

# **T****able of Contents**

[**Table of Contents** 2](#_Toc168348965)

[**1.** **Executive Summary** 4](#_Toc168348966)

[**2.** **Introduction** 5](#_Toc168348967)

[**2.1.** **Problem Statement / Project Vision** 5](#_Toc168348968)

[**2.2.** **System Capabilities** 5](#_Toc168348969)

[**2.3.** **Non-functional Requirements and Design Constraints** 7](#_Toc168348970)

[**2.4.** **System Evolution** 7](#_Toc168348971)

[**2.5.** **Document Outline** 9](#_Toc168348972)

[**3.** **Structural Model** 10](#_Toc168348973)

[**3.1.** **Model Introduction** 10](#_Toc168348974)

[**3.2.** **Class Diagrams** 10](#_Toc168348975)

[**3.3.** **Metadata** 12](#_Toc168348976)

[**4.** **Architecture Design** 34](#_Toc168348977)

[**4.1.** **Architecture Overview** 34](#_Toc168348978)

[**4.2.** **Infrastructure Model** 34](#_Toc168348979)

[**4.2.1.** **Deployment Diagram 1 – Architecture Overview** 34](#_Toc168348980)

[**4.2.2.** **Deployment Diagram 2 – Nodes and Artifacts** 35](#_Toc168348981)

[**4.3.** **Hardware and Software Requirements** 35](#_Toc168348982)

[**4.3.1.** **Hardware Components** 35](#_Toc168348983)

[**4.3.2.** **Required Software Components** 36](#_Toc168348984)

[**4.4.** **Security Plan** 36](#_Toc168348985)

[**4.4.1.** **Security Overview** 36](#_Toc168348986)

[**4.4.2.** **Security Plan** 36](#_Toc168348987)

[**5.** **User-Interface** 38](#_Toc168348988)

[**5.1.** **User-Interface Requirements and Constraints** 38](#_Toc168348989)

[**5.2.** **Window/Screen Navigation Diagram** 39](#_Toc168348992)

[**5.3.** **UI Wireframes** 41](#_Toc168348993)

[**5.4.** **Reports: “Formal Output” Design** 50](#_Toc168348994)

[**6.** **Appendices** 50](#_Toc168348995)

[**6.1.** **Glossary** 50](#_Toc168348996)

[**6.2.** **References / Bibliography** 51](#_Toc168348997)

[**6.3.** **Supporting documentation** 52](#_Toc168348998)

# **Executive Summary**

This document outlines the complete implementation of the first and future versions of the cooking optimization application. The cooking optimization application is a recipe-reading application with tertiary features, allowing users to improve and note recipe changes. It is best thought of as a creative source control for cooking dishes.

This document’s target audience is the developers responsible for developing, improving, and maintaining the cooking optimization application. Familiarity within the team with concepts like cross-platform application development, client-server architecture, and MVC architecture is ideal. The document is designed to minimize confusion and clarify the steps necessary to implement the system as described.

The Customers for this application comprise two primary groups: individual chefs and chain restaurants. Individual chefs’ requirements go hand-in-hand with chain restaurants, as they desire an easy-to-use interface that is easy to navigate, displays recipe steps, and allows them to make notes and collect statistics about their cooking. Executive customers are more complex, as they require a top-level view of each chef’s statistics and variations to improve their dishes. The cooking optimization application should suffice for both of these groups’ needs.

So far, the only supporting work done on this project is a system proposal document. In sections #1-3, the system proposal document sets the initial vision and scope, lists the requirements for operating the application, and assesses the overall feasibility of this project. Section #4 of the system proposal also reviews several requirements for creating this application. Finally, section #5 displays a use case diagram and several individual use cases.

The goal of this document is to completely and thoroughly outline the several features, capabilities, and interfaces that this application requires. Each section should make clear the development and structure of parts of the application. The first section includes an introduction to the system and outlines the high-level features that should be available to users. The following section is a diagram and model of the basic classes and systems necessary to create the minimum viable product (MVP). Afterward, the document displays the proposed details of the architecture structure and several user interface (UI) wireframes. Finally, there is an appendix for more details and information.

# **Introduction**

The purpose of the cooking optimization application is to allow chefs to read, change, and improve recipes, allowing them to make informed decisions about their flavor choices and ingredients. This application acts as a sort of log and guideline for cooking improvements. The goal is to enable users to walk through recipes, add notes, and create variations of recipes. As a result, the optimizer must store multiple recipes, link them with their respective notes, variations, and ingredients, and allow users to add notes, statistics, and changes on the fly for future improvements. In doing so, this application builds a positive feedback loop with cooking, where a chef prepares a recipe, takes notes, and sees trend lines of taste and flavor over long periods, making minor adjustments for improvement and optimization.

## **Problem Statement / Project Vision**

The Cooking optimization application presents itself as a convenient way to view, note, and optimize recipes. The application overcomes the typical limitations of note-taking in a kitchen environment. Using paper to take notes in kitchens is challenging, as liquids, binding agents, and flour dust are usually prevalent, ruining the paper and making it difficult to write. Alongside this, there’s no standard “notes” section on recipe paper, forcing the chef to either write between the lines on the cooking documentation or attach an entirely separate piece of paper referencing the original recipe. Both solutions are a hassle compared to a mobile application. This application aims to solve all these problems, allowing chefs to track their cooking data and change their recipes.

This application’s design prioritizes recipe viewing, data collecting, and optimization of recipes on an individualized level. The design allows individual chefs to walk through recipe instructions while gathering statistics. As a result, individuals should be able to record data and changes on a small scale, and then place changes in a database for later review.

This application should appeal to two product markets: First, it appeals to individual chefs, household cooks, and small businesses with a desire for more scientific methods of cooking. If individual chefs can store and note recipes for free with an easy importing feature, then permanent adoption is much more likely. The other target market is chain restaurant executives who desire to improve their culinary capabilities. If a chain can use data from every one of their independent chefs, send recipe updates quickly, and track individual recipe preparations, then benefits and adoption are assured. Other stakeholders to consider when creating the application include the end consumers, the FDA, and app stores for hosting our application.

## **System Capabilities**

The following list of capabilities should be available for the cooking optimization application by the second version. The list serves as a roadmap for functionality and what functions to consider when building the initial framework. For the first version of the application, enterprise customers should be a lower priority than the main recipe functionality.

1. (Select the recipe for preparation – Use Case #1 – Functional requirement 1.1)
   * A chef must be able to select a recipe from a list to prepare.
2. (Prepare recipe – Use Case #2 – Functional requirement 1.1)
   * A chef should be able to get a step-by-step walkthrough of any recipe in their local database.
3. (Record recipe anomalies and changes – Use Case #3 – Functional requirement 1.3, 1.4, 3.2)
   * When a chef makes a mistake or experiments, they should be able to note it in the application for later reference.
4. (Add recipe data to local database – Use Case #4 – Functional Requirement 1.5, 1.6, 3.1)
   * After a recipe is complete, the chef will want to write the cooking outcome to add it to their overall statistics.
5. (Import Recipe – Use Case #6 – Functional requirement 1.2, 3.1)
   * If a chef or executive finds a link or cookbook with a recipe they want inside the application, they should be able to import it with minimal assistance.
6. (Create a recipe – Use Case #7 – Functional requirement 2.3, 3.1)
   * If a chef or executive cannot import the recipe, or wants to create a new one, they should be able to input the recipe manually.
7. (Update recipe – Use Case #9 – Functional requirement 3.2)
   * Chefs and executives must be able to change or rework a recipe.
8. (Purchase company database access – Use Case #11)
   * A chain executive can decide to purchase our internet database service.
9. (Log in to the chain network – Use Case #12 – Functional requirement 5.1, 5.2, 5.3)
   * Executives and chefs need to confirm their identity with the Internet database service.
10. (Register chef as an employee of a restaurant – Use Case #13)
    * A chain chef can request to be recognized as a chef of a chain restaurant, showing their statistics to their executive by entering the employee key given to the executive.
11. (Visualize chain cooking data – Use Case #14 – Functional requirement 4.1)
    * An executive accesses the internet-enabled database to show full performance metrics for their location.
12. (Store cooking information in the company database – Use Case #15)
    * When a chain chef finishes preparing a meal, the statistics they generate should synchronize with the central database.
13. (Perform database maintenance – Use Case #16)
    * When the application requires maintenance, the system requires a standard protocol of informing companies, doing a controlled shutdown, backing up data, performing maintenance, and bringing the system back up again.
14. (Conduct security inspection – Use Case #17)
    * Our monitoring teams need standards for conducting security inspections on our internet database.
15. (Register as a chef of a chain or business – Use Case #18)
    * If a chef decides to register as part of a chain or business, they can enter their employee key to send an employee request to the executive.
16. (Create a timer for cooking – Use Case #19 – Functional Requirement 2.1)
    * If, while doing a recipe, there is a requirement for a timer or time amount, the application should have built-in functionality that does not require external support.
17. (Visualize individual cooking data – Use Case #20)
    * Individual chefs need to view the recipe cooking data that they have aggregated.
18. (Measurement conversion and substitutions – Functional requirement 2.2)
    * When walking through a recipe, a helpful feature is having substitutions and recipe measurement conversion on hand in case the chef doesn’t have every measurement or ingredient available.

## **Non-functional Requirements and Design Constraints**

The following list is the non-functional requirements (Requirements that are not functions of the application) and constraints of the application. This section should help to clarify some of the overarching goals and testing that should occur during development. For more details on this section, please refer to the system proposal’s non-functional requirements and the constraints section.

* + 1. **Non-functional Requirements**
       1. Input
          - Input errors should be uncommon for most users.
          - Inputting information needs to be simple and not time-consuming.
       2. Installation, Maintenance, and Data Management
          - The software must be portable, lightweight, and easy to install regardless of the platform.
          - Our responsibilities for the data we receive are system maintenance, backups, and issues.
          - Maintenance should only happen at low-traffic times, and customers should be notified beforehand.
          - The installation process needs to be simple enough that Non-tech-savvy individuals can install our application without help.
          - The application must be available on mobile and desktop devices.
       3. Administration
          - Purchasers of our application need to view their aggregated statistics.
          - Purchasers need to view and manage employee data.
       4. Support
          - Documentation for application usage is necessary for end users.
          - An optional tutorial would be beneficial to teach end-users about the application.
    2. **Constraints**
       1. There is an extreme time limit inherent within the food service industry. For this reason, we cannot apply this application to many fast-paced food service businesses.
       2. For localization and legal reasons, the application’s primary audience is the United States.
       3. Mass data collection is off-limits when building this application on moral and technical grounds.

## **System Evolution**

The first version of the cooking optimization application should focus on the core features of recipe display, editing, and optimization. While database architectures, paying customers, and connections to the food service industry are excellent to remember when designing, a solid initial feature set and optimized design will help more for long-term adoption than an untested version moving straight into a corporate marketplace. The initial version of the application should implement the functionality needed to modify and optimize recipes individually. However, this document section details the system’s evolution after a solid support base is available.

* + 1. **Version 2 Changes**

The second version of the application should primarily focus on the quality-of-life improvements and the expansion of systems that had the groundwork laid in Version 1. These include integrated timers, expansion of the variation system, Stopwatches, and stop/start times for working. The company-owned database, as described, should be enacted during this version.

1. Convert recipe measurements.
   * The system must have a standardized recipe and measurement chart of multiple measurements, conversions, and formulas to match it.
   * The application should allow users to convert figures and statistics into a preferred format.
2. Produce recipe adjustment for a circumstance.
   * Given a selected circumstance, it should be possible to change a recipe for another purpose, for example, allergies or changes in pressure.
   * The application should then prompt the user with given circumstances to adjust for, such as lack of ingredients, removing certain ingredients for allergies, increasing a particular property, etc.
   * To automatically adjust a recipe for a circumstance, the application needs to scan the recipe for commonality in recipe-making steps, then find points where modification can happen (i.e.), replacing eggs in a recipe.
   * The application should pull from a dedicated database of common modifications, display the change for the user along with its intended side effects, and ask them if they want to change their recipe in this way.
     1. **Version 3 and beyond Changes**

The third version of the application will focus more on user interaction and collaboration, allowing users to share and collaborate on recipes via the Internet. The following are some primary use cases to support such functionality.

1. Share recipe
   * The application should allow a user to share an entire recipe with another individual via social media, email, or other contact.
   * The recipe could be an embedded link or a dedicated webpage so that others with the link can download and modify it.
   * Within the link, there should be an option to download or view the recipe in our application. The link should be subtle, but tempting enough for an interested customer to consider using the application.
2. Troubleshooting information screen
   * A standard troubleshooting or recipe-rescuing feature should be available within a recipe.
   * When presenting the steps of a recipe, the additional options should include a “Rescue at this step button.”
   * The rescue button should analyze the recipe step for mentions of specific ingredients and common problems with them.
   * From this, the application should be able to generate some basic recommendations or steps to get a recipe back on track (i.e., adding more flour, baking soda, salt, etc.)
3. Create a recipe central repository.
   * The system should have a group of dedicated servers hosted for recipe sharing, or the system could implement an extensive Peer-To-Peer network, allowing users to share recipes and details.
   * This central repository should act like software source repositories, with corrections, updates, old versions, revisions, and variations on the recipes at hand.
   * Statistics for recipes, both for users and the original author of the recipe, should be stored within the central repository. The code should also exclude extreme statistical outliers from the count.
   * Popular recipes and those with plenty of statistics should be available for download so the superior recipe can be available to as many people as possible.

## **Document Outline**

The rest of this document details how to implement the individual facets of structure, architecture, and user interface design.

* The structural modeling section includes the class diagram and proposed class descriptions. It also details the procedures and data of the application, along with some functions and variable definitions.
* The architecture design section details the system architecture, components, and plans necessary for the physical implementation and management of the application, along with detailing a plan for its physical security.
* The user interface section gives a detailed overview of the user interface requirements and navigation. It also includes several wireframes that help visualize the application.

# **Structural Model**

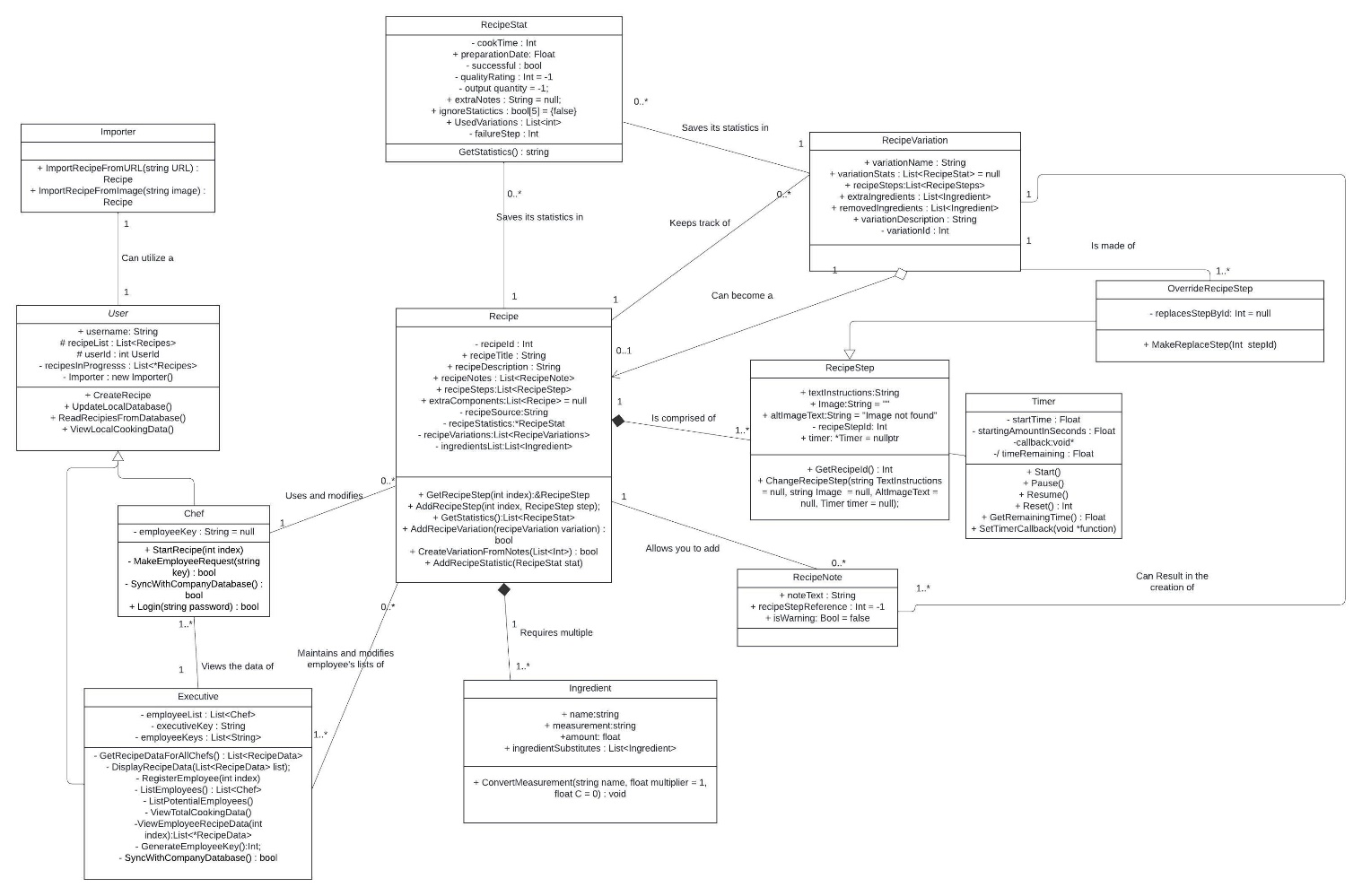
## **Model Introduction**

The following section shows the structure of the application of cooking optimizers. It uses the Unified Modelling Language (UML) to display a broad representation of the physical classes, relationships, attributes, and functions available to each part of the system. The metadata section describes each class’s functionality in much more detail and is critical to view when building the application.

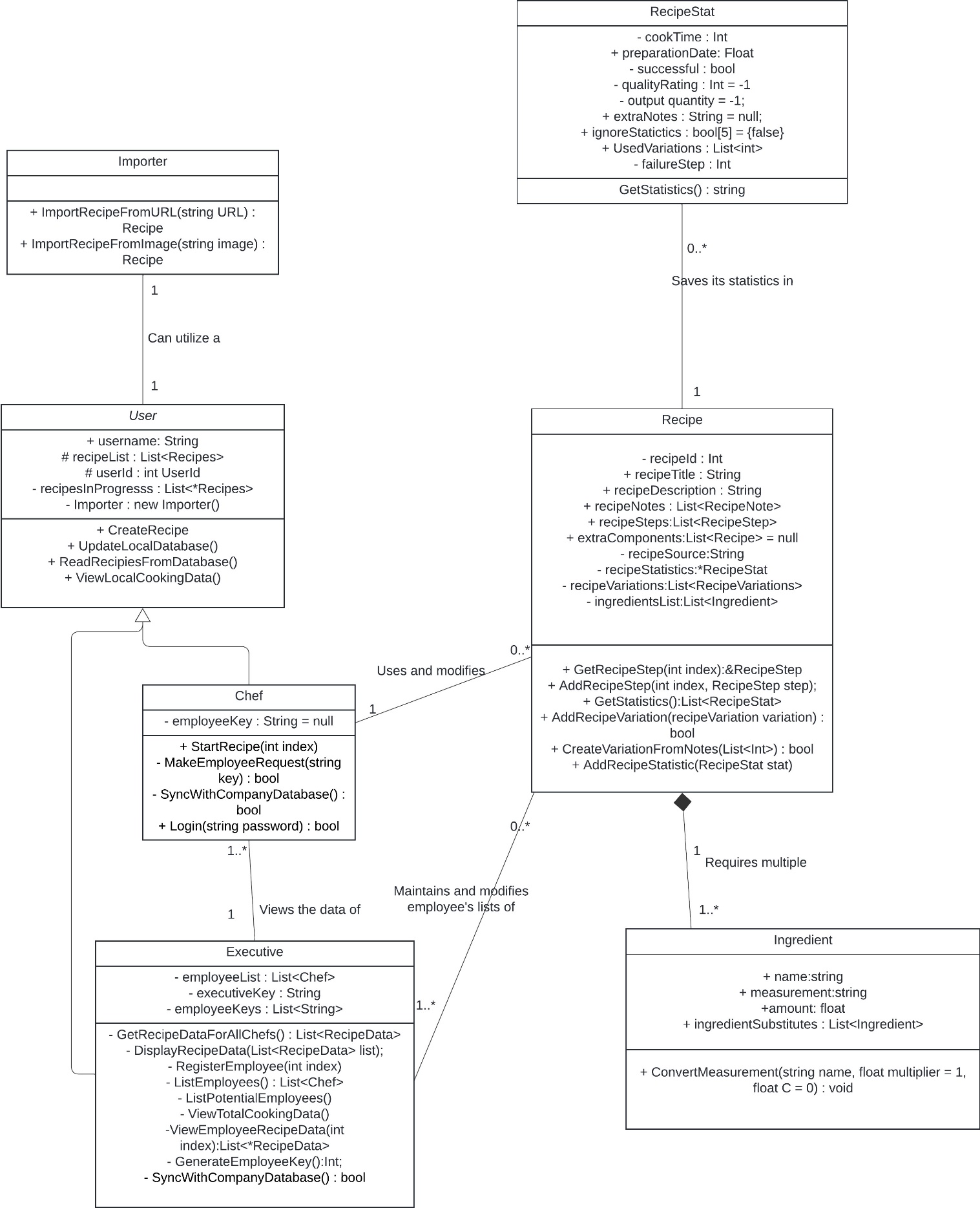
## **Class Diagrams**

This section relays the class diagram for the Cooking Optimization application. The internet-enabled database and some related functionalities are not displayed.

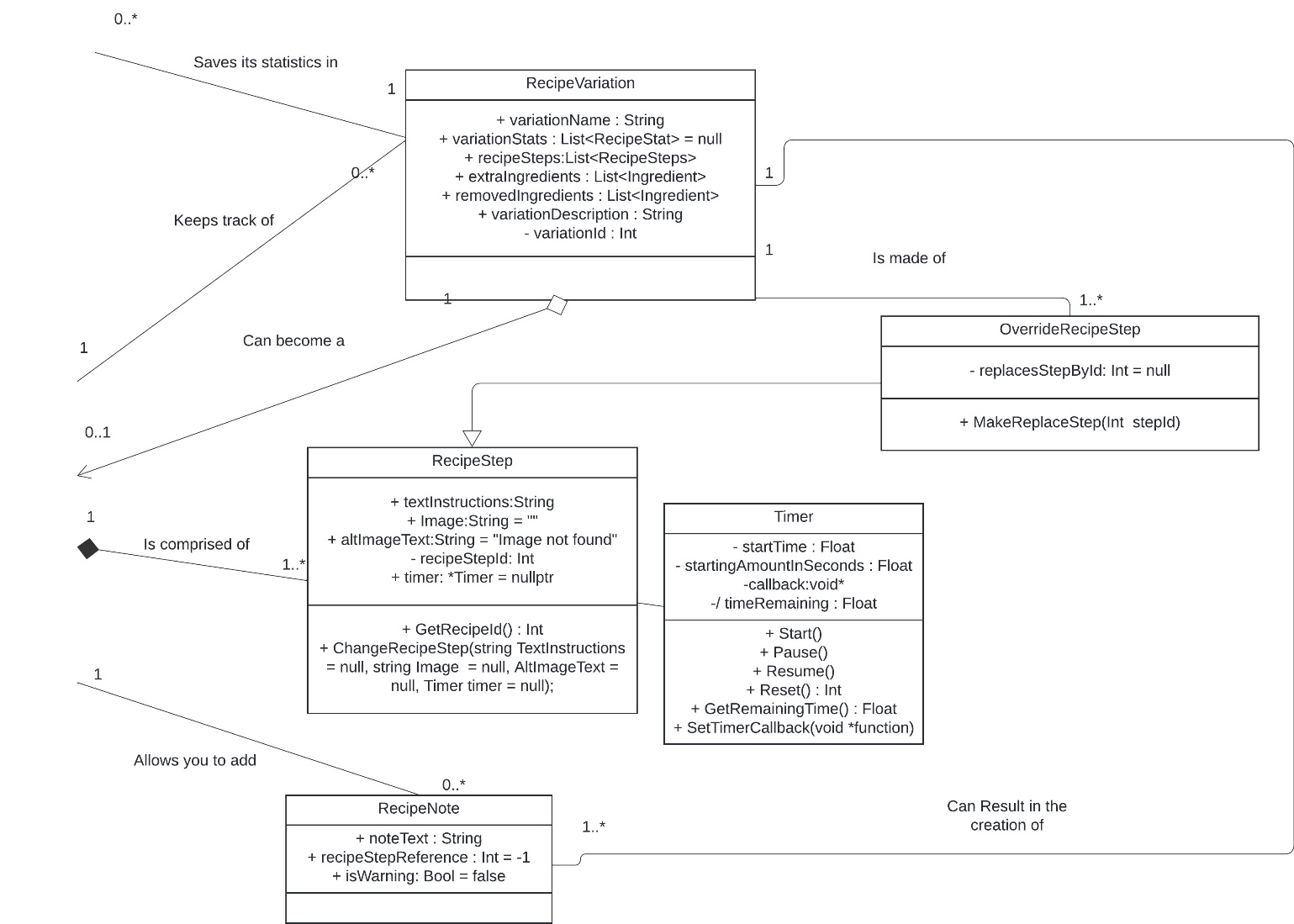
Combined:



The combined class diagram



Top-Level user view



Recipe parts view

## **Metadata**

Table of Contents

[Chef 14](#_Toc168349001)

[Executive 16](#_Toc168349002)

[Importer 19](#_Toc168349003)

[Ingredient 20](#_Toc168349004)

[OverrideRecipeStep 21](#_Toc168349005)

[Recipe 22](#_Toc168349006)

[RecipeNote 25](#_Toc168349007)

[RecipeStat 26](#_Toc168349008)

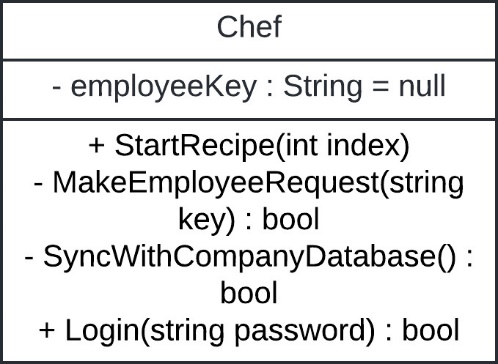
[RecipeStep 28](#_Toc168349009)

[RecipeVariation 29](#_Toc168349010)

[Timer 30](#_Toc168349011)

[User 32](#_Toc168349012)

1. Chef



Description: A class representing an individual or hired chef. This class inherits from the User class.

Visibility: Public

Is Abstract: No

Additional information:

**Attributes**

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Description | Read only? | Multiplicity |
| employeeKey | The optional key that registers a chef as part of a restaurant chain. | No | 1 |

**Operations**

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Description | Is query? | Is Polymorphic? |
| StartRecipe(int index) | Begin creating a recipe. Shows step-by-step instructions to the user and allows them to enter notes and start timers. | No | No |
| MakeEmployeeRequest | Request to join a chain network by entering the employee key. | No | No |
| SyncWithCompanyDatabase | Synchronize the local database with the company database. | No | No |
| Login | Log in to the employee network. | No | No |

**Processing Outlines ->**

Void StartRecipe(int index);

The function selects a recipe from the inherited recipeList for preparation. Fetches the recipe steps via GetRecipeStep and walks the user through them.

From here, the entire recipe steps flow starts, and the application runs through each recipe step. At the end, statistics are collected and tied to the recipe. Upon completion, the recipe data is stored in the local database for later use.

MakeEmployeeRequest;

The employee, if having a key, makes a request to the backend server to be part of a chain restaurant organization.

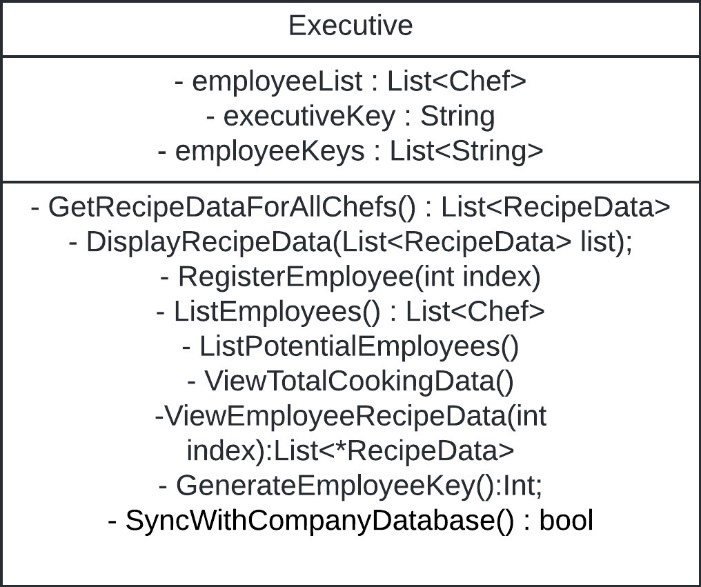
SyncWithCompanyDatabase

If the chef is part of a chain restaurant organization, they can sync with the company database at any time by pressing this button. This process happens via web requests to the purchased company server.

Login

The employee sends their key, username, and password hash to the server owned by the company, requesting authorization to sync with the company database.

1. Executive



Description: Executive class, inherits from User.

Visibility: Public

Is Abstract: No

Additional information:

**Attributes**

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Description | Read only? | Multiplicity |
| employeeList | A list of employees at the | No | 1 |
| executiveKey | The key for the executive user. | No | 1 |
| employeeKeys | The list of employee keys currently enrolled. | No | 1 |

**Operations**

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Description | Is query? | Is Polymorphic? |
| List<RecipeData> GetRecipeDataForAllChefs | Get the recipe data of all chefs at the chain | Yes | No |
| DisplayRecipeData | Display all the recipe data for chefs at the chain. | No | No |
| RegisterEmployee | Of the potential employees, send a request to the backend to add the one at the index to the Current Employees list. | No | No |
| List<Chef> ListEmployees | List the current employees at the chain | Yes | No |
| ViewTotalCookingData | View the cooking data available for the chain chefs. | No | No |
| ViewEmployeeRecipeData | View the recipe data of an individual chef. | No | No |
| GenerateEmployeeKey | Generate a key for an employee. | No | No |
| Bool SyncWithCompanyDatabase | Synchronize the local database with the company database. | No | No |

**Processing Outlines ->**

List<RecipeData> GetRecipeDataForAllChefs

Queries the private list “employeeList”, for each recipe in the list, queries “GetStatistics,” and tabulates it into a single list of recipe data.

DisplayRecipeData

Visually displays the recipe data for all the chefs in the list for the executive. Calls GetRecipeDataForAllChefs

RegisterEmployee(int index)

Register an employee from the “ListPotentialEmployees” list that corresponds with the index. Calls ListPotentialEmployees, adds the one at the index to the “employeeList” data member.

List<Chef> ListEmployees

Getter method for the employeeList data member. The function returns a list of chefs.

ViewTotalCookingData

Viewing method for the recipe data available to the executive. Calls GetRecipeDataForAllChefs, and displays it however specified.

ViewEmployeeRecipeData(int index)

View an individual employee’s recipe data by calling this method. Accesses employeeList at the index, then displays the recipeData of that chef as specified.

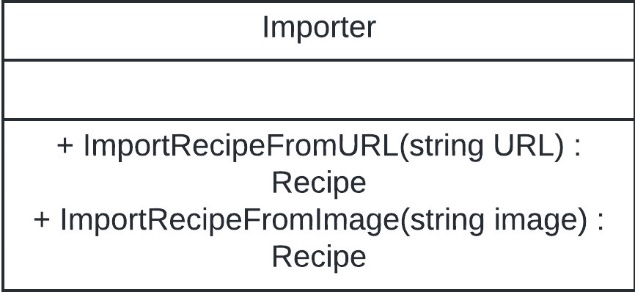
Int GenerateEmployeeKey

The function generates an employee key and adds it to the local and company database. The function then returns the key’s value.

Bool SyncWithCompanyDatabase()

Synchronizes the local database with the company database. Does all actions related to connecting, sending requests, and disconnecting. Assumes precedence over the current employee list, and other relevant data members.

Importer



Description: A class designated for importing recipe data from external sources.

Visibility: Public

Is Abstract: No

Additional information:

**Attributes**

N/A

**Operations**

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Description | Is query? | Is Polymorphic? |
| ImportRecipeFromURL | Import a recipe from a web URL | No | No |
| ImportRecipeFromImage | Imports a recipe from an image. The function will probably send image data to a web URL for processing. | No | No |

**Processing Outlines ->**

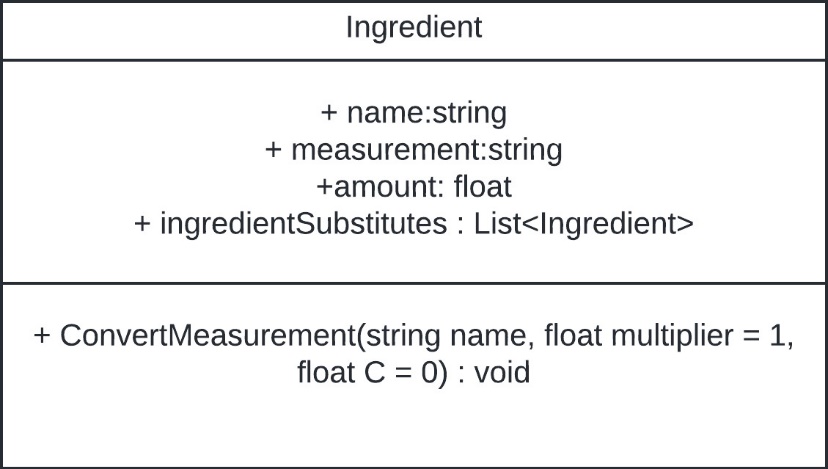
Recipe ImportRecipeFromUrl(string URL)

Fetches the given URL. Searches for, in the visible mode of the web browser, terms such as “Ingredients,” “Instructions,” “Steps,” and other key terms. Scrapes the recipe from the returned web data and returns it in our “Recipe” format.

Recipe ImportRecipeFromImage(string Image)

Given a base64 representation of an image string, send the data to an external service for image processing. The image processing service should identify the raw text, which is then used by helper functions of ImportRecipeFromUrl to convert to a recipe format.

1. Ingredient



Description: An ingredient for a recipe.

Visibility: Public

Is Abstract: No

Additional information:

**Attributes**

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Description | Read only? | Multiplicity |
| Name | The ingredient’s name | No | 1 |
| Measurement | The measurement name to compare to. | No | 1 |
| Ingredient | The ingredient amount. | No | 1 |
| ingredientSubstitutes | A list of potential substitutes to the ingredient. | No | 1 |

**Operations**

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Description | Is query? | Is Polymorphic? |
| ConvertMeasurement | Convert the recipe’s measurements to another value. | No | No |

**Processing Outlines ->**

Void ConvertMeasurement(string name, float multiplier = 1, float C = 0)

Changes the measurement of a recipe via a set conversion multiplier and C. Formula is as follows: amount = (amount \* multiplier) + C, a simplistic way to convert various measurements.

1. OverrideRecipeStep



Description: A step override used for recipe variations to note which steps should be changed. The class inherits from RecipeStep.

Visibility: Public

Is Abstract: No

Additional information:

**Attributes**

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Description | Read only? | Multiplicity |
| replacesStepById | The step that this recipe will replace. | No | 1 |

**Operations**

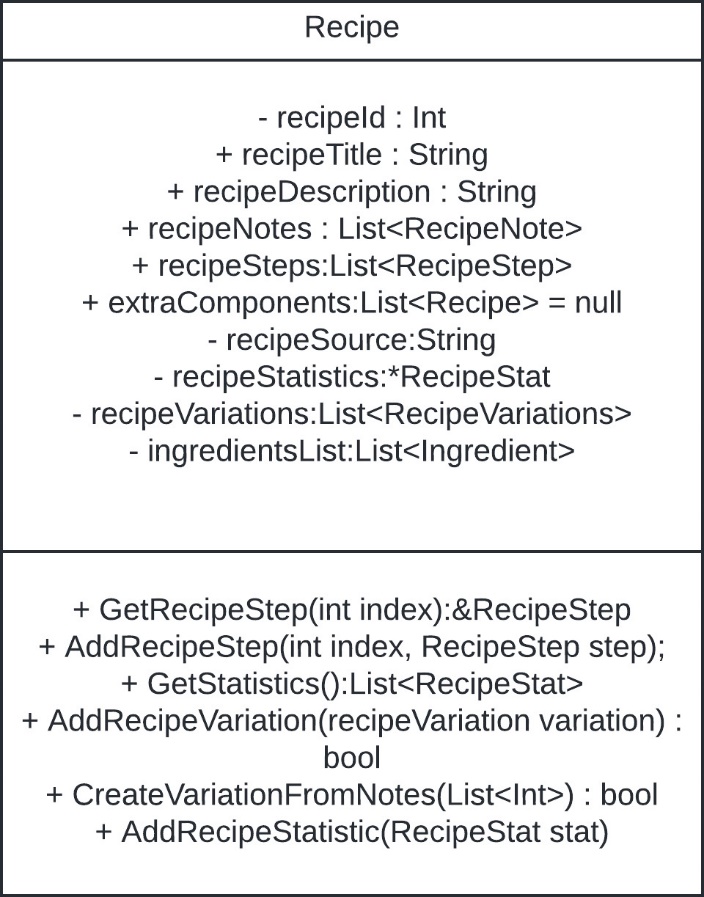
|  |  |  |  |
| --- | --- | --- | --- |
| Name | Description | Is query? | Is Polymorphic? |
| MakeReplaceStep(int stepId) | Mutator for replacesStepById | No | No |

**Processing Outlines ->**

MakeReplaceStep(int stepId)

Mutator for replacesStepById. Sets the step that this class will replace.

1. Recipe



Description: A representation of a complete recipe. Includes a title, notes, steps, ingredients, tertiary recipes, the source, the statistics, and variations.

Visibility: Public

Is Abstract: No

Additional information:

**Attributes**

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Description | Read only? | Multiplicity |
| recipeId | The identifying number of the recipe. | Yes | 1 |
| recipeTitle | The recipe’s title. | No | 1 |
| recipeDescription | The description of the recipe. | No | 1 |
| recipeNotes | A list of notes that are added to the recipes. | No | 1 |
| recipeSteps | The instructions needed to prepare the recipe. | No | 1 |
| extraComponents | Component recipes that might be part of this recipe: I.E frosting, sauces, spreads. | No | 1 |
| recipeSource | Initialized on construction. The recipe’s original source or cookbook. | Yes | 1 |
| recipeStatistics | Internal statistic tracker. A list of recipeStatistics. | No | 1 |
| recipeVariations | A list of variations for the recipe. | No | 1 |
| ingredientsList | The list of ingredients necessary for the recipe. | No | 1 |

**Operations**

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Description | Is query? | Is Polymorphic? |
| GetRecipeStep(int index) | Get a recipe step from the list. | Yes | No |
| AddRecipeStep | Add a step to the recipe. | No | No |
| GetStatistics | Get the statistics for the recipe. | Yes | No |
| AddRecipeVariation | Add a variation to the given recipe. Adds a recipe variation to the recipeVariationsList | No | No |
| AddRecipeStatistic | Add a statistic to the recipe’s list of statistics. | No | No |

**Processing Outlines ->**

&RecipeStat GetRecipeStep(int index)

Returns the recipe step at the given index from recipeSteps.

AddRecipeStep(int index, RecipeStep step)

Adds a recipe step to the recipeSteps list at the given index.

List<RecipeStat> GetStatistics

Returns a list of recipe statistics to the

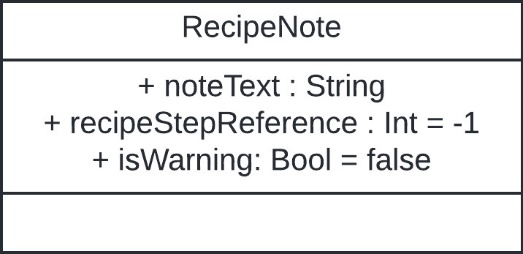
Bool AddRecipeVariation(recipeVariation variation)

Add a recipeVariation to the recipeVariations list.

AddRecipeStatistic(RecipeStat stat)

Add a recipe statistic to the recipeStatistics list.

1. RecipeNote



Description: A note added to a recipe.

Visibility: Public

Is Abstract: No

Additional information:

**Attributes**

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Description | Read only? | Multiplicity |
| noteText | The text for the note. | No | No |
| recipeStepReference | The step of the recipe to apply this note to. | No | No |
| isWarning | Boolean that represents if the note is a warning for common mistakes. | No | No |

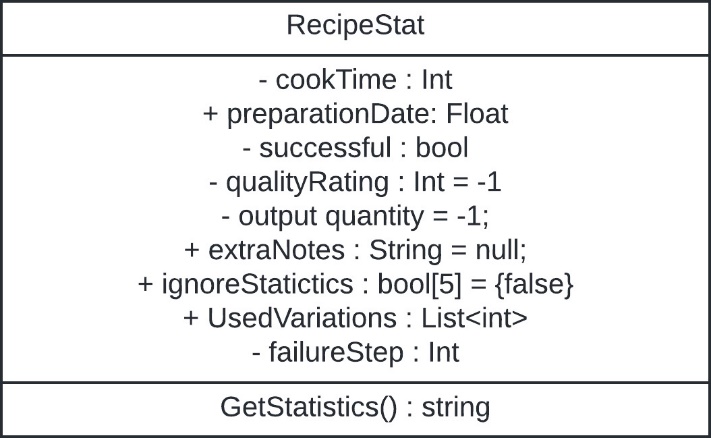
**Operations**

N/A

**Processing Outlines ->**

N/A

1. RecipeStat



Description: A collected statistic from a recipe.

Visibility: Public

Is Abstract: No

Additional information:

**Attributes**

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Description | Read only? | Multiplicity |
| cookTime | The time it took for the recipe to be prepared. | No | 1 |
| preparationDate | The date, in epoch, on which the recipe was prepared. | No | 1 |
| Successful | A Boolean representing if the attempt was successful. | No | 1 |
| qualityRating | A rating out of five representing the quality of the recipe. | No | 1 |
| outputQuantity | The amount, if applicable, of units produced. | No | 1 |
| extraNotes | Notes for statistics that might be outliers. | No | 1 |
| ignoreStatistics | An array of Booleans representing the statistics that should not be counted in official tallies. | No | 1 |
| failureStep | The step in which the user failed the recipe, if applicable. | No | 1 |

**Operations**

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Description | Is query? | Is Polymorphic? |
| GetStatistics | Get a string representation of the stored statistics | No | No |

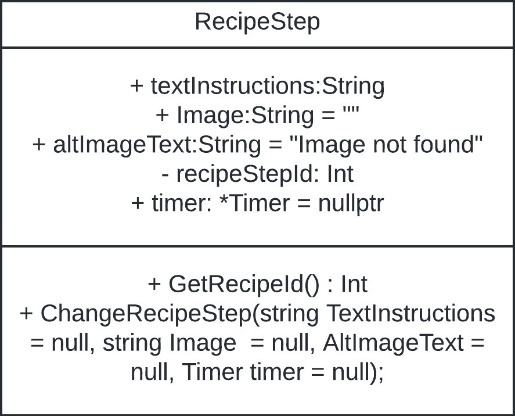
**Processing Outlines ->**

String GetStatistics()

Getter method for RecipeStat

Returns the values in a string JSON format, as the class properties should not be changed.

1. RecipeStep



Description: An individual step of a recipe.

Visibility: Public

Is Abstract: No

Additional information:

**Attributes**

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Description | Read only? | Multiplicity |
| textInstructions | The string representing the text instructions for the recipe step | No | No |
| Image | The base64 string representation of the recipe image. | No | No |
| altImageText | The alternate image text supplied if the image cannot load. | No | No |
| recipeStepId | The ID number for this recipe step. | Yes | No |
| Timer | A point to an optional timer attached to this recipe step | No | No |

**Operations**

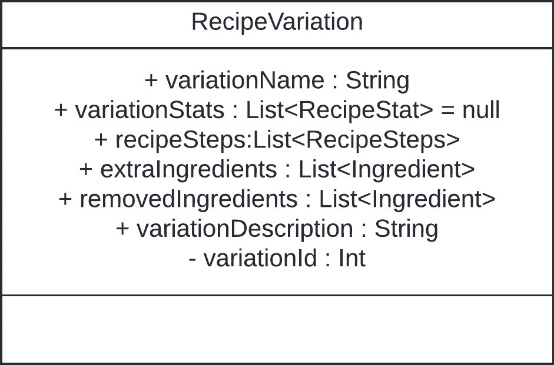
|  |  |  |  |
| --- | --- | --- | --- |
| Name | Description | Is query? | Is Polymorphic? |
| GetRecipeId | Accessor method for RecipeId | Yes | No |

**Processing Outlines ->**

GetRecipeId()

Returns a deep copy of RecipeId.

1. RecipeVariation



Description: A variation of the recipe, stored as steps and ingredients.

Visibility: Public

Is Abstract: No

Additional information:

**Attributes**

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Description | Read only? | Multiplicity |
| variationName | The name of the recipe variation. | No | 1 |
| variationStats | The list of recipe stats attached to this variation. | No | 1 |
| extraIngredients | A list of additional ingredients that should be available for the recipe variation. | No | 1 |
| removedIngredients | A list of ingredients which should be removed from the recipe when using this variation. | No | 1 |
| Notes | A string describing the recipe’s notes. | No | 1 |

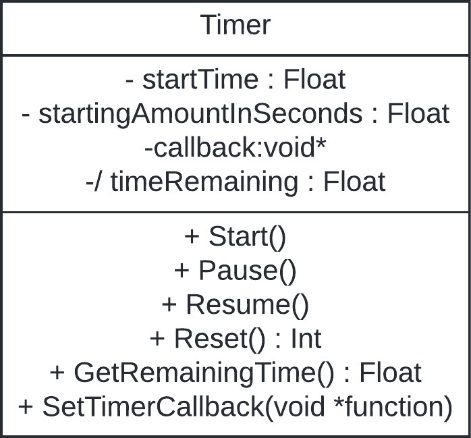
**Operations**

N/A

**Processing Outlines ->**

N/A

1. Timer



Description: A timer class for the recipe step. It may be possible to remove this class for a standard library function.

Visibility: Public

Is Abstract: No

Additional information:

**Attributes**

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Description | Read only? | Multiplicity |
| startTime | Integer representation of the starting time of the timer. | No | 1 |
| startingAmountInSeconds | A float represents the original value of the timer when it started. Practical if the timer needs to be reset. | No | 1 |
| Callback | The callback which is to be called when the timer expires. | No | 1 |
| timeRemaining | Derived attribute. Represents the time remaining on the timer | No | N/A |

**Operations**

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Description | Is query? | Is Polymorphic? |
| Start | Start the timer | No | No |
| Pause | Pause the timer | No | No |
| Resume | Resume the timer | No | No |
| Reset | Reset the timer to its original value | No | No |
| GetRemainingTime | Get the time remaining on the timer. | Yes | No |
| SetTimerCallback | Sets the function that will be called when the timer | No | No |

**Processing Outlines ->**

Start

Set startTime to the current time. Start an async process to check when startTime – currentTime = startingAmountInSeconds.

When the timer finishes, call the set callback function.

Pause

Sets startTime to the current time, holds a temp variable to determine when and where to resume from.

Resume

Continue the async function with startTime modified.

Reset

Reset the startTime to now and stop the async process.

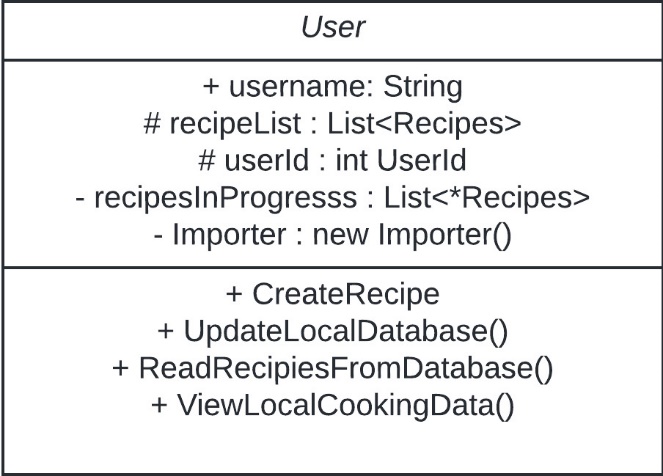
GetRemainingTime

Return Get currentTime – startingAmountInSeconds.

SetTimerCallback

Set the timer’s callback function.

1. User



Description: Abstract class for a user of the application.

Visibility: Public

Is Abstract: Yes

Additional information:

**Attributes**

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Description | Read only? | Multiplicity |
| Username | The name of the user using the application | No | 1 |
| recipeList | The list of recipes that the user can read and write to, generated from a local database. | No | 1 |
| userId | The unique identifier of a user. | Yes | 1 |
| recipesInProgress | The list of recipes that are in progress for the user but are not complete. | No | 1 |
| Importer | The importer object designated for the user. | No | 1 |

**Operations**

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Description | Is query? | Is Polymorphic? |
| CreateRecipe | Create a recipe and add it to the recipeList | No | No |
| UpdateLocalDatabase | Update the local database. | No | No |
| ReadRecipiesFromDatabase | Read recipes from the database and add them to the recipe list. | Yes | No |
| ViewLocalCookingData | Go through the list of recipes and aggregate statistics to show the user about their cooking. | No | No |

**Processing Outlines ->**

CreateRecipe()

Create a recipe and add it to the recipe list. The function should call several other helper functions to get the several pieces of data necessary for a recipe.

UpdateLocalDatabase()

Send an update to the locally stored database.

ReadRecipiesFromDatabase(string query)

Function is called on initialization. Reads all the recipes from the local database and places them into the user object.

ViewLocalCookingData()

Get and display recipe statistics from each recipe in the RecipeList.

# **Architecture Design**

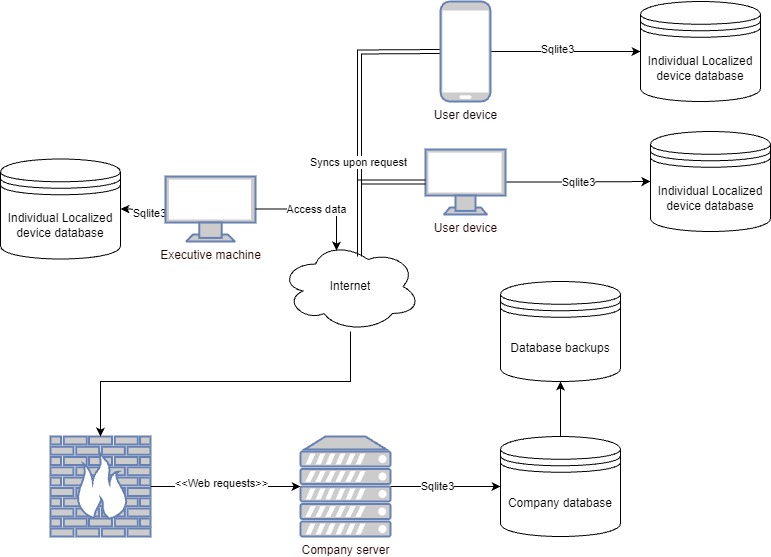
## **Architecture Overview**

The cooking optimization application will use two separate architectures depending on the mode of operation. A Model-View-Controller (MVC) architecture is necessary for individual users. This structure is so that the view handles the front-end business while delegating database work to the backend. An MVC architecture also benefits the application by keeping the local database separate from the network. However, the architecture changes to a client-server relationship if the user switches to a corporate database structure. In this architecture, the clients are the individual devices that sync and query the central corporate database server. Data will be stored in a relational database, and the front-end structure will probably be .NET MAUI.

The following two diagrams show the architecture and contact design of the application, using a first overall graphical view and then a closer view with a node diagram. Afterward, necessary software and hardware are listed, along with a security plan in case of emergencies or breaches.

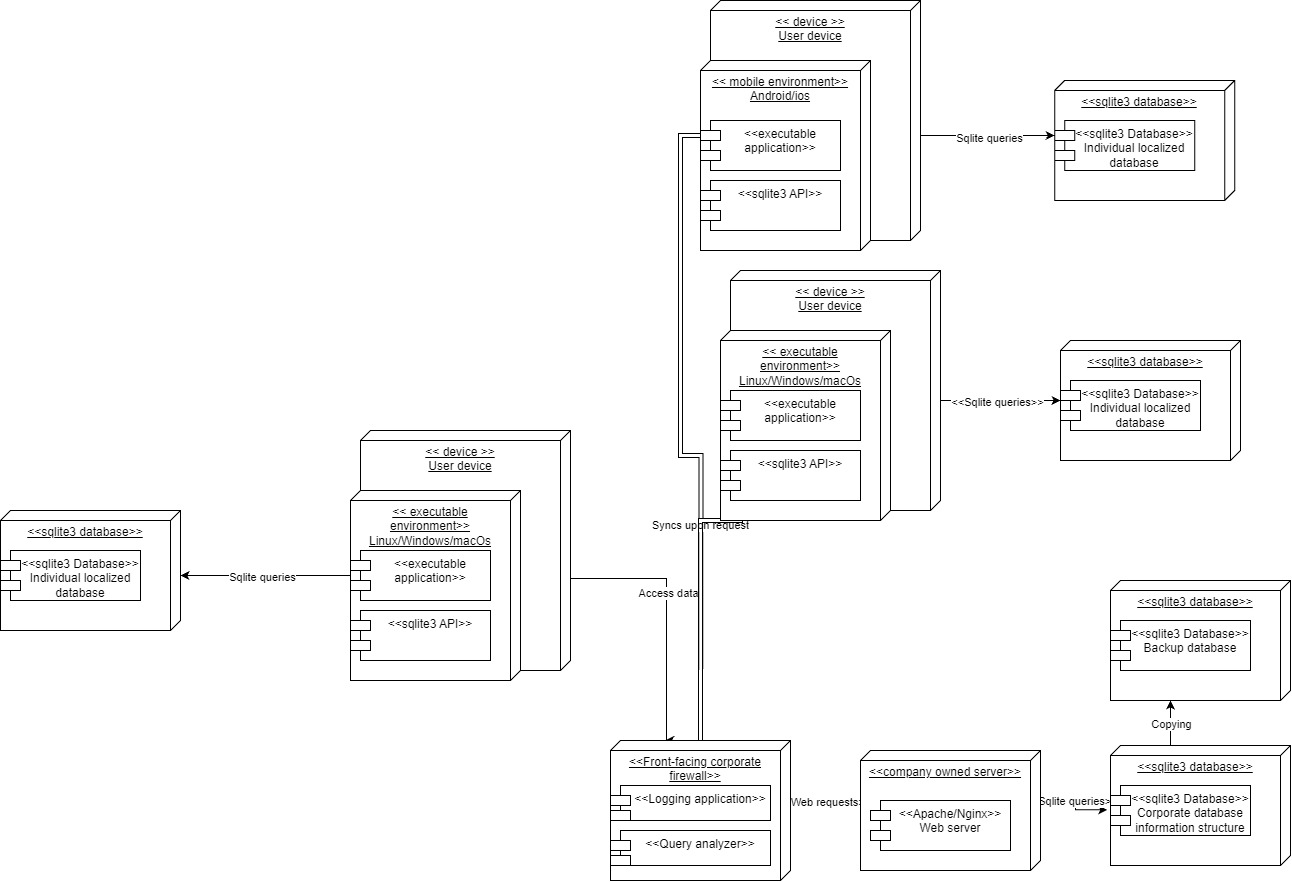
## **Infrastructure Model**

* + 1. **Deployment Diagram 1 – Architecture Overview**



Deployment Diagram

* + 1. **Deployment Diagram 2 – Nodes and Artifacts**



Node and artifact diagram

## **Hardware and Software Requirements**

On-hand parts and hardware should be sufficient to develop the listed application. The only hardware requirements are an on-hand server for executive users and a front-facing web server and database management.

* + 1. **Hardware Components**

The architecture, when adopting a corporate structure, requires extra hardware to ensure functionality for its customers. Customers’ external devices have no additional hardware requirements. However, the system also requires hardware to process and control requests to a corporate-rented database. As a result, the necessary hardware components are a backend company database, a backup database, and a front-facing web server able to parse requests.

* Company webserver: A rented or locally hosted webserver must receive and process web requests for the owned backend database.
* Company database: This application uses Sqlite3, a relatively lightweight database structure requiring minimal hardware and storage. Easy expansion and scalability, in this case, are secondary problems until traction is established.
* The users’ hardware shouldn’t be a significant problem. So long as their system is reasonably current, they should be able to use the application.
  + 1. **Required Software Components**
* Visual Studio 2022: Required for developing with .NET and .NET MAUI 8. Relatively simplistic and easy-to-use Integrated development environment IDE for experienced and new developers.
* Android Emulator: Required for testing on Android without physical devices.
* Ios Emulator: Required for testing iOS without physical devices.
* MacOS images: These are required to test MacOS in a virtual environment.
* Windows 11 images: Windows images are required to ensure the application works in a Windows environment.
* GNU/Linux development Environment (Ubuntu, Debian, OpenSUSE, Long term support (LTS) distributions): Required to test package necessities, prerequisites, and functionality in standard distributions.
* VirtualBox virtualization software 7.0: This software is required to run several images in a stable environment.
* Sqlite3 .NET package: Needed to maintain contact with backend Sqlite3 databases using .NET MAUI 8. Several implementations are available.
* Sqlite3 on backend server: To complete required web requests.
* Apache/Nginx webserver: Required to serve web requests and queries.
* .NET MAUI 8: Needed for simultaneous Apple, Android, and PC development.
* Git version control: This software is necessary for source control. Git keeps progress secure and reduces the risk of redundant and unusable code.
* .NET Testing Framework: Provides a way to test multiple features independently and efficiently.

## **Security Plan**

* + 1. **Security Overview**

Security within the cooking optimization application is not a significant concern for individual users. The attack surface of the individualized application is minimal, as the software shouldn’t have any contact with the internet until the user requests to be part of a corporate network. These factors make attacks on individuals rare and solvable by a simple backup system.

On the corporate side of this application, hosting a dedicated database, server, and backend requires planning, access controls, and backups. The three-part plan is as follows: First, have a registered disaster and incident response plan available so that staff and others know who to contact and how to handle the situation when something goes wrong. Second, strict access controls and encryption of web requests must be enforced. Finally, regular backups and security testing help close the remaining security gaps.

* + 1. **Security Plan**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **Disruption, intrusion, destruction, disaster** | | | | | Unauthorized Access | | |
| Components\Threats | **Fire** | **Flood** | **Power loss** | **Circuit failure** | **Computer virus** | **External intruder** | **Internal intruder** | **Eavesdrop on communications** |
| Company servers | 1, 5, 2 | 1, 5, 8 | 1, 5 | 1, 5 | 2, 3, 5, 7, 10, 11 | 2, 3, 5, 11 | 2, 3, 5, 10, 11 | 6, 10 |
| Executive rented database | 1, 5 | 1, 5 | 1, 5 | 1, 5 | 2, 3, 5, 7, 10, 11 | 2, 3, 5, 11 | 2, 3, 5, 11 | 6, 10 |
| Localized databases |  |  |  |  | 4 | 2 |  |  |
| Personal devices |  |  |  |  | 4 | 2 |  | 6, 10 |
| People | 1 |  |  |  | 1 | 4 | 2, 3, 4 | 6 |

Controls

* + 1. Disaster recovery plan
    2. Sprinkler system
    3. Internet isolation from the localized database.
    4. Login and access controls.
    5. Employee and executive access keys
    6. Local backup capability.
    7. Frequent off-site backups of the company database.
    8. HTTPS enforcing
    9. Firewalls
    10. Frequent security inspections and tests
    11. Incident response plan

# **User-Interface**

## **User-Interface Requirements and Constraints**



A primary goal in creating this application is to minimize the number of clicks to get between individual sections. The current design makes getting specific information within three clicks or fewer impossible. Hence, the next best thing is to group individual elements, so that related information is within arm’s length for the user.

Another essential part of the User interface’s design was ensuring that the user could easily understand when navigating between recipes and the effect of variations. When using the application, the user should not be confused about their location or wonder how to reach an individual screen they can’t find.

Finally, a vital part of the display design is ensuring that the user never thinks about the localized database on their device. Generally, this should be hidden from the user, as it introduces unnecessary confusion and complication.

## **Window/Screen Navigation Diagram**

Below is the navigation diagram for the cooking optimization application. It has several connections and buttons. The back buttons are not pictured, and database functionality is absent from this initial draft of the user interface.

A diagram of a computer system

Description automatically generated with medium confidence

Screen navigation diagram



Screen navigation diagram again. A larger, vertical view.

## **UI Wireframes**

The following section of the document displays several wireframes for the cooking optimization application. The application is designed with a mobile-first approach, so only Android and iOS wireframes are available. These should clarify the complex user interface necessary for this application.

A screenshot of a computer

Description automatically generatedA menu of food on a white background

Description automatically generated

Homepage display and a few key menus.

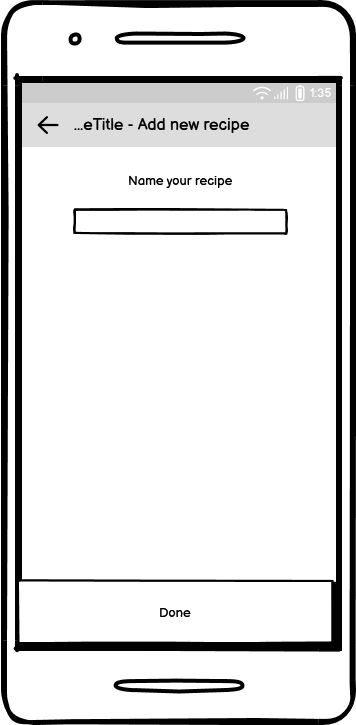
A wireframe of a recipe

Description automatically generatedA screenshot of a recipe

Description automatically generated  
  
A screenshot of a recipe

Description automatically generated

Display for selecting and considering a recipe and its variations.

A white rectangular object with black lines

Description automatically generated with medium confidence

A black and white notepad with lines

Description automatically generatedA screenshot of a recipe

Description automatically generated

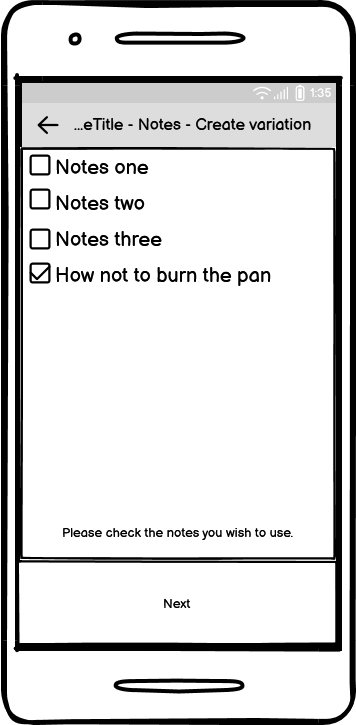
Wireframes that arerelevant for adding a new recipe to the database.

A white rectangular sign with black text

Description automatically generatedA white rectangular object with black lines

Description automatically generated with medium confidenceA screenshot of a recipe

Description automatically generated

A recipe list on a cellphone

Description automatically generatedA screenshot of a computer

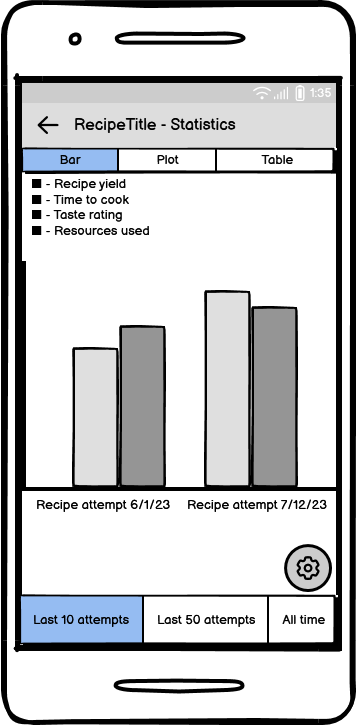
Description automatically generatedPa

Wireframes for creating a recipe variation – Part #1

A screen shot of a login form

Description automatically generated

Wireframe for creating a recipe variation – Part #2

A screen shot of a graph

Description automatically generatedA screenshot of a phone

Description automatically generated

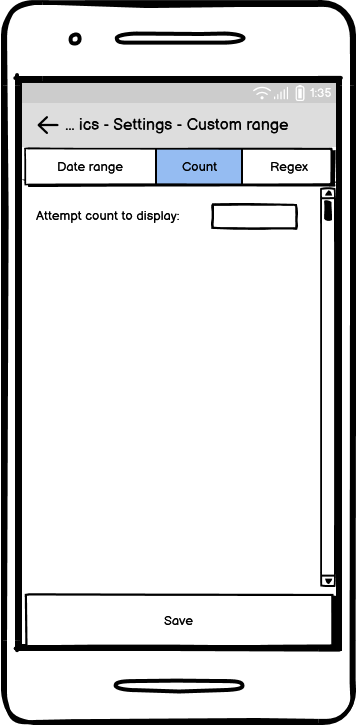
A screenshot of a computer

Description automatically generatedA screen shot of a white board

Description automatically generatedA screenshot of a calendar

Description automatically generated

Recipe statistics display and settings.

A screenshot of a computer

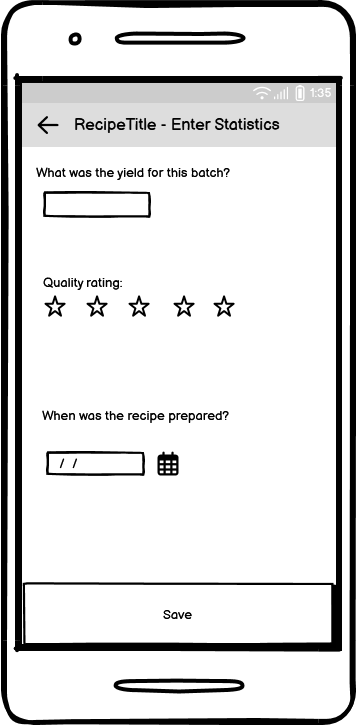
Description automatically generated

Recipe statistics display settings part #2.A white and black recipe

Description automatically generated with medium confidenceA close-up of a recipe

Description automatically generatedA screen shot of a computer

Description automatically generated

A white rectangular sign with black text

Description automatically generatedA white rectangular object with a black border

Description automatically generated

Relevant wireframes for doing a recipe and entering statistics.

## **Reports: “Formal Output” Design**

N/A

# **Appendices**

## **Glossary**

|  |  |
| --- | --- |
| **Term** | **Definition** |
| Access keys | Cryptographic keys which are used to verify a user’s identity or affiliation. |
| Android | The operating system developed by Google and installed on most mobile devices. |
| Database | A usually large collection of data organized especially for rapid search and retrieval (as by a computer) |
| Disaster recovery plan | A specialized plan designed to recover systems and data in the event of a natural or caused disaster situation. |
| Emulator | Hardware or software that permits programs written for one computer to be run on another computer |
| Firewalls |  |
| GNU/Linux | An open-source operating system that is popular with the developer community. |
| HTTPS | Hypertext Transfer Protocol Secure. Protocol used for most network communication. |
| IDE | Integrated development environment. The application in which code is written, run, and tested. |
| Incident response plan | A specialized plan designed to give staff and security personnel a quick guide in case of a breach or security incident. |
| iOS | Apple’s proprietary mobile operating system. |
| Long term support (LTS) | Describes the length of time that a particular piece of software will be actively maintained and supported. |
| .NET MAUI | .Net Multi-platform app UI. A framework designed by Microsoft for cross-platform development. |
| Model, view, controller (MVC) | A computer design pattern is meant to represent splitting the system into three distinct parts: The model, representing the data; the view, representing the interface; and the controller, representing the operations and in-between for the view and model. |
| Unified modeling language (UML) | A common modeling language used to describe classes and relationships inside applications. |
| User Interface (UI) | The interface that a user interacts with. |
| Webserver | A server that handles and serves web requests. |
| Windows | Ubiquitous operating system used by most home users. |
| .NET | A framework built by Microsoft for building C# applications. |

## **References / Bibliography**

*Adobe photoshop*. (n.d.). Adobe Photoshop. https://www.adobe.com/products/photoshop.html

*Balsamiq: Fast, focused wireframing for teams and individuals | Balsamiq*. (n.d.). Retrieved June 4, 2024, from https://balsamiq.com/

davidbritch. (2024, January 19). *Net maui local databases—. Net maui*. https://learn.microsoft.com/en-us/dotnet/maui/data-cloud/database-sqlite?view=net-maui-8.0

*Definition of database*. (2024, June 3). https://www.merriam-webster.com/dictionary/database

*Definition of emulator*. (2024, May 19). https://www.merriam-webster.com/dictionary/emulator

*Flowchart maker & online diagram software*. (n.d.). Retrieved June 4, 2024, from https://app.diagrams.net/

Foundation, B. (n.d.). Blender. Org—Home of the blender project—Free and open 3d creation software. *Blender.Org*. Retrieved June 4, 2024, from https://www.blender.org/

Larman, C. (n.d.). *Applying UML and patterns: An introduction to object-oriented analysis and design and iterative development* (3rd ed). Prentice Hall PTR, c2005.

*Lucid visual collaboration suite: Log in*. (n.d.). Retrieved June 4, 2024, from https://lucid.app/users/login

*Mvc design pattern*. (2017, August 18). GeeksforGeeks. https://www.geeksforgeeks.org/mvc-design-pattern/

*Net | build. Test. Deploy.* (n.d.). Microsoft. Retrieved June 4, 2024, from https://dotnet.microsoft.com/en-us/

*Net multi-platform app ui  (Net maui)|. Net*. (n.d.). Microsoft. Retrieved June 4, 2024, from https://dotnet.microsoft.com/en-us/apps/maui

*Sqlite home page*. (n.d.). Retrieved June 4, 2024, from https://www.sqlite.org/index.html

*Terms of service—Dotdash meredith*. (n.d.). Retrieved June 4, 2024, from https://www.dotdashmeredith.com/brands-termsofservice

*What does copyright protect? (Faq)| u. S. Copyright office*. (n.d.). [Web page]. Retrieved June 4, 2024, from https://www.copyright.gov/help/faq/faq-protect.html

## **Supporting documentation**

N/A