the food items he has. The user can search from the search button after entering the materials. This page is the main page of the application, and most of the basic operations that the user will do can be done through this page.

Figure1- Screenshot of Main Page

As seen in the screenshot above, there are fundamental parts on the home page. Each letter represents one part of application.

- (A) In this section, the user should toggle which cuisine to search for the recipes from.
- (B) In this section, the user should add the materials they have here. The most important rule is that the ingredients must be added one by one. After an ingredient is written, it should be saved to the application with the orange plus button on the right.
- (C) In this section, the user can clear the recipes that appear after searching with clear button.
- (D) In this section, the user can search according to the ingredients she/he added with search button.
- (E) In this section, the search button allows the user to return to the main page.
- (F) In this section, the Favorite button launches the page showing the recipes that the user has added to their favourites.

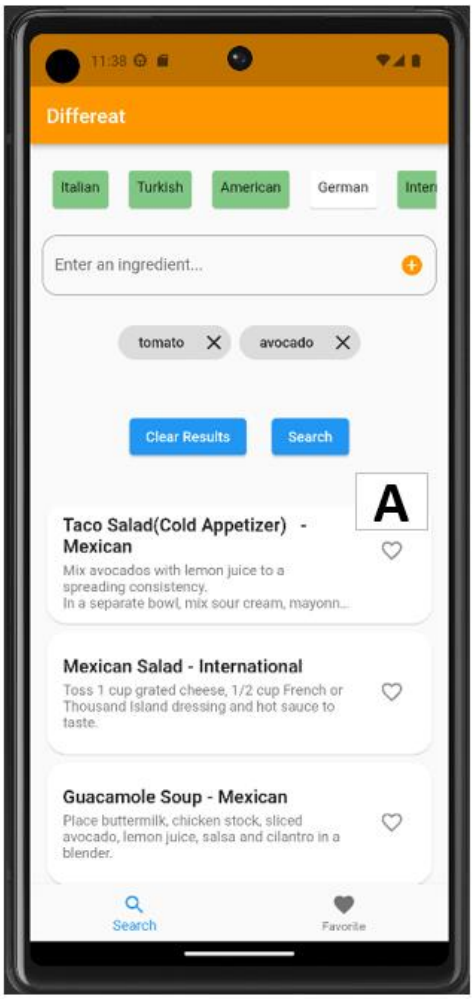


Figure2- Screenshot of Example Search Page

As seen in the screenshot above, an exemplary search has been made. Avocado and tomato were chosen as the ingredient. The results found are listed below in card

format. The user can reach the detailed recipe by pressing the cards of the recipes she/he likes.(A)

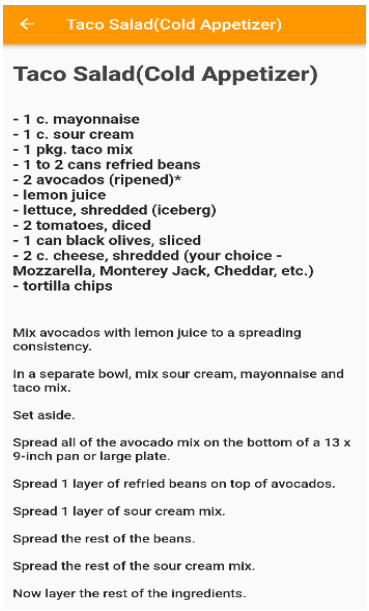


Figure3- Screenshot of Example Recipe Page

In addition, the user adds the recipes to the favorite page when the user presses the heart mark on the right of the cards.

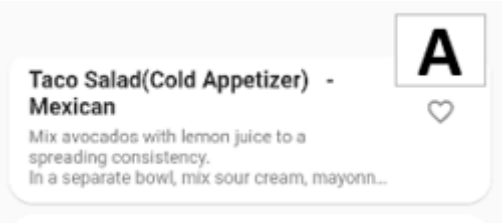


Figure4- Screenshot of Example Recipe Page

2. Favorite Page of Application

On this page, the user can see the favorite recipes that I have added before. In addition, the user can delete any of the favorite recipes on this page from their favorites.

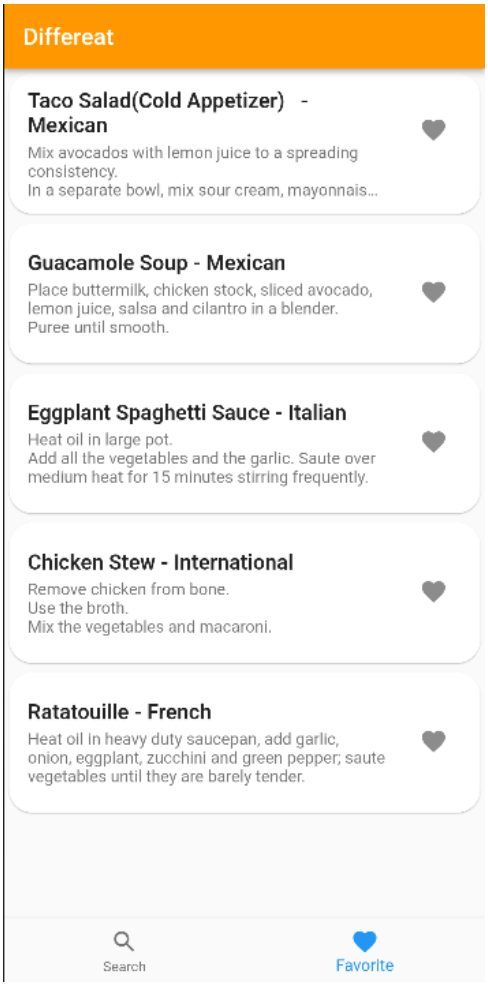


Figure4- Screenshot of Example Favourites Page

3.4. Installation Instructions

1. Open with Android Studio

To open a Flutter mobile app project from a GitHub repository in Android Studio, you can follow these steps:

1. Install Git: Ensure that Git is installed on your computer. If it's not installed, you can download it from https://git-scm.com/downloads and follow the installation instructions for your operating system.

2. Clone the Repository:

- Open a terminal or command prompt.

- Navigate to the directory where you want to store the project.

- Run the following command to clone the GitHub repository:

" git clone https://github.com/bozkurttarda/Differeat1.git"

- This command will download the project files to your local machine.

3. Install Flutter and Dart:

- If you haven't already, make sure Flutter and Dart are installed on your system. You can download and set up Flutter from https://flutter.dev/docs/get-started/install.

4. Open Android Studio:

- Launch Android Studio

5. Open the Project:

- In Android Studio, click on "File" in the top menu.

- Select "Open" and navigate to the directory where you cloned the GitHub repository in step 2.

- Choose the root directory of the Flutter project (the one containing the `pubspec.yaml` file) and click "OK."

6. Install Dependencies:

Once the project is open, Android Studio may prompt you to install missing dependencies. If prompted, click "Get dependencies" or "Install" to download and configure the required packages specified in the `pubspec.yaml` file.

7. Configure Emulator/Device:

Set up an Android emulator or connect a physical Android device to your computer. Ensure that the emulator/device is recognized by Android Studio.

8. Run the App:

In Android Studio, you can click the "Run" button (green triangle) to build and run the app on the emulator/device you configured in the previous step.

-Wait for Android Studio to compile the code and launch the app.

You should now have the Flutter mobile app project from the GitHub repository open in Android Studio and running on an emulator or device.

Remember to check the project's README or documentation for any specific setup or configuration instructions provided by the project owner, as additional steps may be required based on the project's dependencies and requirements.

2. Open as APK File

To open a Flutter mobile app project from a GitHub repository in Android Studio, you can follow these steps:

1. Install Git: Ensure that Git is installed on your computer. If it's not installed, you can download it from https://git-scm.com/downloads and follow the installation instructions for your operating system.

2. Clone the Repository:

- Open a terminal or command prompt.

- Navigate to the directory where you want to store the project.

- Run the following command to clone the GitHub repository:

" git clone https://github.com/bozkurttarda/Differeat1.git"

- This command will download the project files to your local machine.

3. Install Flutter and Dart:

- If you haven't already, make sure Flutter and Dart are installed on your system. You can download and set up Flutter from https://flutter.dev/docs/get-started/install.

4. Open Android Studio:

- Launch Android Studio

5. Open the Project:

- In Android Studio, click on "File" in the top menu.

- Select "Open" and navigate to the directory where you cloned the GitHub repository in step 2.

- Choose the root directory of the Flutter project (the one containing the `pubspec.yaml` file) and click "OK."

6. Install Dependencies:

Once the project is open, Android Studio may prompt you to install missing dependencies. If prompted, click "Get dependencies" or "Install" to download and configure the required packages specified in the `pubspec.yaml` file.

7. Configure Emulator/Device:

Set up an Android emulator or connect a physical Android device to your computer. Ensure that the emulator/device is recognized by Android Studio.

To open and build a Flutter mobile app from a GitHub repository and generate an APK file using Android Studio, follow these steps:

8. Build the APK:

- In Android Studio, select "Build" from the top menu.

- Choose "Build Bundle(s) / APK(s)" and then select "Build APK(s)".

9. Locate the APK:

- After the build process is complete, Android Studio will generate the APK file(s).

- You can find the APK file(s) in the project's `build/app/outputs/flutter-apk/` directory within your project folder.

10.Transfer the APK to an Android Device:

- If you built the APK for testing, you can transfer it to your Android device by connecting the device to your computer and copying the APK file to the device's storage.

11. Install and Run the APK:

- On your Android device, navigate to the location where you copied the APK file.

- Tap the APK file to initiate the installation process.

- Follow the on-screen instructions to complete the installation.

You should now have successfully cloned, built, and installed the Flutter mobile app from the GitHub repository as an APK file on your Android device. You can run and use the app on your device as you would with any other Android application.

4.Testing

The following section presents the comprehensive testing strategy and results for the Differeat Application, a culinary assistance software designed to provide users with personalized recipe recommendations based on their available ingredients and cuisine preferences. This testing initiative aims to ensure that the application operates seamlessly and reliably, in line with the specified use cases and functional requirements. Through a systematic approach to testing, this document evaluates the application's core functionalities, including user registration, login, ingredient input, recipe selection, and cuisine filtering. By rigorously testing these components, this document aims to validate the robustness and accuracy of the Differeat Application, providing insights into its performance and usability. The results of these tests will guide further refinement and enhancement of the application, ensuring an exceptional user experience. The tested version is a first prototype and is not aimed to be release.

4.1. Test cases

Test Case #1: Sign Up User (UC1)

- **Goal:** To test the functionality of the "Sign Up User" use case.
- **Environment:**
- **Driver:** Differeat application
- **Stub:** None
- **Oracle:** Validation of user registration in the application database
- **Input:** User enters valid information for registration (e.g., name, email, password)
- **Expected Output:** User account is successfully created and saved in the application database.
- **Actual Output:** This function is not implemented in the application.
- **Result:** Fail
- **Observations:** [Any additional observations or comments related to the test case]

Test Case #2: Login (UC2)

- **Goal:** To test the functionality of the "Login" use case.
- **Environment:**
- **Driver:** Differeat application
- **Stub:** None

- **Oracle:** Validation of user credentials during login
- **Input:** User enters valid credentials (username and password) for login
- **Expected Output:** User is successfully logged in and redirected to the session page.
- **Actual Output:** This function is not implemented in the application.
- **Result:** Fail
- **Observations:** [Any additional observations or comments related to the test case]

Test Case #3: Input Ingredients and Recipe Matching (UC3, R1 - S1, R2, R3, R4 - S2)

- **Goal:** To test the functionality of allowing users to select ingredients they have from a predefined list of ingredients in a database (R1) and matching the selected ingredients with recipes in a database (R2). The matching algorithm should prioritize recipes with a higher number of matching ingredients (R3) and consider the cuisine preference selected by the user, if any (R4).
- **Use Case:** Choose a Recipe (UC5)
- **Environment:**
- **Driver:** Differeat application
- **Stub:** Database containing a list of ingredients and matching recipes based on user-inputted ingredients and cuisine
- **Oracle:** Validation of user-submitted ingredients for matching recipes
- **Input:** User selects valid ingredients from the list provided by the application.
- **Expected Output:**
- User-submitted ingredients are successfully recorded and processed for recipe matching.
- User-selected recipe details (name, ingredients, cooking instructions, cooking time) are displayed on the screen.
- **Actual Output:** Testing by this output, code has been properly tested. Recipes compatible with the ingredients are presented to the user in the form of small cards. Later, when the user chooses what user wants from these cards, he can see the detailed recipe and how to make it.
- **Result:** Pass
- **Observations:** [Any additional observations or comments related to the test case]

Test Case #4: Cuisine Selection (UC4, R7, R8 - S4)

- **Goal:** To test the functionality of allowing users to choose a specific cuisine for the recipe recommendations (R7). The application should have a database of recipes for each cuisine available (R8).
- **Use Case:** Select a Cuisine (UC4)
- **Environment:**
- **Driver:** Differeat application
- **Stub:** Database containing recipes for different cuisines
- **Oracle:** Validation of user-selected cuisine and availability of recipes for the selected cuisine
- **Input:** User selects a valid cuisine from the available options.
- **Expected Output:** User-selected cuisine is successfully recorded and used for recipe filtering.

- **Actual Output:** Testing by this output, code has been properly tested. After the user has added his/him materials,user can choose the type of kitchen user wants. A single or more than one kitchen selection can be made. The recipes in the cuisine types that the user chooses are displayed.
- **Result:** Pass
- **Observations:** [Any additional observations or comments related to the test case]

Test Case #5: Choose a Recipe (UC5, R5, R6 - S3)

- **Goal:** To test the functionality of displaying the top 5 matching recipes to the user (R5). The application should display the recipe name, ingredients, cooking instructions, and cooking time for each recipe (R6).
- **Use Case:** Choose a Recipe (UC5)
- **Environment:**
- **Driver:** Differeat application
- **Stub:** Database containing matching recipes and their details
- **Oracle:** Validation of recipe details displayed to the user
- **Input:** User selects a valid recipe from the list of matching recipes.
- **Expected Output:** User-selected recipe details (name, ingredients, cooking instructions, cooking time) are displayed on the screen.
- **Actual Output:** Testing by this output, code has been properly tested. when the user chooses what he wants from these cards, user can see the detailed recipe and how to make it.
-
- **Result:** Pass
- **Observations:** [Any additional observations or comments related to the test case]

Test Case #6: Basic User Management (UC1, UC2, R9, R10 - S5)

- **Goal:** To test the functionality of allowing users to create a new account and login (R9). The application should also allow users to update their account information (e.g., name, email address, password) (R10).
- **Use Cases:** Sign Up User (UC1), Login (UC2)
- **Environment:**
- **Driver:** Differeat application
- **Stub:** None (for login) and Database containing user accounts (for sign up and update)
- **Oracle:** Validation of user account creation, login, and account information update
- **Input:**
- For Sign Up User (UC1): User enters valid information for registration (e.g., name, email, password)
- For Login (UC2): User enters valid credentials (username and password) for login
- **Expected Output:**
- For Sign Up User (UC1): User account is successfully created and saved in the application database.
- For Login (UC2): User is successfully logged in and redirected to the session page.

- **Actual Output:** This function is not implemented in the application.
- **Result:** Fail
- **Observations:** [Any additional observations or comments related to the test case]

4.2. Recapitulation table of testing

Test Case #	Test Case Description	Use Case (UC)	Requirement (R)	Result
1	Sign Up User (UC1)	UC1	R9	Fail
2	Login (UC2)	UC2	R10	Fail
3	Input Ingredients and Recipe Matching (UC3, R1 - S1, R2, R3, R4 - S2)	UC5	R1, R2, R3, R4	Pass
4	Cuisine Selection (UC4, R7, R8 - S4)	UC4	R7, R8	Pass
5	Choose a Recipe (UC5, R5, R6 - S3)	UC5	R5, R6	Pass
6	Basic User Management (UC1, UC2, R9, R10 - S5)	UC1, UC2	R9, R10	Fail

4.3. Conclusions on testing

The testing of the Differeat Application has yielded insightful results, shedding light on its functionality and performance. The table of test case results presents a mixed outcome, highlighting areas of success and potential areas for improvement.

The Sign-Up User and Login functionalities, encompassing Use Cases UC1 and UC2 respectively, were met with some challenges. These components, governed by Requirements R9 and R10, exhibited unexpected behaviour, leading to a "Fail" result. Further investigation is needed to rectify these issues and ensure a seamless user registration and login experience.

In contrast, the Input Ingredients and Recipe Matching feature, governed by Use Case UC5 and Requirements R1, R2, R3, and R4, demonstrated a positive outcome with a "Pass" result. This indicates that the application efficiently processes user-submitted ingredients and matches them with suitable recipes, considering both ingredient compatibility and cuisine preference.

The Cuisine Selection functionality, driven by Use Case UC4 and Requirements R7 and R8, also attained a "Pass" result. The application adeptly allowed users to select a specific cuisine, effectively filtering recipes based on culinary preferences.

Similarly, the Choose a Recipe feature, under UC5 and Requirements R5 and R6, achieved a "Pass" result, successfully displaying recipe details such as name, ingredients, cooking instructions, and cooking time.

Lastly, the Basic User Management aspect, encompassing Use Cases UC1 and UC2 along with Requirements R9 and R10, experienced challenges similar to the Sign-Up User and Login functionalities, resulting in a "Fail" outcome.

In conclusion, while the Differeat Application exhibited success in several key areas, particularly in ingredient input and matching, cuisine selection, and recipe display, there are

noteworthy concerns in user registration and login processes. These observations serve as valuable insights for further development and refinement of the application, with the aim of providing users with an enhanced and trouble-free culinary experience.

2. Conclusion

The "Differeat" project has been a remarkable journey in applying software engineering principles to design and develop a functional prototype that addresses a practical challenge. The project's primary objective, as outlined in the abstract, was to create a user-friendly software application that assists users in preparing meals with available ingredients, emphasizing simplicity, efficiency, and customization. In terms of results, the project demonstrated a successful implementation of core functionalities, including ingredient selection, recipe matching, cuisine selection, and recipe display, with these aspects performing admirably during testing. Nevertheless, some aspects of user management require further refinement.

From a software engineering perspective, this project allowed us to delve deep into the intricacies of Flutter and Dart, capitalizing on their strengths to deliver a cross-platform, performance-driven user interface with an object-oriented foundation. Through iterative development, we embraced the principle of software engineering, following all the steps of the V-cycle.

The learning experience throughout this project was invaluable. It not only enriched our technical skills but also instilled a sense of teamwork, software project management, and adaptability in response to evolving priorities. Moreover, the project demonstrated the importance of documenting every stage of a software development project.