UNIVERSITÀ DI PISA

RECIPE SHARE

DOCUMENTATION

Lorenzo Mazzei, Leonardo Bargiotti, Matteo Manni

GitHub Repository: [***https://github.com/MatteoManni99/RecipeShare***](https://github.com/MatteoManni99/RecipeShare)

Index

1 INTRODUCTION AND REQUIREMENTS

Recipe Share is a social networking application that gives you the opportunity to share your recipes with other people. Users who register to the service will become an Author and he'll be able to interact with the recipes already published by other people, giving them ratings as well as report them among other things. Data come from two different Dataset: one from Kaggle sourced from the website **food.com** and the other generated through web scraping on **allrecipes.com**. The application allows the Authors to describe their recipes in detail, specifying ingredients, a brief description, recipe instructions, adding images and so on.

The service also includes moderators in order to keep track of the recipes that contains out of place information, such as bad language or inappropriate images and so on.

1.1 Functional Requirements

In this application there can be three different types of users: User, Author and Moderator. More details about the actions that can be performed by each of them are shown below:

A User will be able to:

- Create an account, becoming an Author.
- Login to the service

An Author will be able to:

- If he/she was offered a promotion by a Moderator, they can reject/accept the offer.
- If he/she accept, then he/she can create an account as a Moderator.
- Browse recipes.
- Find recipes.
- Once a recipe is selected, he/she can see its details.
- Once a recipe is selected, he/she can write a review of the recipe.
- Once a recipe is selected, he/she can report the recipe to the Moderators.
- Update his/her personal profile (changing avatar image or password).
- Browse their own recipes.
- Find their own recipes.
- Once one of their recipes is selected, they can delete it.
- Browse other Authors.
- Find Authors.
- Once an Author is selected, he/she can see the details of the Author and the Recipes he/she wrote.
- Follow/Unfollow other Authors.
- View the top-rated recipes.
- Consult the recipes suggested by the Application.
- Logout.

A Moderator will be able to:

- Browse Authors.
- Find Authors.
- Once An author is selected, he/she can see the details of the author and the Recipes he/she wrote.
- Browse Recipes Reported by the Authors.
- Once a Recipe is selected, they can approve/reject the Recipe.
- Offer an Author the possibility to become a Moderator.
- Browse Malicious Authors (Authors who have created Recipes that have been reported)
- Logout

1.2 Non-Functional Requirements

As said previously, the structure of the application is based on the idea of a social network where authors can share and visualize recipes. Therefore, the points shown below are critical to provide an efficient and pleasant service:

- The system must be Fast-Responsive
- The system must guarantee High-Availability of Contents
- The system must be Fault-Tolerant, such as that the failure of a node doesn't compromise the service.
- The system must be User-Friendly and intuitive to use.
- The system must be available 24/7.

2 SPECIFICATIONS

2.1 Actors and Use Case Diagram

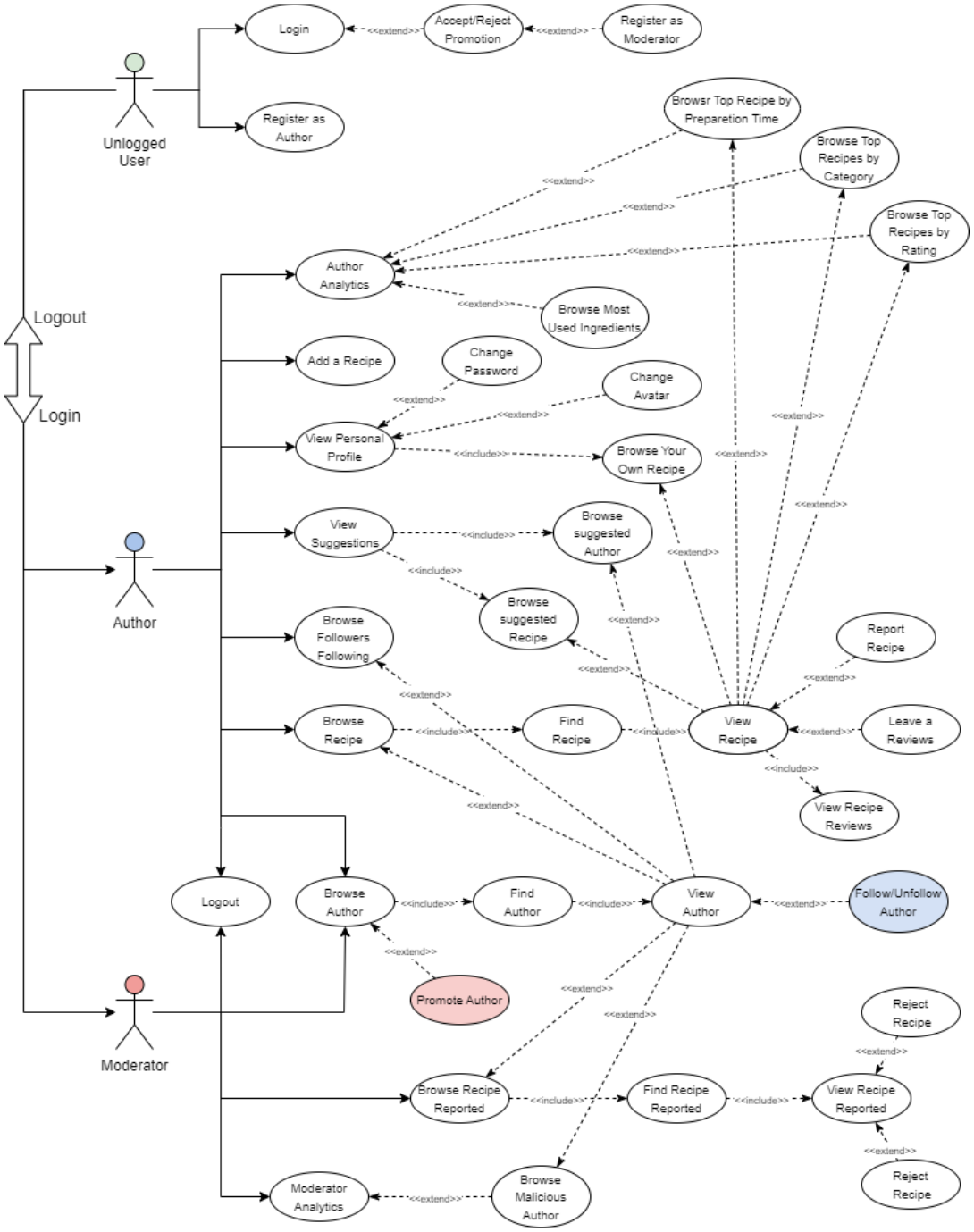
There are 3 actors in the application as specified previously:

User: only able to register or login to the application.

Author: logged user, can interact with all the services of the application, may also be able to become a moderator if offered.

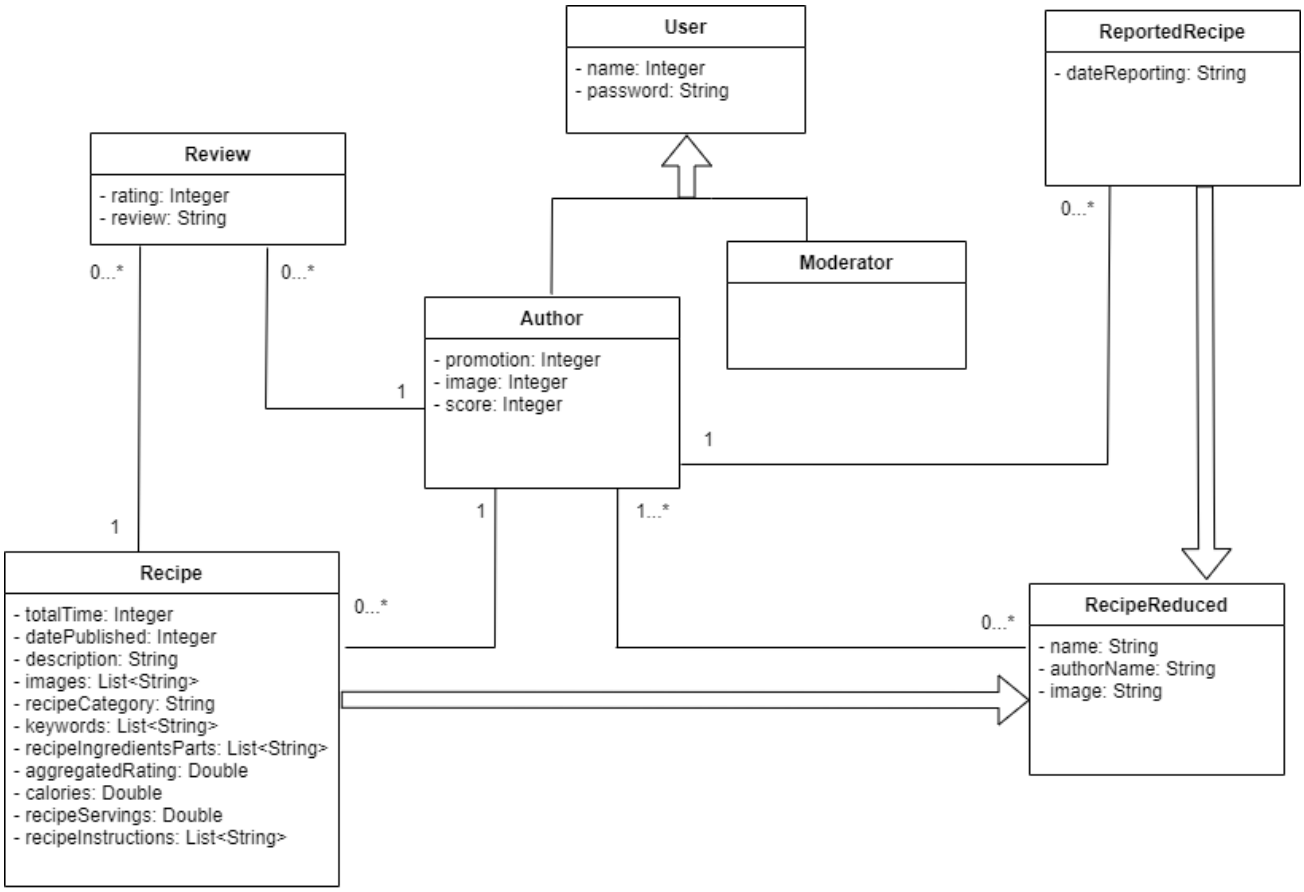
Moderator: able to view authors and reported recipes which they can delete if necessary.

Use-Case diagram:



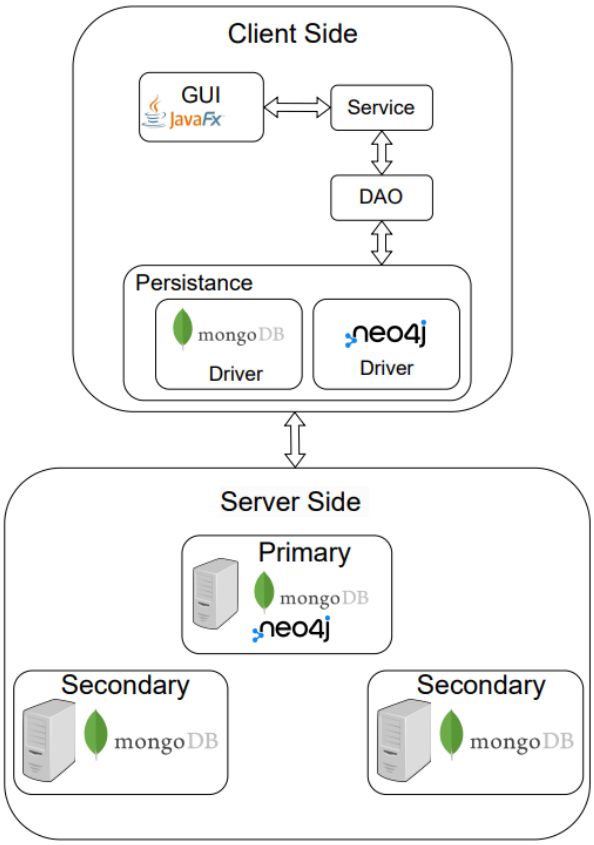
2.2 Analysis Class Diagram

The following image shows the main classes that define the structure of the application:



3 ARCHITECTURAL DESIGNS

3.1 Software Architecture



3.2 Packages

Recipe Share was developed as a JavaFX application. The code was organized in packages, more specifically the packages are:

- Dao
- Gui
- Model
- Persistence
- Service

3.2.1 GUI Package

This package contains the classes that implements the graphic interface and the events which handle the interactions of the user with application. In particular, the application uses tables to present a significant amount of data to the user: the tables are managed by two sub-packages:

- TableView: contains the classes that define the general structure of each table.
- Row: contains the classes that define the structure of the rows of each table.

3.2.2 DAO Package

This package is split into two sub-packages:

- Mongo: it contains the classes that perform the queries to MongoDB.
- Neo: it contains the classes that perform the queries to Neo4j.

3.2.3 SERVICE Package

It has the purpose to prevent direct access to the Databases from the Business logic, basically the classis in this package stands in the middle between the business classes and the Dao classes.

3.2.4 PERSISTENCE Package

It contains the classes that implements the drivers both for MongoDB and Neo4j. These classes are used in the code every time an access to a database occur.

3.2.5 MODEL Package

It contains the classes that represent the main entities of our application:

- Author
- Moderator
- RecipeReduced
- Recipe
- ReportedRecipe
- Review

3.2.6 Configuration File

In addition to the packages mentioned above, there is also a configuration file where the string for the connections to the databases are stored, as well as the name of the collections and the paths to the avatar images that the user can choose as his/her profile picture.

3.3 Dataset Composition

The application revolves around Recipes, therefore the raw data used for the application come from datasets characterized by information of recipes already published online with reviews integrated. For the sake of variety, we took data from two different sources:

- **kaggle.com -> food.com**
  https://www.kaggle.com/datasets/irkaal/foodcom-recipes-and-reviews
- **allrecipes.com**
  https://www.allrecipes.com

The Raw data we took also had nicknames of the people who wrote the recipe, so we decided to base the dataset of the authors on the already existing nicknames, in addition to artificial nicknames randomly generated by us to populate the database. The password of the authors was also randomly generated.

Regarding the reported recipe dataset, it started as empty and was populated dynamically using the associated service of reporting in the application.

Some modifications we made from the raw data: the IDs were removed and we decided to use the names of the authors and recipes as identification field in their respective dataset.

3.4 Databases Choices

Two different types of databases were used in the making of the application: MongoDB and Neo4j. Here below more details and explanations regarding these choices:

- **MongoDB**: it was chosen because in the stored data it allows the use of Indexes in order to retrieve information quickly: given that our application was made with the intent of being fast-responsive, meaning that we want reading operations to be fast, the indexes can help us in this aspect. Also, as it'll be shown later, reviews of the recipes were stored as embedded documents in the recipe collection, so the embedding feature of MongoDB was also important in the realization of the service.
- **Neo4j**: It was fundamental for the realization of another social-network part of our application: authors that can follow each other. This aspect is also taken into account in the analytic section, in this sense a Graph Database allows us to perform this kind of queries in an optimal way, exploiting the relationship properties between nodes.

An aspect that can be discussed regards the location of the author dataset: we decided to store it as a collection in MongoDB, but there was another possibility which was to store it in Neo4j. The motivations behind our choice come from the fact that our application has to look into all the stored data of the authors: despite these occurrences being very few, having to scan all the nodes in Neo4j would have slow down the operations too much, and instead MongoDB guarantees us a quicker retrieval of the data.

4 MONGODB - DESIGN AND IMPLEMENTATION

This database is composed of 4 collections:

- Author
- Moderator
- Recipe
- ReportedRecipe

Details about them are written here below.

4.1 Collection

4.1.1 Author Collection

Once a user register to Recipe Share, his/her account is created based on the nickname, password and profile image they chose. The information is therefore stored in the collection. The "promotion" field is set to 0 when the user registers and it is used to keep track of the promotion offers that can be proposed to the author by one of the moderators. Accesses to this collection occur when other authors and/or moderators wants to see the profile of the selected author, it can be done by clicking the name of the author in every table that contains this information. Below an example of a Document:


```
{
  "_id": {
    "$oid": "63e92285f8fda838bafd26c5"
  },
  "authorName": "Malarkey Test",
  "password": "eTHuB766",
  "promotion": 0,
  "image": 7
}
```


4.1.2 Moderator Collection

The information regarding moderators is stored in this collection. An author can't become a moderator unless a promotion offers by a moderator. A new moderator document is stored in the database only when an author accepts a promotion offer: at this point the application will have the author create another account which will be his/her account as moderator while they'll be able to keep the previous account, therefore having the possibility to login as an author or as a moderator. Accesses to this collection occur when a user login as moderator. Below an example of a Document:

```
{
  "_id": {
    "$oid": "63e922acf8fda838bafe2ace"
  },
  "moderatorName": "Mueller",
  "password": "QU012zhZ"
}
```


4.1.3 Recipe Collection

The biggest collection, all the information of recipes is stored in this collection. Given the size of a single document from this collection, we decided to show only some of them to the authors during the browsing phase (name, first image, recipe's author). The complete information regarding a recipe can be visualized when an author or a moderator click on the name of the recipe in every table that contains them. Therefore, we can reduce the access time to the database retrieving few information when an author/moderator browse the recipes, and the complete access to all the data is performed only in the aforementioned case.


```
{
  "_id": {
    "$oid": "63e91d58f8fda838bafb3ba3"
  },
  "Name": "Butter Pecan Cookies",
  "AuthorName": "mommyoffour",
  "TotalTime": 25,
  "DatePublished": "2008-05-27 23:25:00 +00:00",
  "Description": "Make and share this Butter Pec
  "Images": [
    "https://img.sndimg.com/food/image/upload/w_
  ],
  "RecipeCategory": "Dessert",
  "Keywords": [
    "Cookie & Brownie",
    "< 30 Mins"
  ],
  "RecipeIngredientParts": [
    "butter",
    "egg",
    "baking powder",
    "pecans",
    "sugar",
    "flour",
    "vanilla",
    "salt"
  ],
  "AggregatedRating": 5,
  "Calories": 2093.5,
  "RecipeServings": null,
  "RecipeInstructions": [
    "Cream butter and sugar.",
    "Add egg.",
    "Add flour, salt, baking powder and pecans.
    "Mix in vanilla.",
    "Chill dough 1 hour.",
    "Roll out 1/8 inch thick and cut with any s
    "Bake at 400 for 8 minutes or until nice ar
  ],
  "Reviews": [
    {
      "AuthorName": "CulinaryGourmetQueen",
      "Rating": 5,
      "Review": "I doubled the batch, and made
    },
    {
      "AuthorName": "EddyGirl",
      "Rating": 5,
      "Review": "I actually made these last nig
    }
  ]
}
```

As said previously, the reviews are stored as an embedded document in this collection. This redundancy allows us to quickly retrieve all the reviews regarding a specific recipe by simply accessing the recipe document and scan the “Reviews” field.

4.1.4 ReportedRecipe Collection

This collection contains the necessary information to keep track of every occurrence of authors reporting a recipe. The motivation behind this action is to be searched in possible out of place description containing scurrile language, images that do no not represent the recipe, and so on with the other key aspects of a recipe. The field “authorName” and “reporterName” contains respectively the name of the author who wrote the recipe and the name of the author who reported the recipe, allowing a fast retrieve of all the authors information. Same can be said for the information of the recipes. Accesses to this collection occur when a user login as a moderator. Visualizing the full page of a recipe, the moderator will be able to reject or approve a reported recipe. Below an example of a Document:


```
{
  "_id": {
    "$oid": "63e23c45fe72d508c38a39c1"
  },
  "name": "Cabbage Soup",
  "authorName": "katii",
  "reporterName": "Dancer",
  "dateReporting": "2023-02-07",
  "image": "https://img.sndimg.com/food/image/up
}
```


4.2 CRUD Operations

4.2.1 Create

The functions used to create documents are:

- AuthorMongoDAO.registration: create an author document.
- ModeratorMongoDAO.checkRegistration: create a moderator document.
- RecipeMongoDAO.addRecipe: create a recipe document.
- ReportedRecipeMongoDAO.addReportedRecipe: create a reportedRecipe document.

4.2.2 Read

The functions used to read documents are:

- AuthorMongoDAO.tryLogin: check username and password for the author login.
- AuthorMongoDAO.getAuthor: get the author document with the name specified.

- AuthorMongoDAO.checkIfUsernameIsAvailable: check if the author document with the name specified exists.
- AuthorMongoDAO.searchAuthors: retrieve all author documents that contains the string specified using partition, meaning that at last only x authors will be shown at a time, where x is a number to be specified.
- ModeratorMongoDAO.checkModeratorName: check if the moderator document with the name specified exists.
- ModeratorMongoDAO.tryLogin: check if the moderator document with the name and password specified exists.
- RecipeMongoDAO.getRecipeByName: get the recipe document with the name specified.
- RecipeMongoDAO.calculateAggregatedRating: returns the aggregated rating of the recipe document with the specified recipe name.
- RecipeMongoDAO.checkIfReviewSpaceIsFull: check if the recipe document with the specified name has reached the maximum number of reviews it can have.
- RecipeMongoDAO.checkIfNameIsAvailable: check if the recipe document with the specified name exists.
- RecipeMongoDAO.getRecipeFromAuthor: get the name and first image of all the recipe documents with the authorName specified.
- RecipeMongoDAO.getRecipeFromName: get the name, the authorName and the first image of all the recipe documents that contains the specified string in the name, visualizing only x of these documents at a time, where x is the number specified.
- RecipeMongoDAO.findTopRecipesForRangesOfPreparationTime: returns the results of the query “Top Recipes For Ranges Of Preparation Time”.
- RecipeMongoDAO.findMostUsedIngredients: returns the results of the query “Most Used Ingredients”.
- RecipeMongoDAO.findRecipesWithHighestRating: returns the results of the query “Recipes With Highest Rating”.
- RecipeMongoDAO.findTopRecipesForEachCategory: returns the results of the query “Top Recipes For Each Category”.
- ReportedRecipeMongoDAO.getListReportedRecipes: get all the reportedRecipe documents that contain the specified string in the name, visualizing only x of these documents at a time, where x is the specified number.

- ReportedRecipeMongoDAO.onHighestRatioQueryClick: returns the results of the query “Highest Ratio”.

4.2.3 Update

- AuthorMongoDAO.updateImage: update the image in the author document with the specified name.
- AuthorMongoDAO.updatePromotion: update the promotion in the author document with the specified name.
- AuthorMongoDAO.changePassword: update the password of the specified author.
- RecipeMongoDAO.setAggregatedRating: update the aggregated rating of the recipe document with the specified name.
- RecipeMongoDAO.addReview: update the reviews embedded documents in the recipe document with the specified name by adding a review to it.

4.2.4 Delete

- RecipeMongoDAO.deleteRecipe: delete the recipe document with the specified name.
- ReportedRecipeMongoDAO.removeReportedRecipe: delete the reportedRecipe document with the specified name.

4.3 Queries Analysis

As shown above, our application is strongly based on read operations, while the write operations are less frequent. We implemented paginations method in order to balance the access to database, this way only a certain number of documents (10) will be visualized at a time. Taking into consideration the non-functional requirements, especially high-availability and fast-responsive, we decided to set the following options:

- Write concern: 1.
- Read preference: nearest.

The first option allows the write operations to be performed in the fastest way: in fact, the write will be done in the primary node and implements the eventual consistency regarding the secondary nodes, meaning that they will be updated eventually in the future (we chose to focus on the AP of the CAP triangle, so the Consistency isn’t a strict requirement). For the second option we concluded that “nearest” was the best possibility, given that it satisfies the fast-response requirement: this way, the read operations are performed on the node that guarantees the least network latency among the nodes.

We also decided to create a second connection to the database with the following settings:

- Write concern: 3.
- Read preference: nearest.

This part has the purpose of guaranteeing consistency in operations in the sections of the applications that regard critical adding of entities to the database. For example, in the class RegisterController the username which the user chose to create a new account is checked to make sure it doesn't already exist one with the same name. This is a critical part of the application: if due to a lack of consistency two user with the same name are added, eventually an error will occur when the application tries to retrieve a document that has that name as key because the system won't be able to understand which one to pick between the two of them.

4.4 Analytics

Some analytics were created in order to retrieve useful information from the recipes posted by the authors during time and from interactions of the authors with social network. Details are shown below.

4.4.1 Top Recipes for Ranges of Preparation Time

Select the recipes with the highest rating for each range of preparation time.

Parameters:

- `lowerLimit:` Lower bound of the range.
- `upperLimit:` Upper bound of the range.
- `minNumberReviews:` Minimum number of reviews that the recipes need to have.
- `limitRecipes:` Maximum number of recipes returned by the analytic.

Method: RecipeMongoDAO.findTopRecipesForRangeOfPreparationTime

```
db.recipe.aggregate([
  {
    $match: { TotalTime: { $gt: lowerLimit, $lte: upperLimit } }
  },
  {
    $match: { [Reviews.minNumberReviews]: { $exists: true } }
  },
  {
    $sort: { AggregatedRating: -1 }
  },
  {
    $limit: limitRecipes
  },
  {
    $project: {
      Name: 1,
      TotalTime: 1,
      AggregatedRating: 1,
      Images: { $first: $Images }
    }
  }
])
```

4.4.2 Most Used Ingredients

Select the ingredients that were used the most in the making of the recipes.

Parameters:

- `minNumberReviews:` Minimum number of reviews that the recipes need to have.
- `limitIngredients:` Maximum number of ingredient returned.

Method: RecipeMongoDAO.findMostUsedIngredients

```
db.recipe.aggregate([
  {
    $match: { [Reviews.minNumberReviews]: { $exists: true } }
  },
  {
    $project: { RecipeIngredientParts: 1 }
  },
  {
    $unwind: $RecipeIngredientParts
  },
  {
    $group: {
      _id: $RecipeIngredientParts,
      count: { $sum: 1 }
    }
  },
  {
    $sort: { count: -1 }
  },
  {
    $limit: limitIngredients
  }
])
```

4.4.3 Recipes with Highest Rating

Select the recipes with the highest aggregated rating.

Parameters:

- `skipRecipes:` Number of recipes skipped by the analytic.
- `limitRecipes:` Number of recipes returned by the analytic.
- `minNumberReviews:` Minimum number of reviews that the recipes need to have.

Method: RecipeMongoDAO.findRecipesWithHighestRating

```
db.recipe.aggregate([
  {
    $match: { [Reviews.minNumberReviews}]: { $exists: true } }
  },
  {
    $sort: { AggregatedRating: -1 }
  },
  {
    $skip: skipRecipes
  },
  {
    $limit: limitRecipes
  },
  {
    $project: {
      Name: 1,
      AggregatedRating: 1,
      Images: { $first: $Images }
    }
  }
])
```

4.4.4 Top Recipes for Each Category

Select the recipes with the highest rating for each category a recipe can be classified as.

Parameters:

- `minNumberReviews:` Minimum number of reviews that the recipes need to have.

Method: RecipeMongoDAO.findTopRecipesForEachCategory

```
db.recipe.aggregate([
  {
    $match: { [Reviews.minNumberReviews]: { $exists: true } }
  },
  {
    $sort: { AggregatedRating: -1 }
  },
  {
    $group: {
      _id: $RecipeCategory,
      Name: { $first: $Name },
      AggregatedRating: { $first: $AggregatedRating },
      Images: { $first: $Images }
    }
  }
])
```

4.4.4 Author Score

Select the authors with the lowest ratio of recipes posted compared to the number of his/her recipes that were reported. The lower the ratio, the worse the author's reputation.

$$Author\ Score = \frac{Number\ of\ Recipes\ posted}{Number\ of\ reports\ of\ her/his\ recipes}$$

Parameters:

`reportedAuthor`: The authors whose recipe was reported. It's a list dynamically created with the first query. `**(List) reportedAuthor.add(document.getString("_id"))**`

Method: ReportedRecipeMongoDAO.onLowestScoreQueryClick

```
db.reportedRecipe.aggregate([
  {
    $group: {
      _id: "$authorName",
      count: { $sum: 1 }
    }
  },
  {
    $match: { "count" : { $gte: 1 } }
  }
])
```

```
db.recipe.aggregate([
    {
        $match: {
            "AuthorName": {
                $in: reportedAuthor
            }
        }
    },
    {
        $group: {
            "_id": "$AuthorName",
            "count": {
                $sum: 1
            }
        }
    }
])
```

4.5 Indexes Structure

In order to improve the efficiency of read operations from the database, indexes were implemented. The details are shown below.

4.5.1 Search Index for Recipe

We took into consideration the analytics regarding the recipe collection, and we tried to implement an index for each of them, because they all have some kind of filtering.

Find Top Recipes for Ranges of Preparation Time

In this case, excluding the parameters for the pagination, we had two filters possible to work on: *totalTime* and *AggregatedRating*. The query does use a sort operation on the rating to return the best ones, but the *AggregatedRating* field is updated every time someone writes a review for the recipe itself, so an index on it would be too expensive. Therefore, we decide to create an index based on the *totalTime* field instead of a compound index using also *AggregatedRating*:

Index: TotalTime_1

Case without index:

```
executionTimeMillis: 1028,
totalKeysExamined: 51048,
totalDocsExamined: 51048,
```

The execution time of the query is 1028 milliseconds.

Case with Index:

db.recipe.find({$and:[{TotalTime:{$gt:30}},{TotalTime:{$lt:90}}]}).sort({AggregatedRating:-1}).hint({TotalTime:1}).explain("executionStats")

```
executionTimeMillis: 990,
totalKeysExamined: 51048,
totalDocsExamined: 51048,
```

With the index the execution time is 38 milliseconds faster.

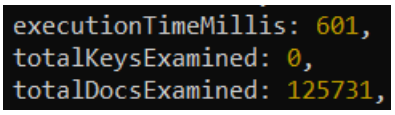
We can conclude that the index we applied is the better alternative between the two cases.

Find Top Recipes for Each Category

In this case we had two filters to work on as well. The AggregatedRating one couldn't be picked for the same motivations explained before, so the only choice we had was trying to create an index based on the RecipeCategory field:

Index: RecipeCategory_1

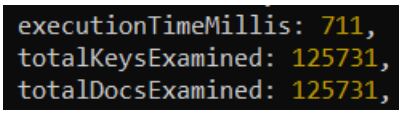
Case without Index:



The execution time is 601 milliseconds.

Case with Index:

db.recipe.aggregate([{ $group: { _id: "$RecipeCategory", Name: { $top: { output: ["$RecipeCategory", "$Name"], sortBy: { "AggregatedRating": -1 } } } } }],{hint:{RecipeCategory:1}}).explain("executionStats")



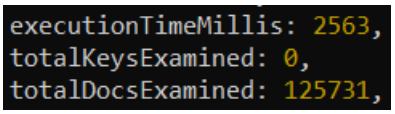
This time the execution of the query is slower with the index. A possible explanation for this outcome could be that the query has to scan the whole collection to return the results no matter what. In this sense an index is basically useless and has no purpose, therefore the only effect it has is a negative one. Therefore, it's better not to use any index this time.

Recipes with Highest Rating

In this case we are only looking for the recipes with the highest rating, therefore an index on the AggregatedRating field is optimal:

Index: AggregatedRating_1

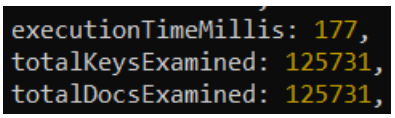
Case without Index:



Execution time is 2563 milliseconds.

Case with Index:

db.recipe.find({}).hint({AggregatedRating:1}).explain("executionStats")



Execution time is only 177 milliseconds.

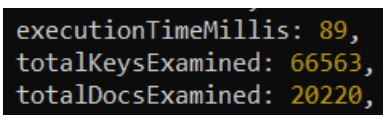
Clearly using the index is the best solution in this situation.

4.5.2 Search Index for Author

In this case we focused on the find operation for the authors' names: the filter was on the substrings contained in the name, so we decided to store an index in the database this way:

Index: authorName_1

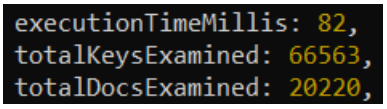
Case without Index:



As it can be seen, the execution time for the query is 89 milliseconds.

Case with Index:

db.author.find({authorName:{$regex:"c",$options:"i"}}).hint({authorName:1}).explain("executionStats")



As it can be seen, the execution time with the index is 7 milliseconds faster than without the index.

Therefore, using the index is a good solution.

5 NEO4J - DESIGN AND IMPLEMENTATION

Here is the schema of the Graph Database:

5.1 Composition

5.1.1 Nodes

Each node represents an entity of our application, in our Neo4j schema there are 2 nodes:

- Author: composed by “name” and “avatar”
- Recipe: composed by “name” and “image”

Where “avatar” and “image” are respectively the index of the 8 possible profile pictures of an author and the URL of the first picture of the recipe.

5.1.2 Relations

- FOLLOWS: Author – FOLLOWS -> Author: this relationship is added when an author starts to follow another author. Whenever an author unfollows the other author, the relationship is removed.
- AUTHOR – WRITES -> RECIPE: this relationship is added when an author creates a new recipe. If the recipe gets removed by the author him/herself or by a moderator because it was reported, the relationship is removed.
- AUTHOR – REVIEWS -> RECIPE: this relationship is added when the author leaves a new review to a recipe.

5.2 CRUD Operations

5.2.1 Create

- RecipeNeoDAO.addRecipe: creates a new Recipe node.
- AuthorNeoDAO.addAuthor: creates a new Author node.
- RecipeNeoDAO.addRelationWrite: adds the relation Write.
- AuthorNeoDAO.addRelationFollow: adds the relation Follow.

- AuthorNeoDAO.addRelationReview: adds the relation Review.

5.2.1 Read

- AuthorNeoDao.checkIfFollowIsAvailable: returns a boolean, false if the Follow relation between two author nodes already exists while it returns true if it is not present and therefore available.
- AuthorNeoDAO.getRecipeAdded: returns all recipes written by an author.

5.2.1 Update

- AuthorNeoDao.changeAvatar: change the avatar parameter to a new value.

5.2.1 Delete

- RecipeNeoDAO.deleteRecipe: deletes a Recipe node and all the relation connected to it.
- AuthorNeoDAO.removeRelationFollow: deletes a Follow relation.
- AuthorNeoDAO.removeRelationReview: deletes a Follow relation.

5.3 Queries

All next queries are focused on who requests the information. The application manages the results through pagination, the parameters used are:

- $authorName: corresponding to the name of the author requesting the query.
- $elementsToSkip: number of elements to skip required for paging.
- $elementsToLimit number of elements to limit required for paging.

5.3.1 Get Authors Suggested

Provides a different list for each Author, recommending the authors he should follow. It does so by selecting the authors who are followed by the authors who he/she follows. Obviously by removing him/herself and the authors he already follows from the result. Then sort in descending order based on the occurrences.

Query:

```
MATCH (a:Author {name: $authorName})-[:FOLLOW]→(f1:Author)-[:FOLLOW]→(f2:Author)
WHERE NOT (a)-[:FOLLOW]→(f2) AND f2.name <> $authorName
RETURN f2.name as Name, f2.avatar as Avatar, COUNT(*) as Frequency
ORDER BY Frequency DESC
SKIP $elementsToSkip
LIMIT $elementsToLimit;
```

Method: AuthorNeoDao.getAuthorSuggested:

5.3.2 Get Recipes Suggested

Provides a different list for each Author, recommending the recipes he/she should look at. It does so by selecting the recipes written by the authors who he/she follows, combined with the recipes positively reviewed by the author who/she follows. Obviously by removing his/her recipes and the recipes that are already reviewed from the result.

Query:

```
MATCH (a:Author {name: $authorName})
CALL{
    WITH a
    MATCH (a)-[:FOLLOW]→(b:Author)-[:WRITE]→(r:Recipe)
    WHERE NOT (r)←[:WRITE]-(a) AND NOT (r)←[:REVIEW]-(a)
    RETURN b,r
    UNION
    WITH a
    MATCH (a)-[:FOLLOW]→(b:Author)-[rev:REVIEW]→(r:Recipe)
    WHERE NOT (r)←[:WRITE]-(a) AND NOT (r)←[:REVIEW]-(a) AND rev.rating>2
    RETURN b,r
}
RETURN b.name as AuthorName, r.name as Name,  r.image as Image
SKIP $elementsToSkip
LIMIT $elementsToLimit
```

Method: AuthorNeoDao.getRecipeSuggested

5.3.3 Get Followers

Returns the list of the authors who he/she follows.

Query:

```
MATCH (f:Author)-[:FOLLOW]→(a:Author {name: $authorName})
RETURN f.name as Name, f.avatar as Avatar
SKIP $elementsToSkip
LIMIT $elementsToLimit
```

Method: AuthorNeoDao.getFollowers

5.3.4 Get Following

Returns the list of the authors who follow him/her.

Query:

```
MATCH (a:Author {name: $authorName})-[:FOLLOW]→(f:Author)
RETURN f.name as Name, f.avatar as Avatar
SKIP $elementsToSkip
LIMIT $elementsToLimit
```

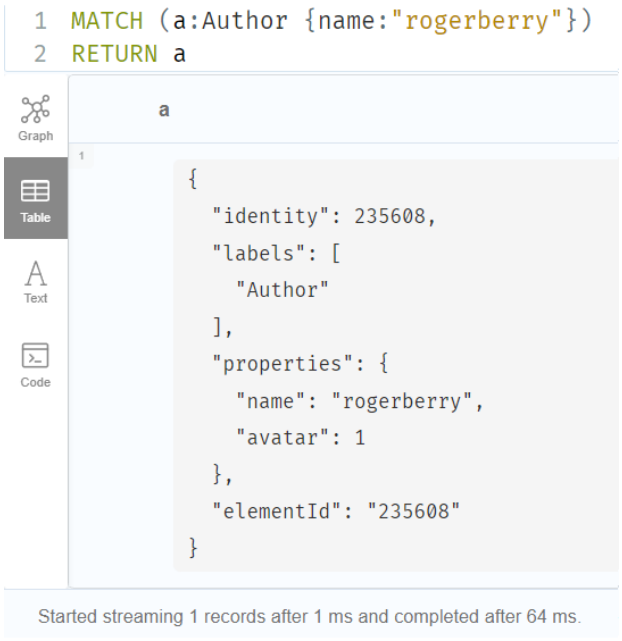
Method: AuthorNeoDao.getFollowing

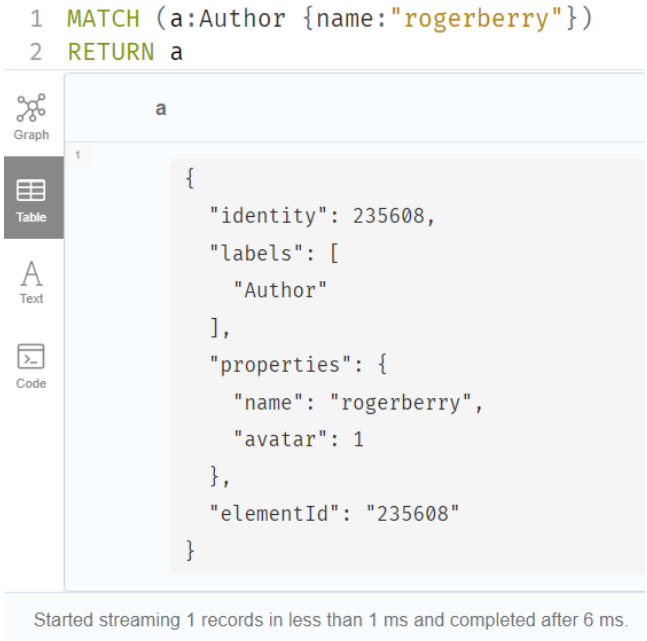
5.4 Indexes

Since in the application there are many reads on database that first look for a node with a specific name, it was considered appropriate to use an index on the Author.name parameter in order to speed up read operations.

```
$ CREATE INDEX ON :Author(name)
```

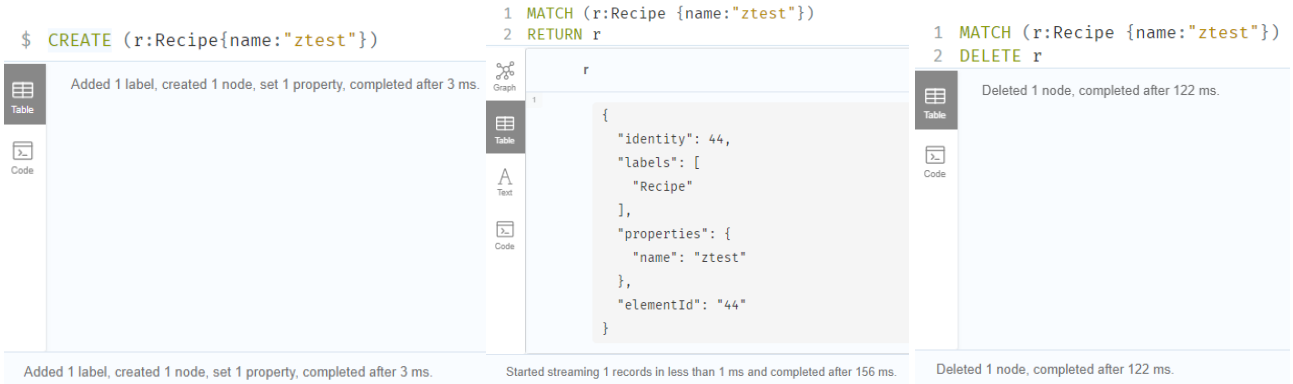
As expected, we can see a remarkable speed increase in finding a node with a specific name, with a tenfold decrease in search time.





Another useful index could be the one on the name of the Recipes. It would speed up the operation of adding the "WRITE" relation. However, it could delay the actual creation of the node. Quantifying the execution times with some tests, it is still convenient to use the index. It also speeds up the delete operation.

Before Recipe(name) index:



After Recipe(name) index:

```
$ CREATE INDEX ON :Recipe(name)
```

6 ADDITIONAL IMPLEMENTATIONS AND DETAILS

6.1 Sharding

As was previously discussed, among the non-functional requirements the availability is critical to the correct functionality of the application. Therefore, a possible sharding of the data was considered. Here is how it could be done for the collections:

- recipe and reportedRecipe: we can exploit respectively the field “datePublished” which represents the creation date of the recipe and “dateReporting” that represents the date when the recipe was reported. This way we can implement sharding by **Range, more specifically the documents can be partitioned into annual shards.**
- author and moderator: These collections contain only account information, meaning no dates or categories, therefore Range sharding cannot be used, same thing for List sharding. The only possible way of sharding remains Hashing, more specifically a shard key based on the “_id” field and the use of a Hash table.

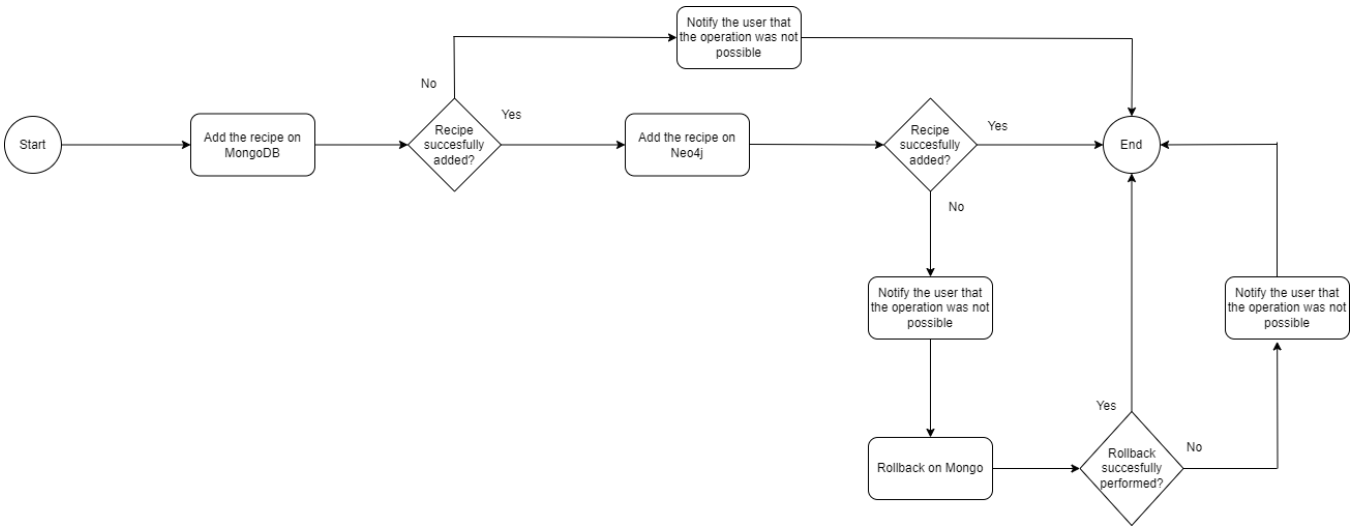
The implementation of this kind of sharding would slow down analytics execution. An excellent solution would be to reduce the range of recipes analyzed to those published in the same year. Maybe being able to let the user choose the year or using the current year.

The combination of Sharding with Replicas also could guarantee fast responses by the system and high availability, which were the features we were looking for.

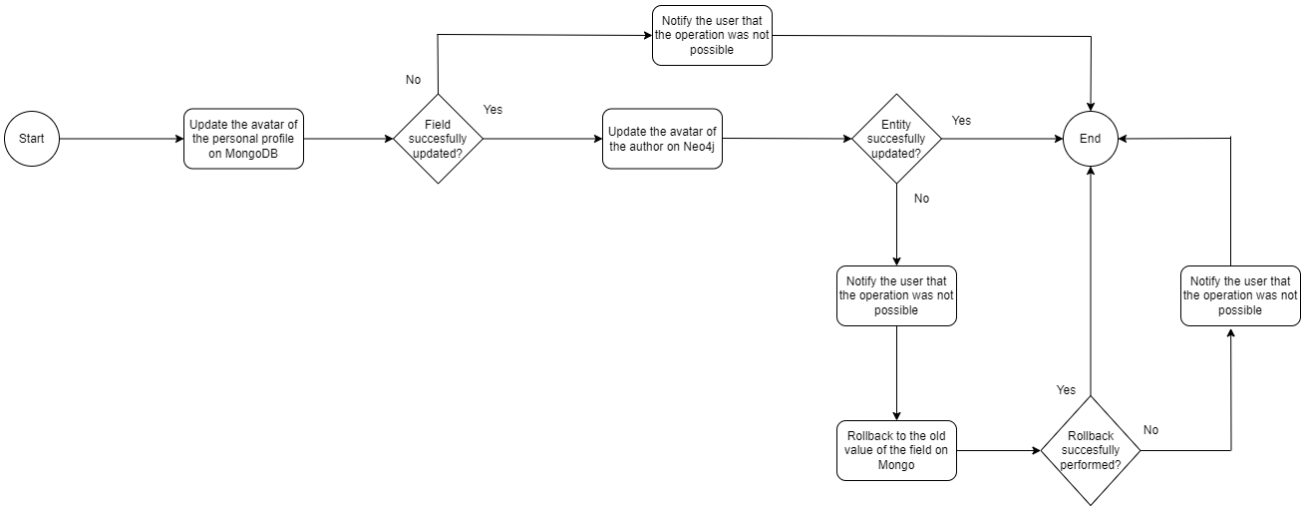
6.2 Cross-Database Consistency

Since author and recipe are both collections in MongoDB and entities in Neo4j, some operations regarding them need to be carried out in both databases in an atomic way, meaning that a failure of the operation in one of the 2 databases could lead to a necessary roll-back if the operation was successful in the other database.

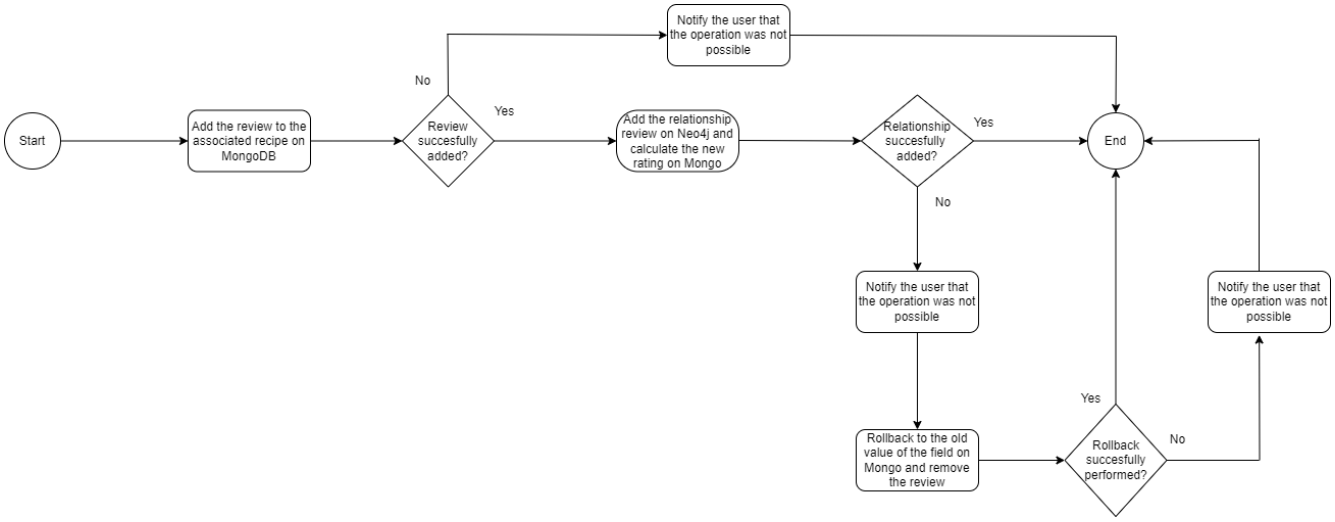
6.2.1 Add an entity (Author or Recipe)



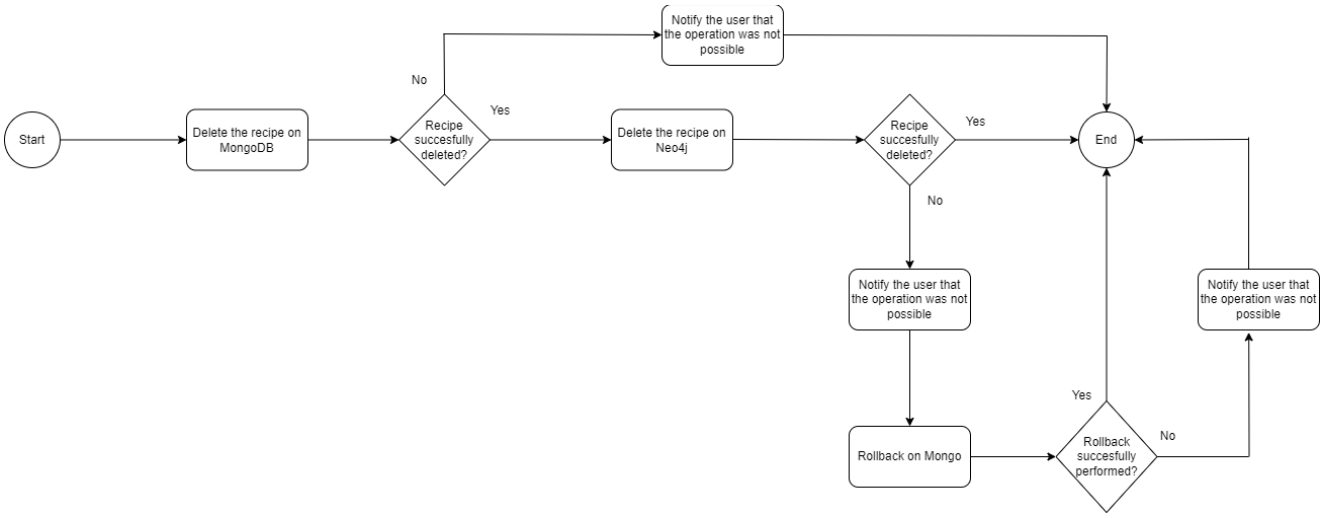
6.2.2 Update the Avatar



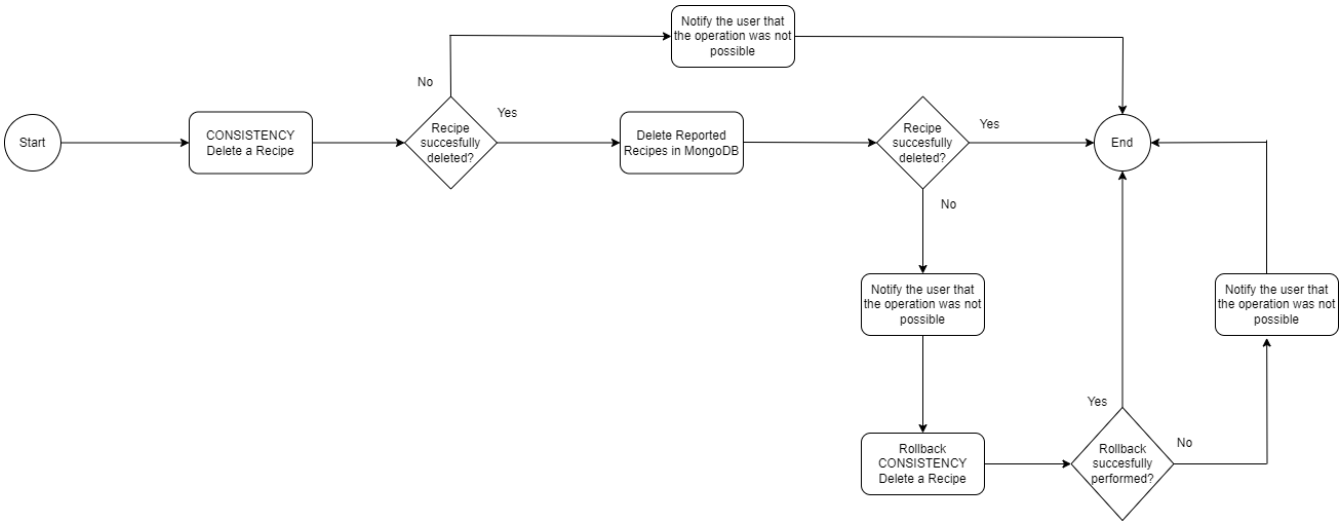
6.2.3 Add a Review to a Recipe



6.2.4 Delete a Recipe

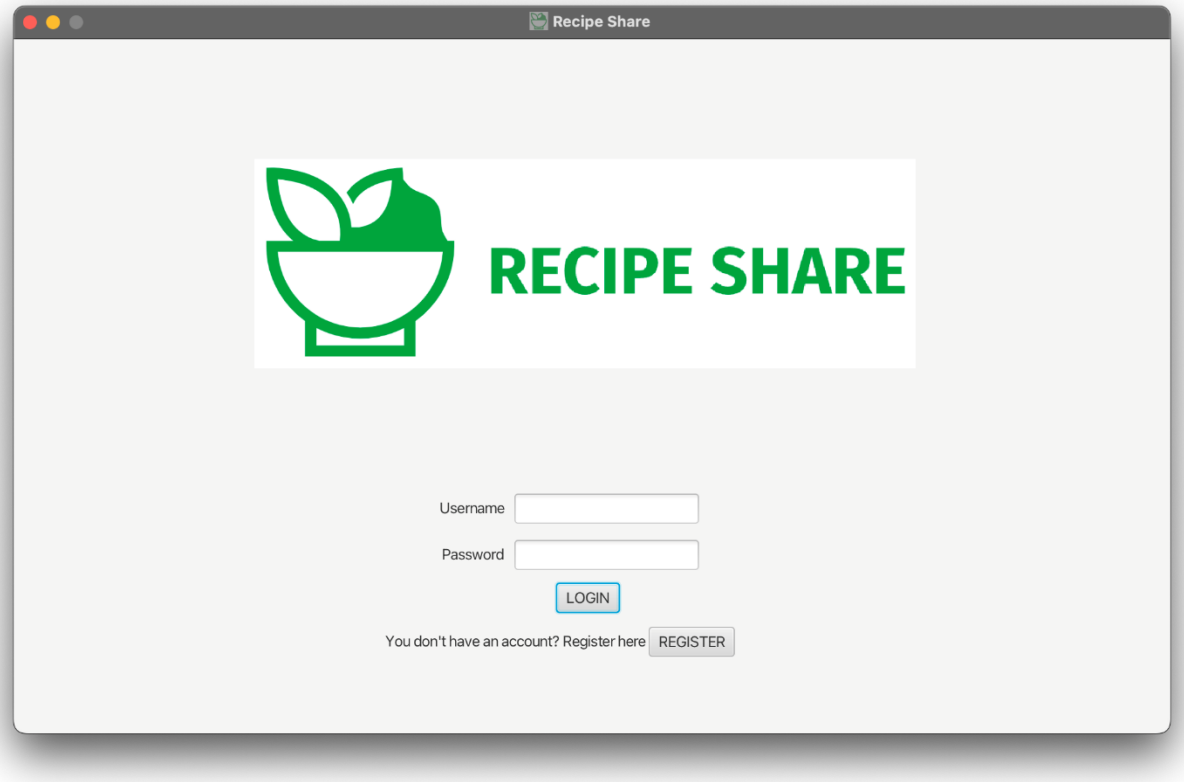


6.2.5 Reject a Reported Recipe



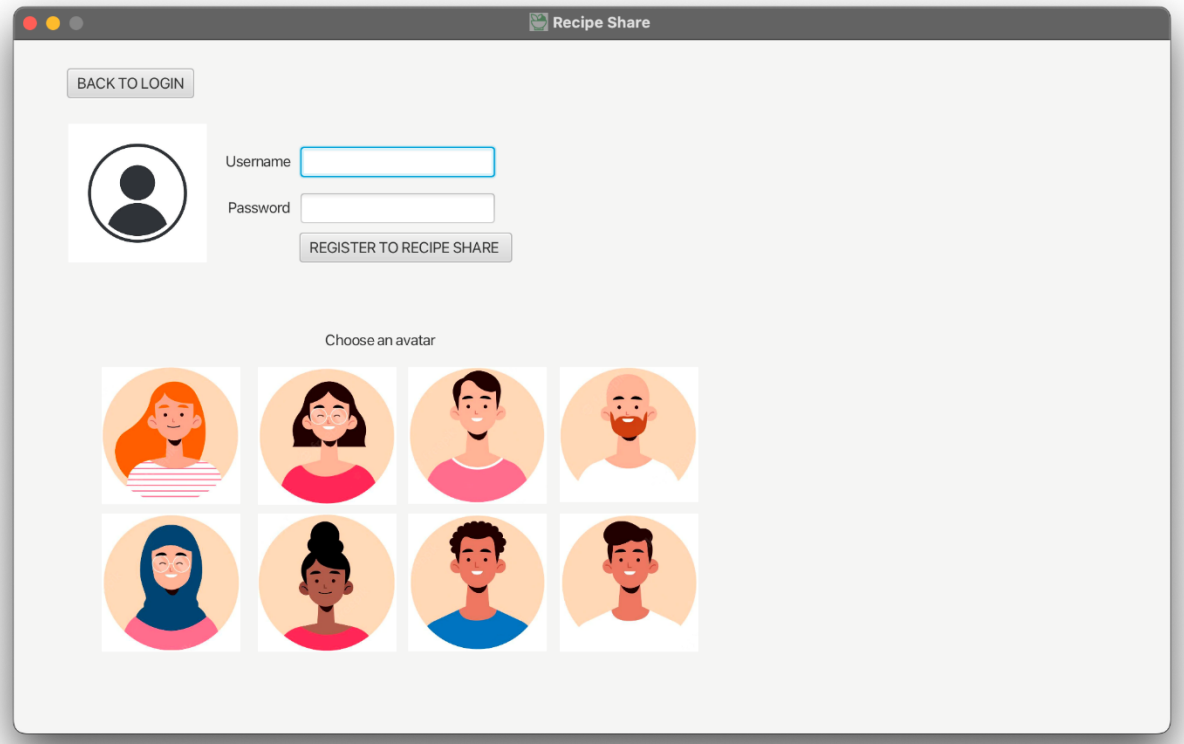
7 USER MANUAL

When the application is started, the home page is shown to the user:



If the user already has an account in Recipe Share, he/she can login to the service by inserting his/her credentials in the “Username” and “Password” fields and then by clicking on the “Login” button. If the login fails, the user will be notified of the error.

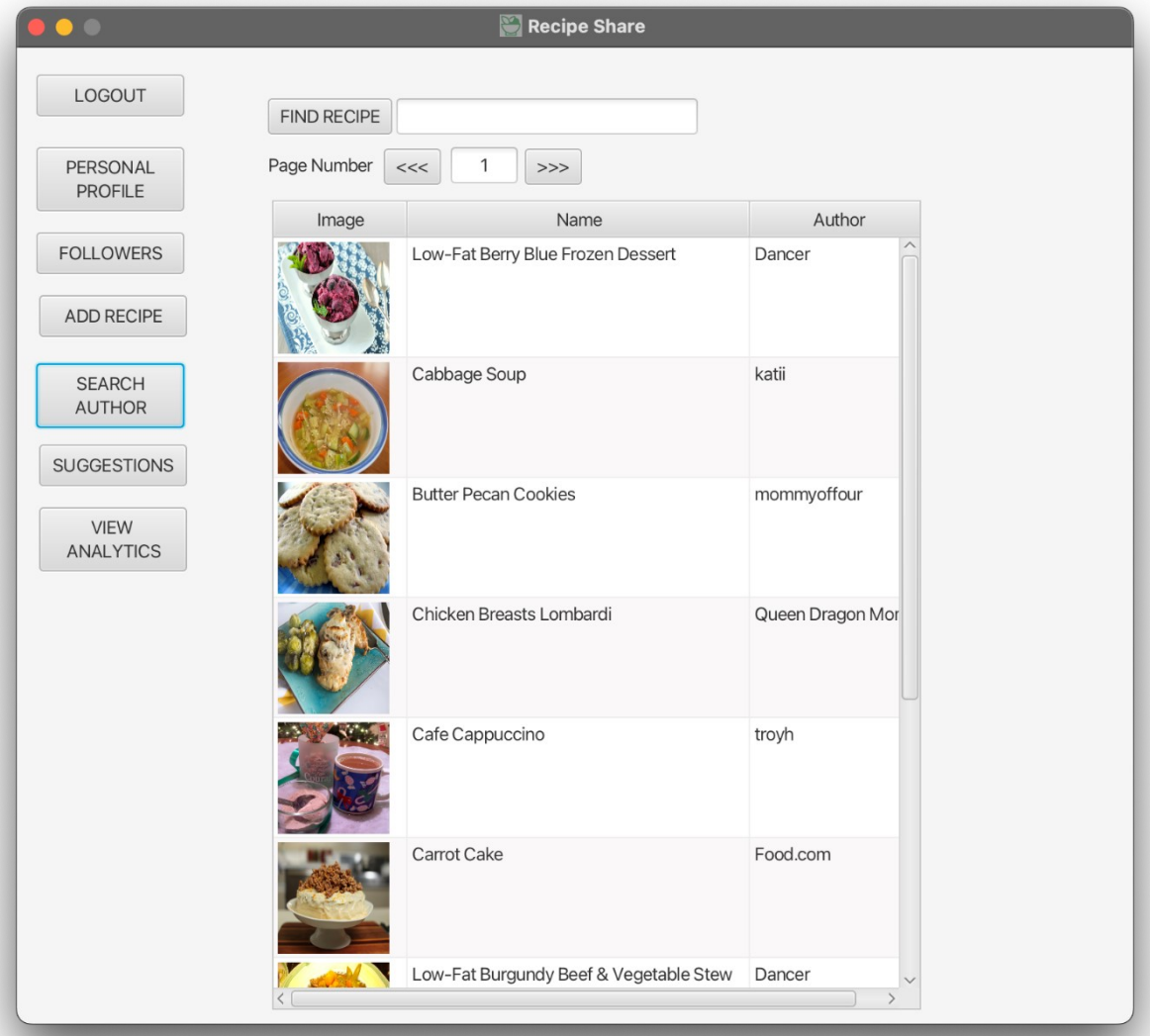
If the user doesn’t have an account, he/she can use the “Register” button to create one. In this case he/she will be taken to the following page:



Here the user can choose the username, the password and the picture he/she wants for his/her account: username and password have to written in their respective fields whilst for the picture the author has to simply click on the one he/she likes, in this case the image will appear on the top-left on the screen, more precisely in the image field under the “back to Login” button. Once everything has been chosen, the user can click on the “Register to Recipe Share” button to complete the registration. If the registration is successful, he/she can use “Back To Login” to get back to the home page, otherwise if the registration is unsuccessful it means that the username picked is already in use and has to be changed and the user will be notified of the error.

Once a user completes the login, he/she will be taken to 2 different pages depending if he/she is an author or a moderator. Firstly, we’ll show the page for the author:

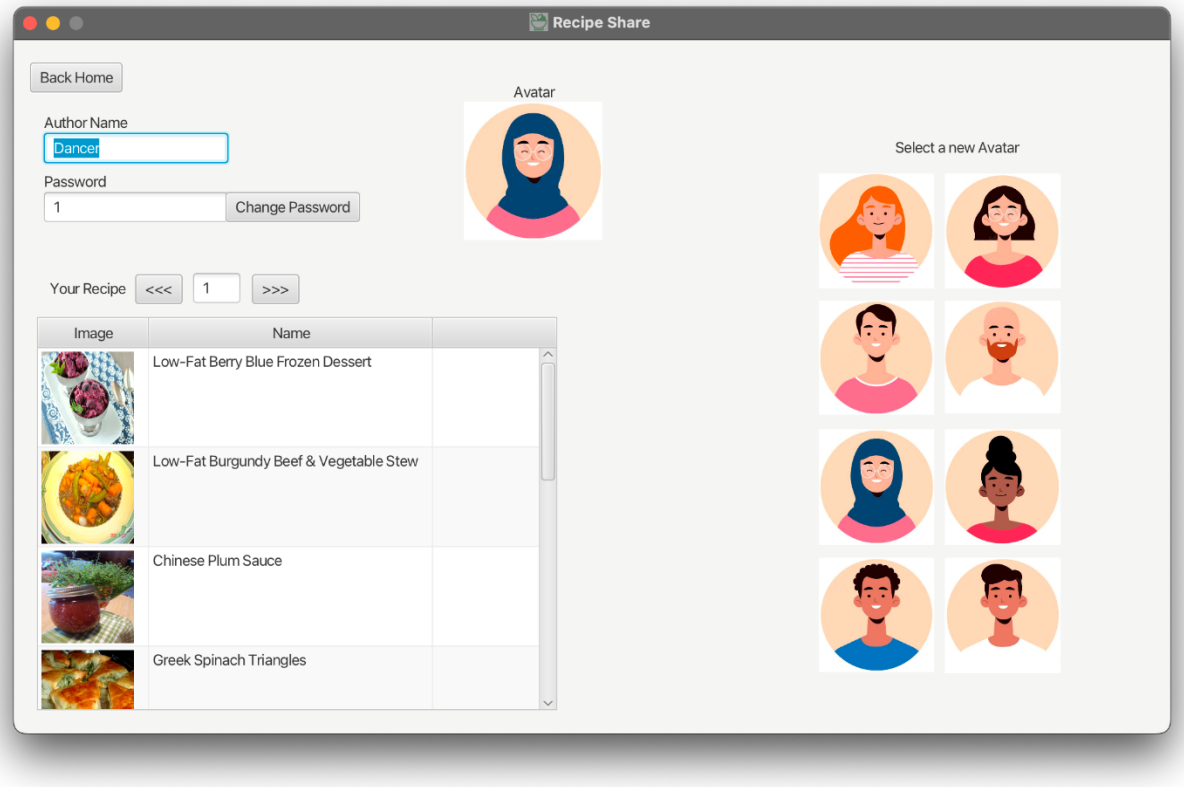
7.1 Author



Here the author is able to browse recipes by scrolling the table cursor, otherwise he/she can insert a string the field next to the "Find Recipe" button, this way the recipes that contain the string inserted in their name will be shown in the table. The "<<<" and the ">>>" buttons let the author browse the recipes by showing 10 recipes at a time in the table, therefore implementing the pagination. Regarding the other buttons:

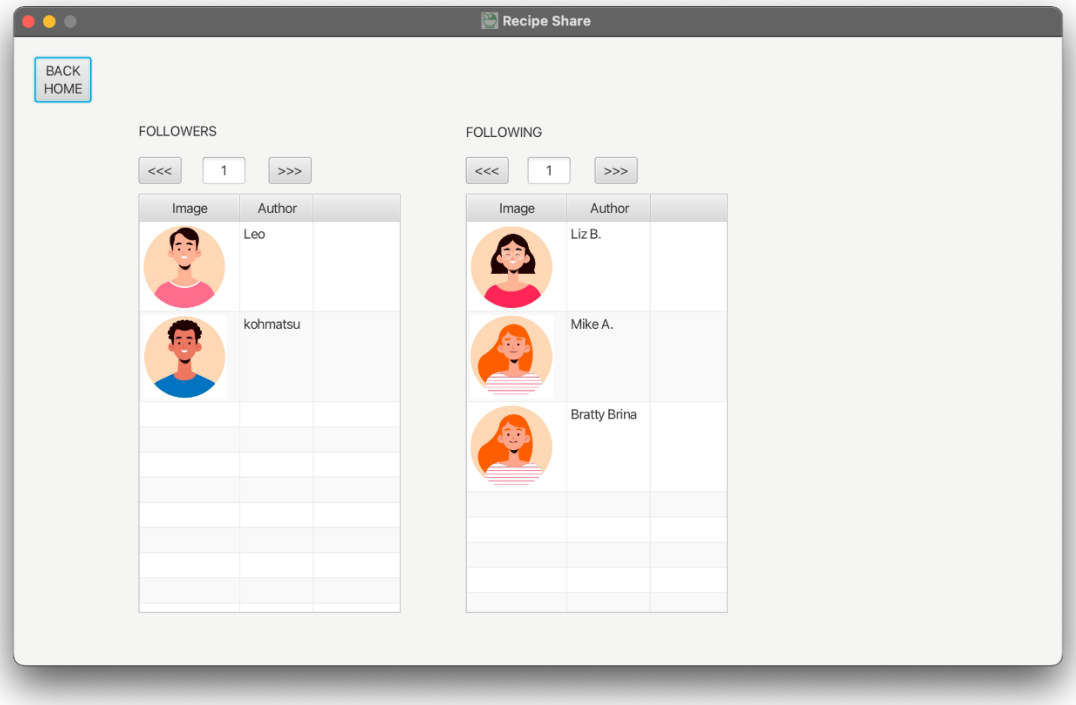
1. **Logout**: takes the author to the home page.
2. **Personal** Profile: takes the author to Personal Profile page.
3. **Followers**: takes the author to the Follow page.
4. **Add Recipe**: takes the author to the Add Recipe page.
5. **Search Author**: takes the author to the Search Authors page.
6. **Suggestions**: takes the author to the Suggestions page.
7. **View Analytics**: takes the author to the Analytics page.

2) **PERSONAL PROFILE PAGE**



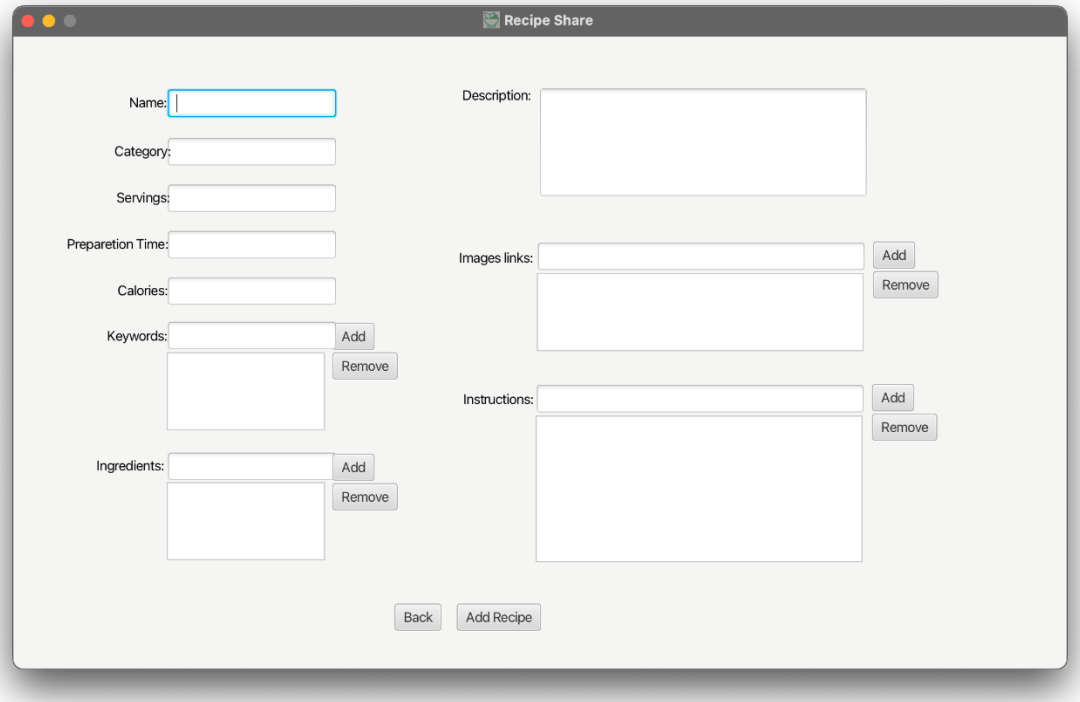
Here the author is able to update his/her account information, meaning that he/she can change his/her password by typing a new one in the field next to the “Change Password” button and then clicking on the button. The Avatar can be updated by simply clicking one of the 8 images available on the right of the screen. The table of the recipes written by the author is also available.

3) **FOLLOW PAGE**



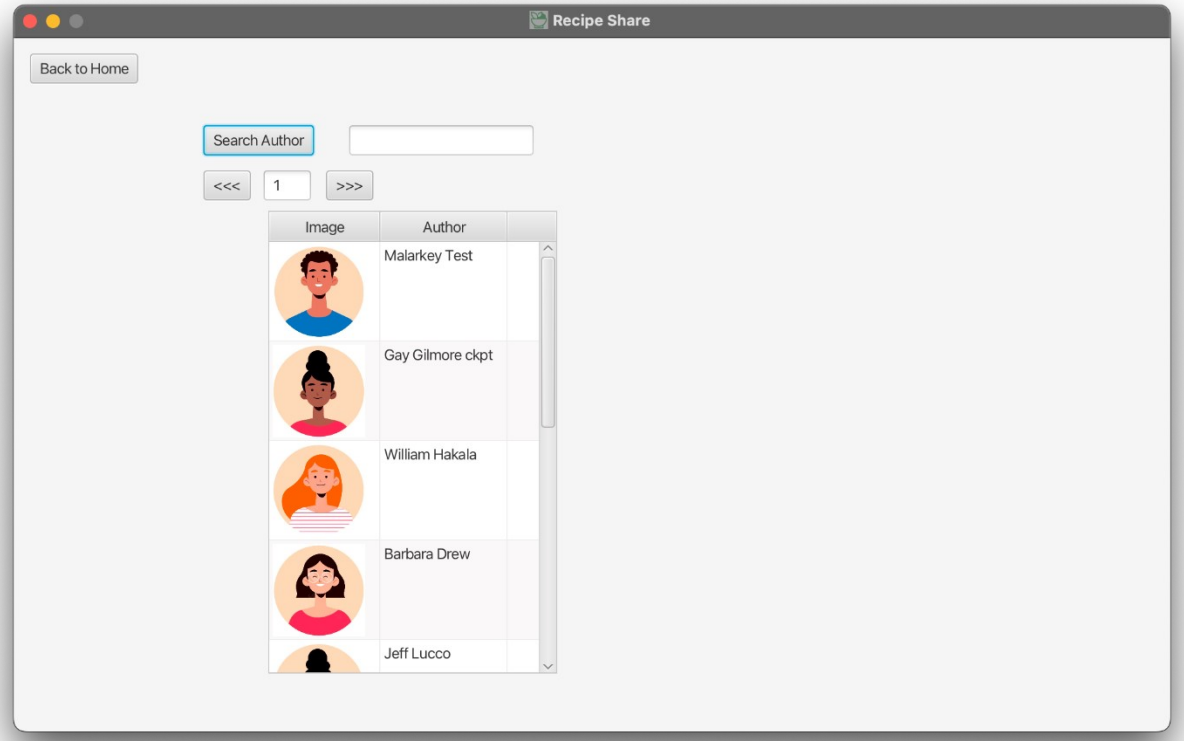
Here the author can visualize the table of the authors who follow him/her and the table of the authors he/she follows.

4) **ADD RECIPE PAGE**



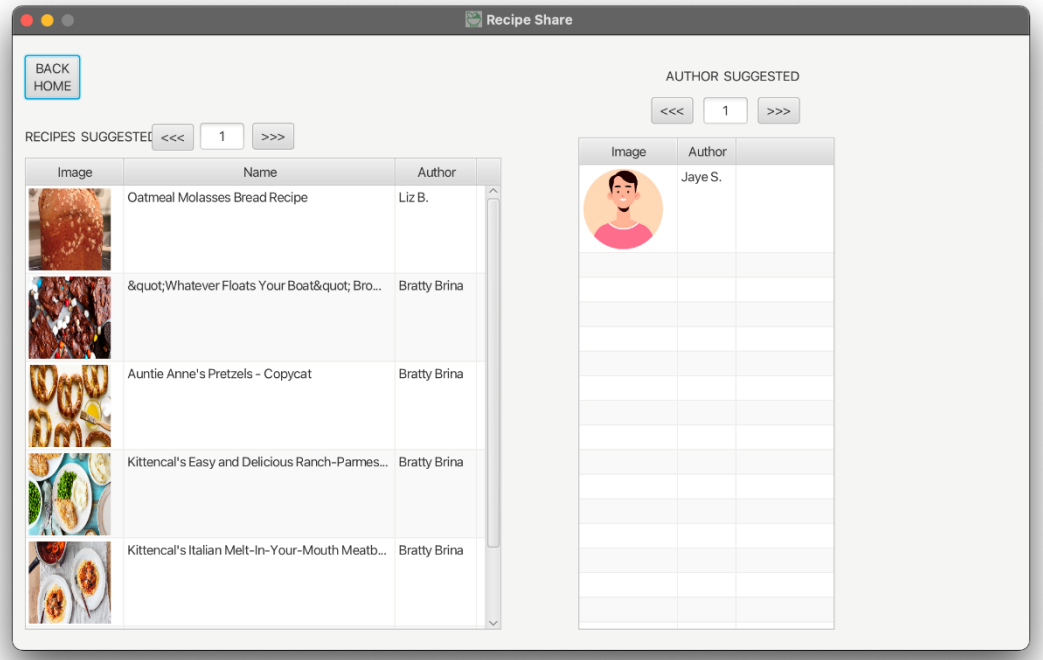
Here the author can insert all the information about the recipe he/she wants to add (the “Name” and “Image Links” fields are mandatory). Once the information has been inserted, the author can click on the “Add Recipe” button to add the recipe to the social network. If one or more mandatory fields are not inserted the author won’t be able to add the recipe.

5) **SEARCH AUTHORS PAGE**



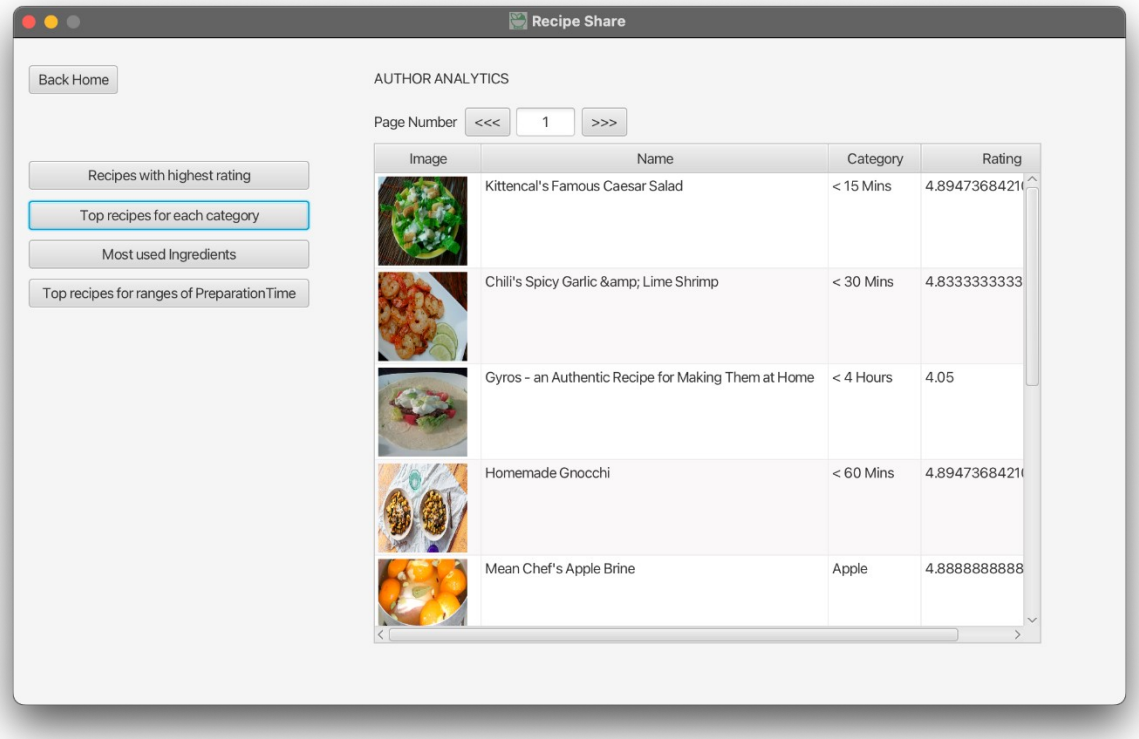
Here the author can browse the authors by scrolling the cursor of the table. The author can also type a string in the field next to the “Search Author” button to visualize in the table only the authors who contain that string in their name. The “<<<” and “>>>” buttons implement the pagination.

6) **SUGGESTIONS PAGE**



Here the author can visualize the suggestions: there are 2 tables, one that contains the suggested recipes and one that contains the suggested authors.

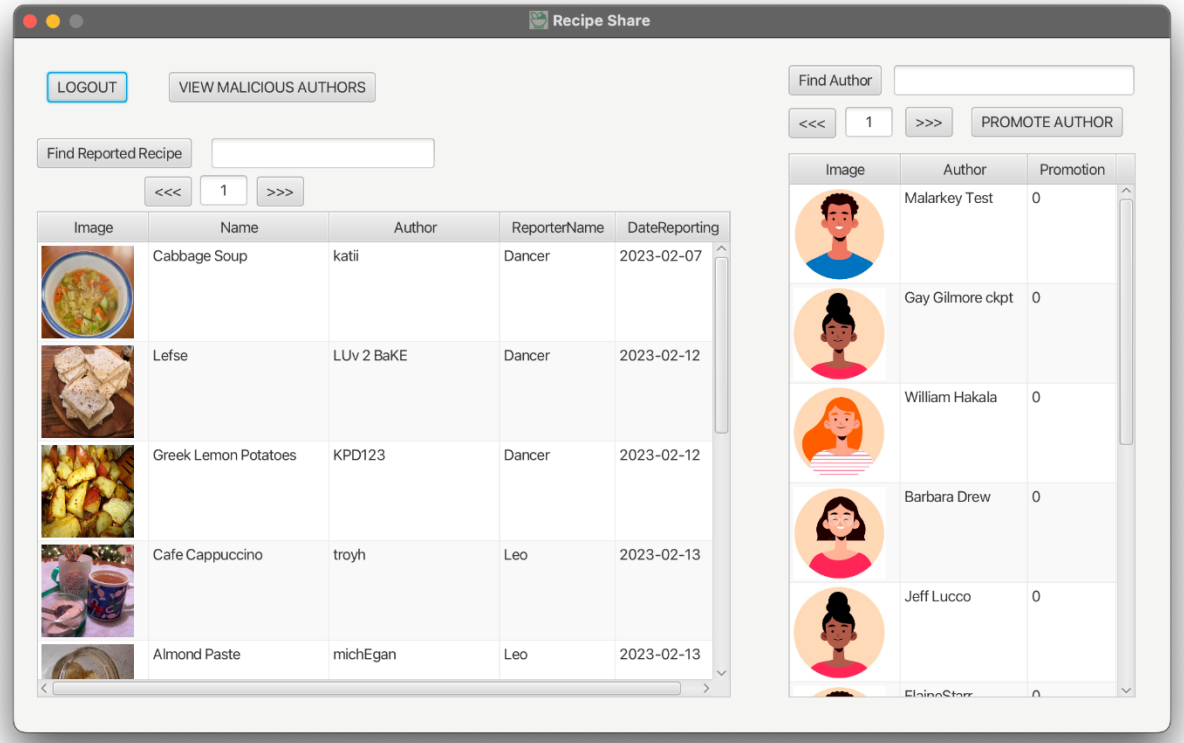
7) **ANALYTICS**



Here the author can visualize the results of each analytic: the results will be shown as a table like in the figure above (in this case the “Top recipes for each category” analytic was selected).

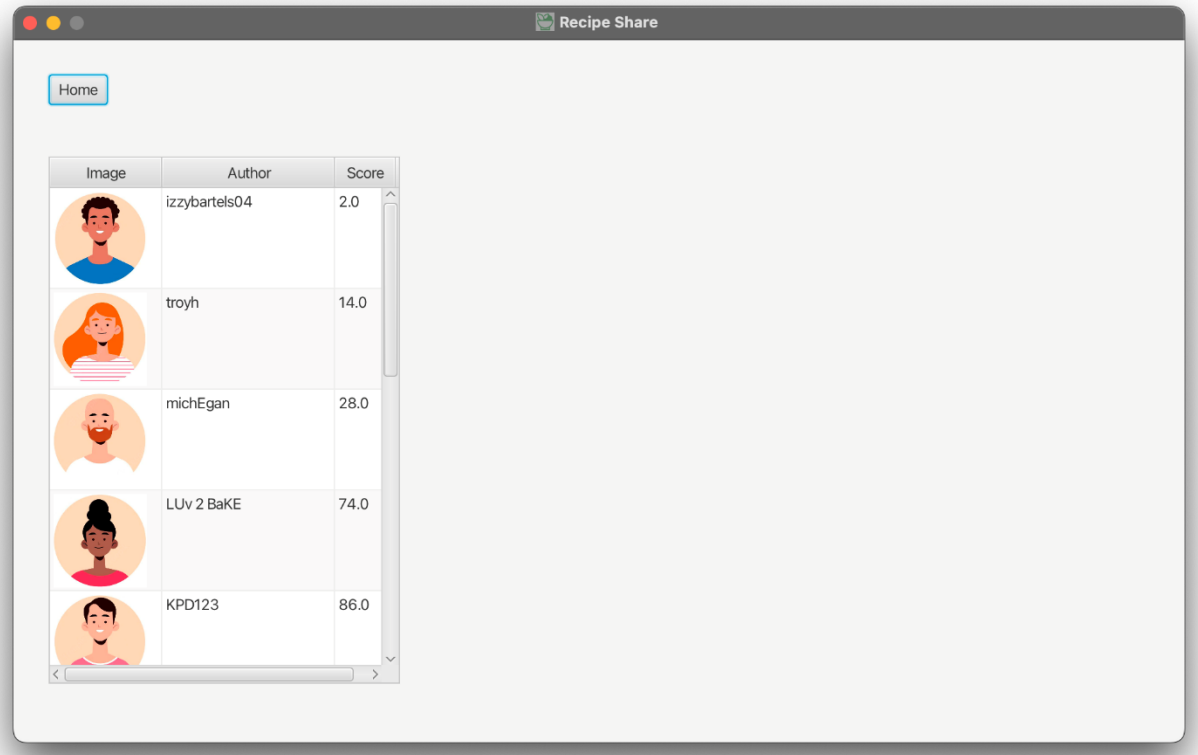
7.2 Moderator

Now let’s see which features are available as a moderator:



Here the moderator can browse the reported recipes and the authors. The moderator can browse both by also typing a string in the fields next to the “Find Reported Recipe” and “Find Author” buttons and then clicking on the buttons: this way in the tables only the recipes and reported recipes that contain the respective string in the name. The moderator can also offer a promotion to role of the moderator to an author by clicking on the table cell of the “Promotion” column next to the author’s name and then clicking on the “Promote Author” button. The “<<<” and “>>>” buttons implement the pagination.

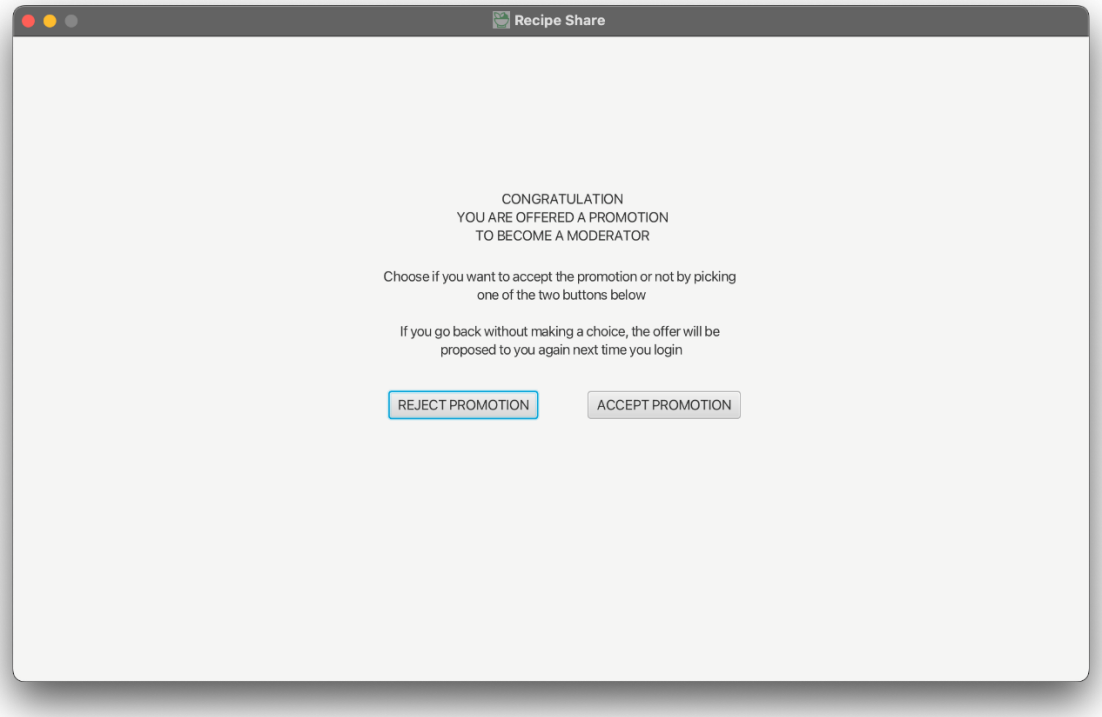
If the moderator clicks on the “View Malicious Authors” button, he/she will be taken on the analytic page for the moderator, which is shown below:



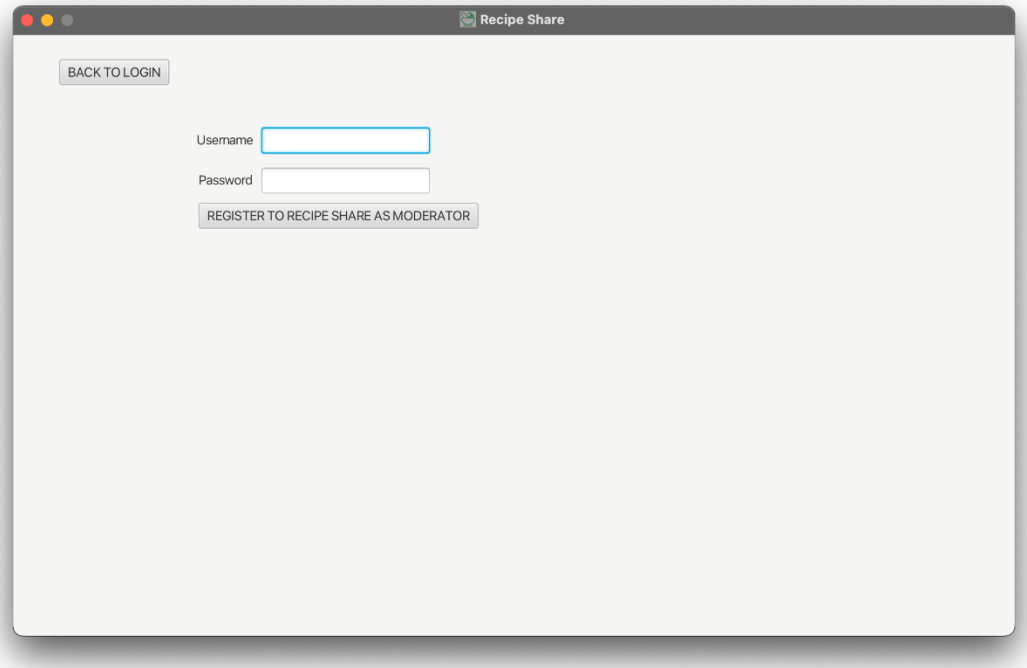
The analytic shows the authors with the worst ratio of recipe added to their recipes being reported.

PROMOTION

If an author is offered a promotion, next time he tries to login the application he/she will be taken to the following page:



If the author decides to reject the offer, he will be taken to the Author home page and nothing else happens. Otherwise if the author accepts the offer, he/she will be taken to the following page where he/she will be able to register a new account as a moderator:

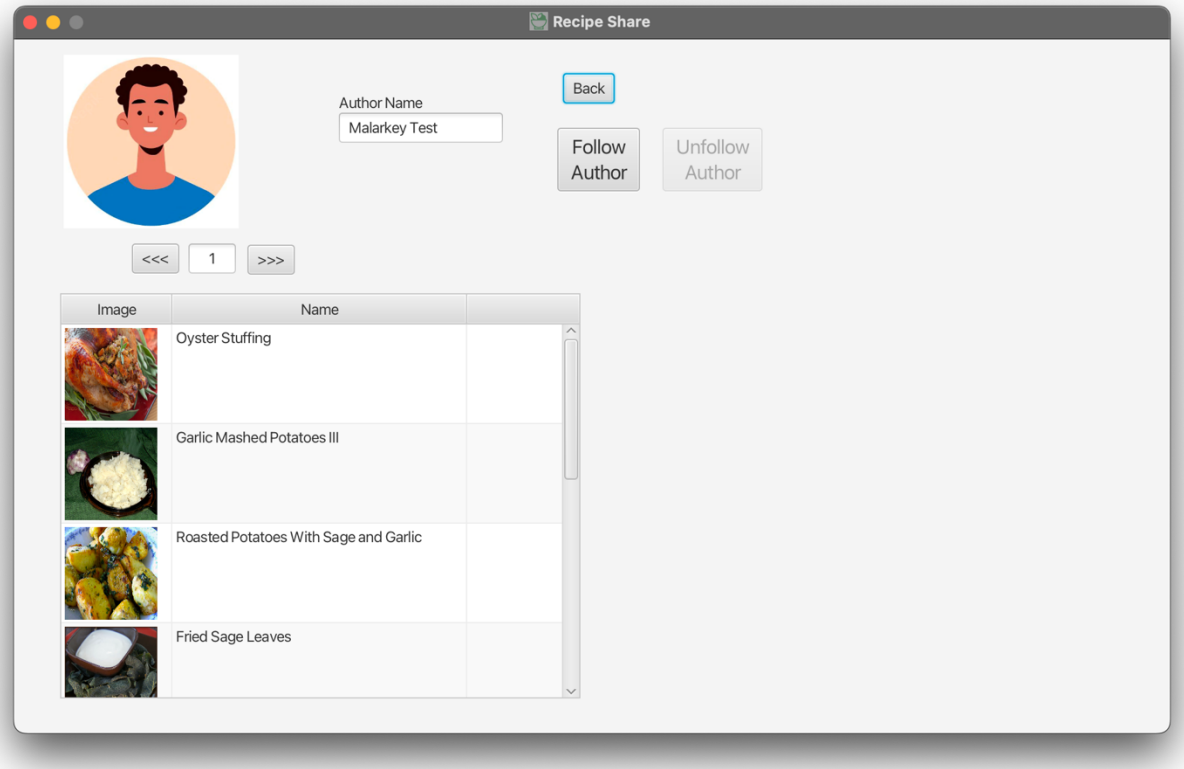


EVENT FOR TABLES

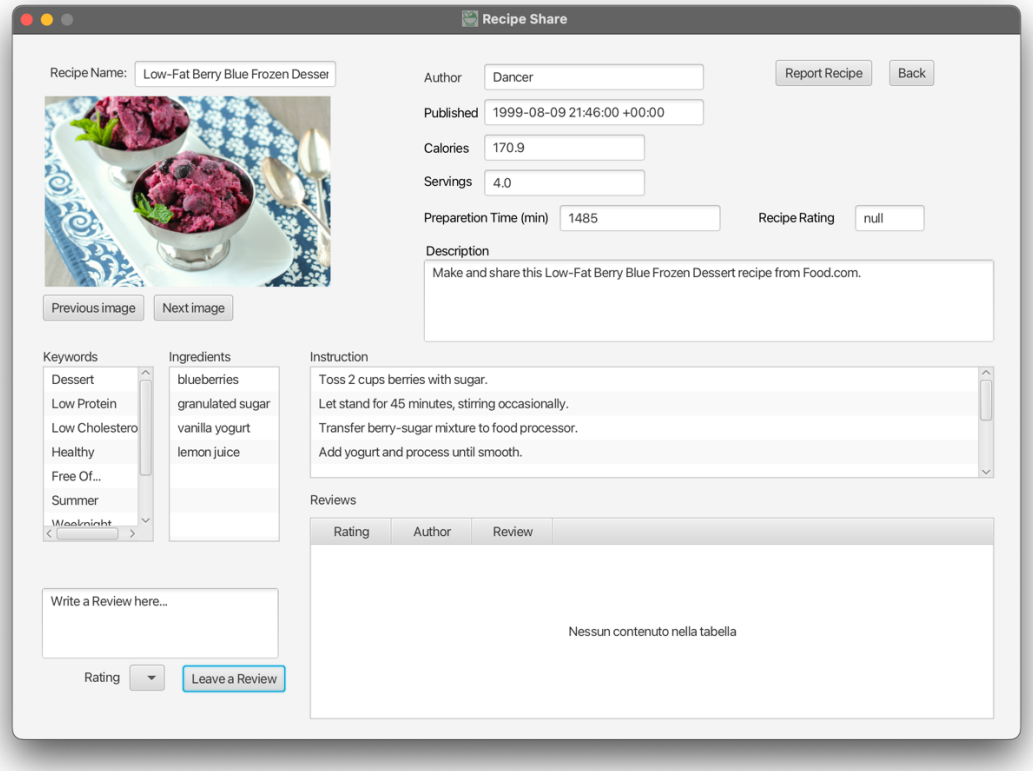
The following images will show features available both for the author and the moderator: as shown by the previous images, in the application the information about recipes, authors and reported recipes are visualized as tables, here are the events associated with the tables:

1) **Authors** table: clicking on the cell that contains the name of the author will take the author/moderator to the page containing information about the author.
2) **Recipes** Table: clicking on the cell that contains the name of the recipe will take the author/moderator to the page containing information about the recipe.
3) **Reported Recipes** table: clicking on the cell that contains the name of the reported recipe will take the author/moderator to the page containing information about the reported recipe.

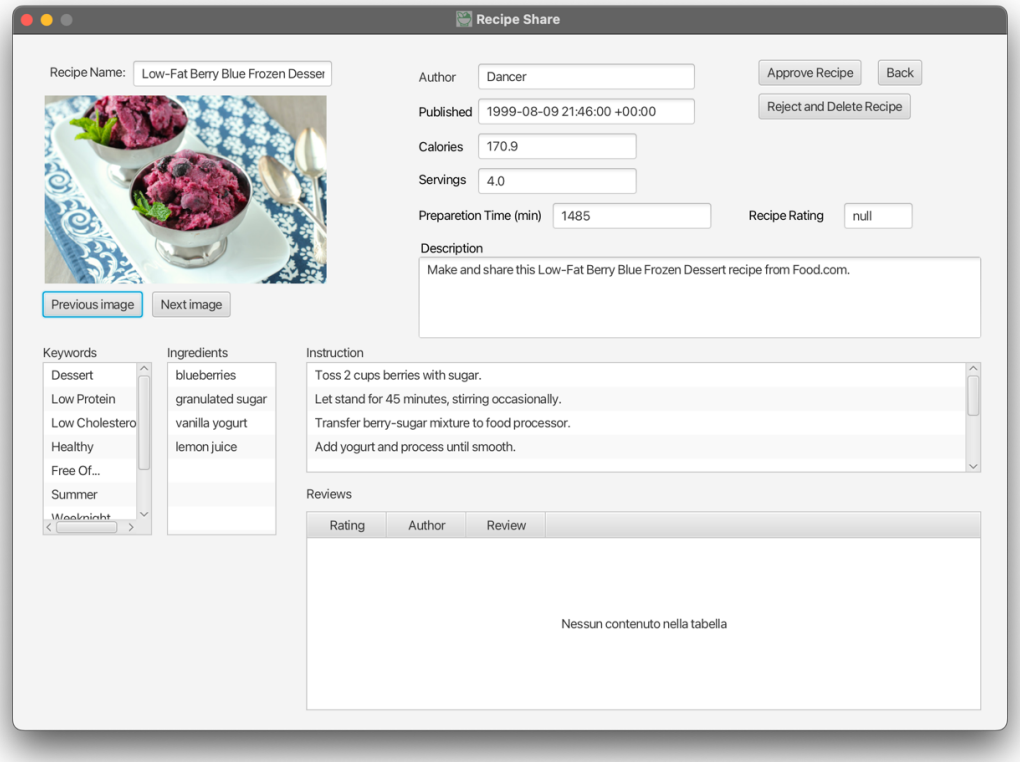
1)



2)


Recipe Share
Recipe Name:
Low-Fat Berry Blue Frozen Desser
Author
Dancer
Report Recipe
Back
Published
1999-08-09 21:46:00 +00:00
Calories
170.9
Servings
4.0
Preparetion Time (min)
1485
Recipe Rating
null
Description
Make and share this Low-Fat Berry Blue Frozen Dessert recipe from Food.com.
Previous image
Next image
Keywords
Dessert
Low Protein
Low Cholestero
Healthy
Free Of...
Summer
Ingredients
blueberries
granulated sugar
vanilla yogurt
lemon juice
Instruction
Toss 2 cups berries with sugar.
Let stand for 45 minutes, stirring occasionally.
Transfer berry-sugar mixture to food processor.
Add yogurt and process until smooth.
Reviews
Rating
Author
Review
Nessun contenuto nella tabella
Write a Review here...
Rating
Leave a Review

3)


Recipe Share
Recipe Name:
Low-Fat Berry Blue Frozen Desser
Author
Dancer
Approve Recipe
Back
Reject and Delete Recipe
Published
1999-08-09 21:46:00 +00:00
Calories
170.9
Servings
4.0
Preparetion Time (min)
1485
Recipe Rating
null
Description
Make and share this Low-Fat Berry Blue Frozen Dessert recipe from Food.com.
Previous image
Next image
Keywords
Dessert
Low Protein
Low Cholestero
Healthy
Free Of...
Summer
Ingredients
blueberries
granulated sugar
vanilla yogurt
lemon juice
Instruction
Toss 2 cups berries with sugar.
Let stand for 45 minutes, stirring occasionally.
Transfer berry-sugar mixture to food processor.
Add yogurt and process until smooth.
Reviews
Rating
Author
Review
Nessun contenuto nella tabella