**Feasibility study**

**Project abstract**

The proposed project is to develop a software application called "**Differeat**" that allows users to input the ingredients they have available and receive recommendations for dishes that can be made with those ingredients. The application will provide the top 5 matching dishes and allow the user to select a specific cuisine (such as Turkish, Chinese, or Italian) for the suggested recipes.

**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, Allrecipes 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.

**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, to allow users to save their favourite recipes, view their search history, and personalize their experience.

|  |  |
| --- | --- |
| **3.1 Overview** | *The overview presents a functional overview of the system.* |
|  | |
| **3.2 Functional requirements ("shall lists")**  **3.2.1**  **3.2.2**  **3.2.3**  **3.2.4** | *Functional requirements describe the high-level functionality of the system.* |
|  | |
| **3.3 Non-functional requirements**  **3.3.1 Usability**  **3.3.2 Reliability**  **3.3.3 Performance**  **3.3.4 Supportability**  **3.3.5 Implementation**  **3.3.6 Interface**  **3.3.7 Packaging**  **3.3.8 ~~Legal~~** | *Non-functional requirements describe user-level requirements that are not directly related to functionality. This includes usability, reliability, performance, supportability, implementation, interface, operational, packaging, and legal requirements.* |
|  | |
| **3.4 System models**  **3.4.1 Scenarios**  **3.4.2 Use case model**  **3.4.3 Analysis object model**  **3.4.4 Dynamic model**  **3.4.5 ~~User interface, navigational paths and screen mock-ups~~** | *System models describes the scenarios, use cases, object model, and dynamic models for the system. This section contains the complete functional specification, including mock-ups illustrating the user interface of the system and navigational paths representing the sequence of screens. The subsections Object model and Dynamic model are written during the Analysis activity.* |

From [Requirements Analysis Document Template (fsu.edu)](https://www.cs.fsu.edu/~lacher/courses/COP3331/rad.html)

*Req document could be organise like showed above, but how to gather that information? What are the steps and the process?*

[StarUML](https://staruml.io/download) to draw uml diagram

**Functional requirements:**

From the 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).
* R11 - *The application should allow users to save their favourite recipes to their account.*
* R12 - *The application should allow users to view their search history and delete it if desired.*

*ARE THE LAST TWO REQ RELEVANT ENOUGH OR NECESSARY?*

\* Sn for scope number n

\*\* Rm for requirement number m

**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 center 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.

**System models**

**Scenarios**:

A student accesses the application and looks for a recipe to cook by entering some ingredients he has. He specifies or not a type of cuisine he prefers.

**Use cases**

The User is the student. From the described scenario, 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 |

**Objects** with their attributes and the relationship between them

* Ingredient: Name, type, and quantity
* Recipe: Name, ingredients, and instructions
* Cuisine: Name and list of typical ingredients
* User: Input
* Application: Recipe name and relevant score (display)

**Classes** are same as objects ???

**The state** is the current configuration in which the object is and is defined by the values of its properties (attributes).

* Initializing: This is the initial state of the system, where it is not yet ready to receive user input.
* Idle Time: This state would represent the initial state of the system, where it is waiting for
* Searching for ingredients: The system is searching for available ingredients based on the user input.
* Displaying available ingredients: The system has retrieved the available ingredients and is displaying them to the user.
* Error: This state would represent a state in which an error has occurred, and the system is unable to generate recipe recommendations.
* Selecting preferred cuisine: The user is selecting a preferred cuisine.
* Generating recipe recommendations: The system is generating recipe recommendations based on the available ingredients and preferred cuisine.
* Displaying recipe recommendations: The system has generated the recipe recommendations and is displaying them to the user.
* Selecting a recipe: The user is selecting a recipe from the recommended options.
* Displaying recipe instructions: The system is displaying the recipe instructions for the selected recipe.
* Cooking: The user is following the recipe instructions and cooking the dish.
* Finished: The user has finished cooking the dish and the system is ready to receive new input.

**The behaviour** of the object determines how it acts and reacts

Ingredient object:

- addQuantity(quantity) - adds the specified quantity of the ingredient to the system

- removeQuantity(quantity) - removes the specified quantity of the ingredient from the system

- getQuantity() - returns the current quantity of the ingredient in the system

- getName() - returns the name of the ingredient

RecipeGenerator object:

- generateRecommendations(criteria) - generates a list of recommended recipes based on the specified criteria

- getRecipeList() - returns the list of recommended recipes

- getRecipeDetails(recipe) - returns additional information about the specified recipe

Recipe object:

- modifyRecipe(modification) - modifies the recipe according to the specified changes

- suggestAlternativeIngredients() - suggests alternative ingredients that can be used in the recipe

- getNutritionalInformation() - returns the nutritional information of the recipe

- getCookingInstructions() - returns the cooking instructions for the recipe

- getIngredients() - returns the list of ingredients needed for the recipe

- getName() - returns the name of the recipe

User object:

- saveRecipe(recipe) - saves the specified recipe to the user's recipe collection

- shareRecipe(recipe, recipients) - shares the specified recipe with the specified recipients

- searchRecipes(criteria) - searches for recipes that match the specified criteria

- generateShoppingList(recipes) - generates a shopping list for the specified recipes

- authenticateUser(username, password) - authenticates the user's login credentials

- updatePreferences(preferences) - updates the user's dietary preferences and restrictions

- getRecipeHistory() - returns the user's cooking history

- getPersonalizedRecommendations() - generates personalized recipe recommendations for the user

Database object:

- retrieveData(dataType) - retrieves the specified data from the database

- updateData(dataType, data) - updates the specified data in the database

- authenticateUser(username, password) - authenticates the user's login credentials

Notification object:

- setReminder(date, message) - sets a reminder for the specified date with the specified message

- sendNotification(recipients, message) - sends the specified message as a notification to the specified recipients

**An attribute** is a feature of the class, and every attribute must be precisely defined

For the Ingredient class:

- name: the name of the ingredient

- category: the category or type of ingredient

- quantity: the current quantity of the ingredient in the system

- unit: the unit of measurement for the ingredient

For the Recipe class:

- name: the name of the recipe

- cuisine: the cuisine or type of food

- difficulty: the difficulty level of the recipe

- ingredients: a list of the required ingredients and their quantities for the recipe

- instructions: a list of steps to follow to prepare the recipe

- rating: the average rating of the recipe based on user ratings

For the User class:

- username: the user's username or login ID

- password: the user's password

- email: the user's email address

- preferences: the user's dietary preferences or restrictions

- favorite\_recipes: a list of the user's favourite recipes

- shopping\_list: a list of the ingredients the user needs to buy for their selected recipes

**An operation** is a transformation that can be applied to an instance of a class

Same as behaviours???

1. Ingredient (superclass)

* Vegetable (subclass)
* Fruit (subclass)
* Meat (subclass)
* Dairy (subclass)
* Grain (subclass)
* Spice (subclass)
* Seafood (subclass)

1. Recipe (superclass)

* Breakfast Recipe (subclass)
* Appetizer Recipe (subclass)
* Lunch Recipe (subclass)
* Dinner Recipe (subclass)
* Dessert Recipe (subclass)
* Vegetarian (subclass)

1. User (superclass)

* Registered User (subclass)
* Premium User (subclass)
* Guest User (subclass)