**E- Recipe Platform**

Project Report

***Student Name***

*Dumitru Rares Bunea*

*Vladimir Rotaru*

***Student No***

*266983*

*266914*

*17626 characters*

*ICT Engineering*

***December 2, 2019***

Table of Contents

[List of figures and tables 3](#_Toc26146330)

[Abstract 4](#_Toc26146331)

[1. Introduction 5](#_Toc26146332)

[2. Requirements 6](#_Toc26146333)

[2.1 Functional Requirements 6](#_Toc26146334)

[2.2 Non-Functional Requirements 7](#_Toc26146335)

[3 Analysis 8](#_Toc26146336)

[3.1 Business Model 8](#_Toc26146337)

[4 Design 11](#_Toc26146338)

[4.1 Data Flow 11](#_Toc26146339)

[4.2 Design Patterns 12](#_Toc26146340)

[4.3 Class Diagram 13](#_Toc26146341)

[5 Implementation 14](#_Toc26146342)

[5.1 Persistence Tier 14](#_Toc26146343)

[5.2 Business Logic Tier 20](#_Toc26146344)

[5.3 Presentation Tier 21](#_Toc26146345)

[6 Testing 25](#_Toc26146346)

[6.1 Persistence Tier 25](#_Toc26146347)

[6.2 Business Logic Tier 27](#_Toc26146348)

[6.3 Presentation Tier 28](#_Toc26146349)

[7 Results and Discussions 29](#_Toc26146350)

[8 Conclusion 30](#_Toc26146351)

[9 Project Future 31](#_Toc26146352)

[Sources of information 32](#_Toc26146353)

[Appendices 33](#_Toc26146354)

## List of figures and tables

[Figure 1 - Use Case Diagram 8](#_Toc26146096)

[Figure 2 - Domain Model 9](#_Toc26146097)

[Figure 3 - Conceptual Diagram 9](#_Toc26146098)

[Figure 4 - System Sequence Diagram 10](#_Toc26146099)

[Figure 5 - System Sequence Diagram 11](#_Toc26146100)

[Figure 6 – Architecture Pattern 12](#_Toc26146101)

[Figure 7 - Model 14](#_Toc26146102)

[Figure 8 -Db Sets 14](#_Toc26146103)

[Figure 9 - Model Builder 15](#_Toc26146104)

[Figure 10 - Database Diagram 16](#_Toc26146105)

[Figure 11 -Seeding 17](#_Toc26146106)

[Figure 12 - IRepository 17](#_Toc26146107)

[Figure 13 - Repository 18](#_Toc26146108)

[Figure 14 - Referencing In Startup 18](#_Toc26146109)

[Figure 15 - DTO 19](#_Toc26146110)

[Figure 16 - Controller 19](#_Toc26146111)

[Figure 17 - Home Page 21](#_Toc26146112)

[Figure 18 -DTO 22](#_Toc26146113)

[Figure 19 - Repository 22](#_Toc26146114)

[Figure 20 - Startup 23](#_Toc26146115)

[Figure 21 - ViewModel 23](#_Toc26146116)

[Figure 22 - Response Types 25](#_Toc26146117)

[Figure 23 - Test Suite 25](#_Toc26146118)

[Figure 24 - Layout 29](#_Toc26146119)

## Abstract

Food is what all living creatures need in order to survive. It comes in different forms and aspects, for each individual and plant. Food might be sun energy, a plant, water or meat, it is always different, but its intent is to furnish energy and nutrition.

For this semester project a new electronic yellow book of recipes was proposed to be implemented. The name of this system followed to be “ERecipe”. The aim of the project was to create a platform where people all around the world could post recipes and also inspire themselves by researching other recipes posted.

The main technical challenges were to build a three-tier architecture system, using Java and C#. The system should consume and expose a web service and have a GUI for each client. The most important aspects of system development such as: analysis, design, implementation and testing; are described in this report

## 1. Introduction

Since ancient times, people as well as animals were hunting for food. It was one of the basic instincts of all living creatures. Throughout years, people have adapted themselves to their environments and hunting has been replaced with agriculture, pastoralism and farming.

Food has always served as a carrier of culture in human society, people were inventing different dishes that were meant to mirror their affiliation to a specific culture, country or region. They were keeping the instructions and ingredients needed to make the dish, also known as recipe, usually in written form, for future generations. It was their treasures and no other individuals could know their secrets of making a particular dish, except their family members.

As the times have changed, people started to share their recipes with the world as a way of promoting their country and culture. Also, traveling to other countries have enhanced the spirit of trying some new dishes. As an example, is famous pizza and croissants have spread the world and now there is no person who doesn’t know these amazing dishes.

From all said above, has come the idea of making an E-Recipe platform available for everyone who wants to share a recipe and inspire themselves with other recipes. The system will make it possible for the user to log in, then, the user can create a new recipe specifying a name, a description, ingredients, steps, author and country from where the recipe has come from.

## 2. Requirements

During the inception phase, some core requirements has been determined in order to create the project. The set of functional and non-functional requirements have settled the system functionalities and features. Both types of requirements will be covered in the upcoming sections.

### 2.1 Functional Requirements

Functional requirements were made in a table containing the name of the feature and the user story corresponding to the name.

Table 1: Functional requirements

|  |  |
| --- | --- |
| NAME | User story |
| USER REGISTRATION | As a user I want to be able to create an account so I can start using the system |
| USER LOG-IN | As a user I want to log in order to have my information personalized |
| USER INFO UPDATE | As a user I want to update my personal information |
| RECIPES OVERVIEW | As a user I want to see recipes posted by other people |
| RECIPE CREATION | As a user I want to be able to create a new recipe |
| RECIPE UPDATE | As a user I want to be able to update my recipe |
| RECIPE DELETION | As a user I want to delete my recipe |
| RECIPE POSTING | As a user I want to post my recipe |
| RECIPE REVIEW | As a user I want to make review to the recipes |
| RECIPE SEARCH | As a user I want to search for recipes |
| RECIPE CATEGORIZING | As a user I want to have categories of recipes |

### 2.2 Non-Functional Requirements

The non-functional requirements are defined below in a table, having an ID for each requirement as well as the description of the requirement.

Table 2: Non-functional requirements

|  |  |
| --- | --- |
| ID | NON-FUNACTIONAL REQUIREMENT |
| 1 | The system must be heterogeneous |
| 2 | The system must be distributed |
| 3 | The system must be a three-tier architecture |
| 4 | The system must consume and expose a web service |
| 5 | The system must use a GUI for each client |
| 6 | The system should implement security features |
| 7 | The system should be maintainable |
| 8 | The system should have different layers of access |

## 3 Analysis

### 3.1 Business Model

During the phase of analysis, based on the user stories outlined in section 2.1 – Functional Requirements, one actor has been identified – the **USER**. The actions that the user need to be able to perform are described below in a Use Case Diagram.

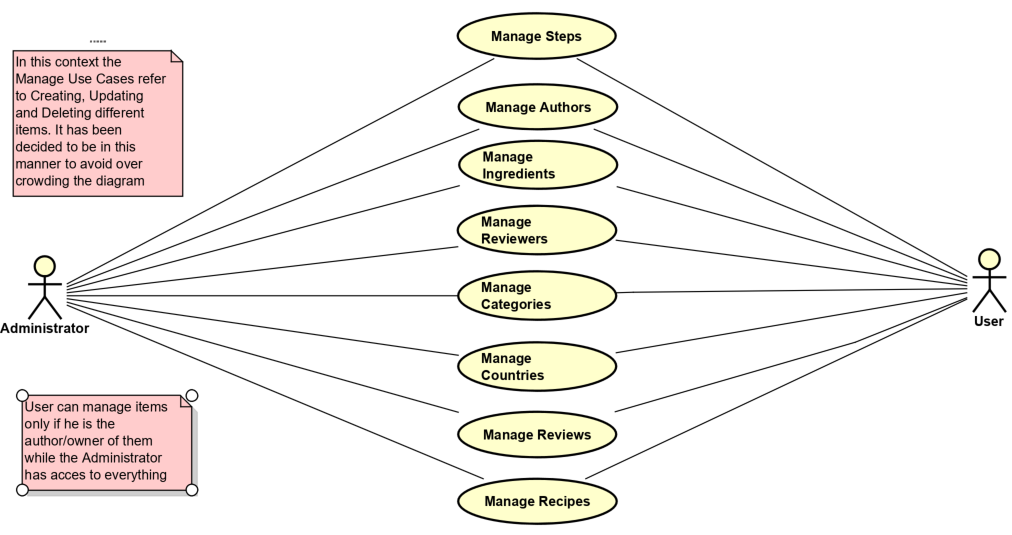


Figure 1 - Use Case Diagram

As displayed in the above diagram, two type of users were designed to be implemented in out system. “User” impersonates a day-to-day person, with passion toward learning more about the culinary art, or just wanting to share his different already known recipes. On the other hand, the “Administrator” can be seen as a watching eye on the well-fare of the community.

The main difference between the two is that while the “User” can manage, which basically means creating, updating and deleting, only item that he has personally created/posted, the administrator can manage everything that would possibly exist in the UI.

The core classes of the system and the way they cooperate between one another are the actions the user can perform. The interaction is shown below in a Domain Model, containing the main objects and classes described earlier.

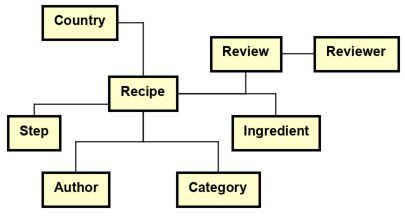


Figure 2 - Domain Model

As of the figure showed above, in the domain model, the main entities to be used in our system got a name and relations between them were made.

This Domain Model diagram was evolved, taking shape of a Conceptual Diagram to better explain the relationships between our entities.

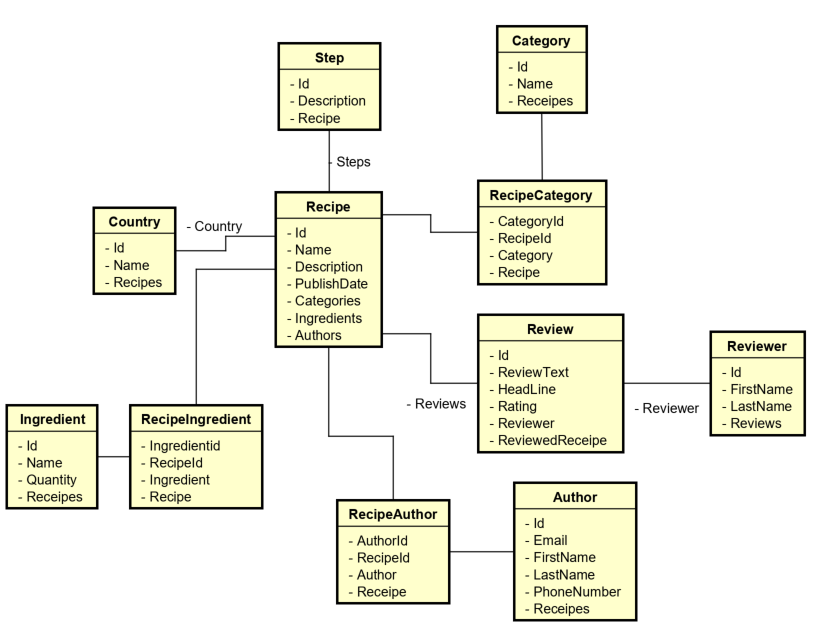


Figure 3 - Conceptual Diagram

With the help of the two previous presented diagrams, it is acknowledged that the entity that is the focus is the “Recipe”. In this business context, a recipe will contain as personal information a name, description and a publish date. With the assistance of the other differentiated entities, the recipe will also contain a country and lists of steps, ingredients and reviews. Due to multiplicity, three “different” entities were created to better form the many-to-many relations in our case: RecipeCategory, RecipeIngredient and RecipeAuthor. A simple explanation for this is that, for example, a recipe can have many authors, but at the same time an author can write more than one recipe. So, to better symbolize this relation, these joined classes were created to improve the navigation between our objects.

Also, to describe the basic flow between the user and the system, a small and simple sequence diagram was made, as shown in the figure below – System Sequence Diagram.

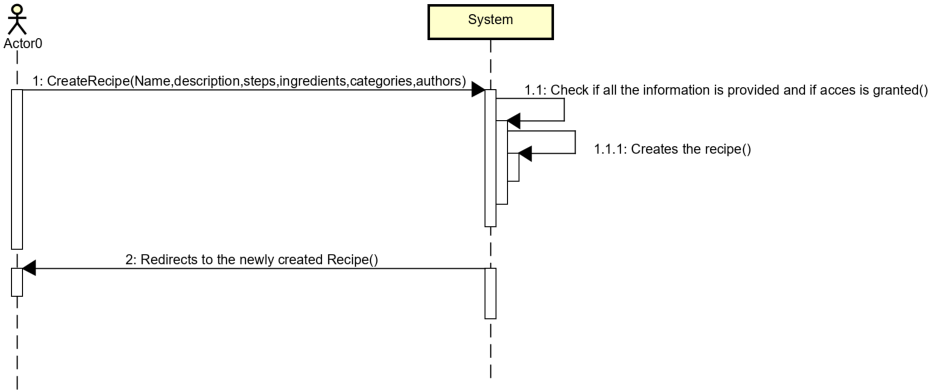


Figure 4 - System Sequence Diagram

The idea would be that every action that would be requested has to be validated. This means that the data has to be checked and users have to be verified if they are allowed to proceed with their actions.

## 4 Design

### 4.1 Data Flow

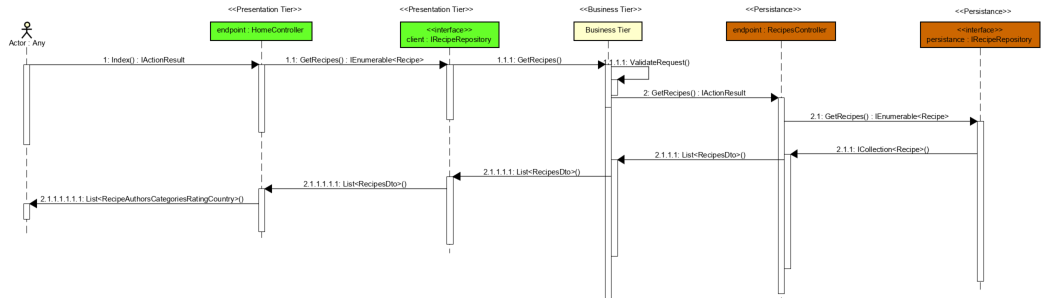


Figure 5 - System Sequence Diagram

This system sequence diagram is designed with the purpose of better understanding how data will be manipulated in our system. When a user makes a request form the Presentation Tier, the request is sent to the Business Logic tier which checks the validity of the data and if the user that made the request is authorized to procced. Then the request is retrieved trough the Persistence Tier of the system with different queries and it is sent back to the user. On the way back from the Persistence Tier to the user, data is altered in such a way that only the relevant information is displayed. Not relevant data, such as navigational proprieties, would not have any value in being presented or event being sent, since it would only consume resources without any true purpose.

### 4.2 Design Patterns

As the requirements suggest, the projects needs to implement a three-tier architecture. This type of architecture follows “Layered Pattern”, being one of the most used patterns in software architecture. The idea of it is to split the system into layers, where each of layers is responsible for particular features of the system and provides sevices to the other layers through endpoints.

A screenshot of a cell phone

Description automatically generated

Figure 6 – Architecture Pattern

As it is shown in the diagram above, each layer has a particular purpose. The **Presentation Layer** is responsible for graphical design and user experience with the application. Also, presentation layer is responsible for interaction between users and Business Logic. In this case, in our project, the layer is written in ASP.NET Core, with a use of modified MVC pattern. The MVC pattern is not a usual one as it has Data Transfer Objects, Controllers, View Models and Views folders, but does not have the actual Models folder. Except MVC pattern, the Presentation Layer uses Razor Pages because it gives more control over the actions happening on the web page. In order to communicate with business logic, JSON serialized objects would be used.

The **Business Logic Layer** should contain the primary logic of the system. It was decided that this layer will be written in Java. The business logic tier should receive appropriate data sent from presentation layer, or the user interface, and should pass it to persistence layer as JSON. Then the persistence layer should send appropriate data back to the business logic, also in a JSON format. The business logic should send data back to the presentation layer.

The **Persistence Layer** is the layer that manipulates the **Database** through a connection string, that represents an individual code, some properties. It is generated by ASP.NET Core when creating the database.

### 4.3 Class Diagram

*Due to the big size, class diagram will be appended*

A class diagram was designed for each tier. They were created to make future implementation easier, but also for a better understanding of the business context and the logic of the system that it is to be implemented.

The implementation of the model classes was “tested” using Doxygen, a software that turns C# code into xml, that Astah can understand and reproduce in diagrams. The implementations matched the initial design.

## 5 Implementation

### 5.1 Persistence Tier

The web server is implemented using C#, more exactly ASP.Net Core technologies.

It was decided that the way to address connecting, creating and populating a database would be using Entity’s Framework code first approach.

This meant that the first step was to create different model classes and specify certain constraints for them. This was done in order to be sure that the final result was equivalent to the initial expectations. An example for such a model could be the “Author”.

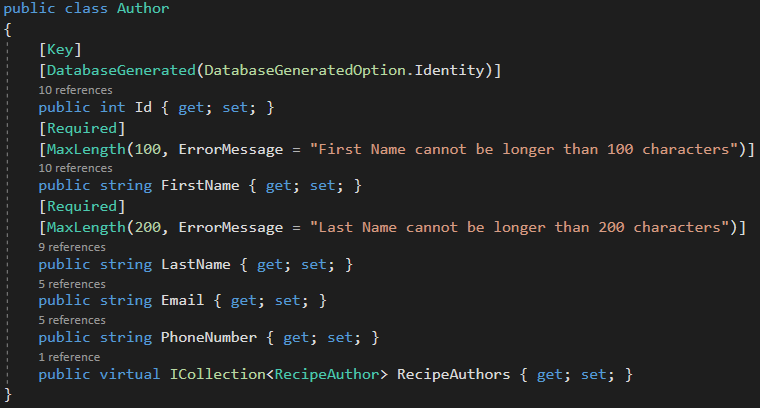


Figure 7 - Model

The RecipeDbContextClass was created. This class contain all the DbSets that will be exchanged with the database.

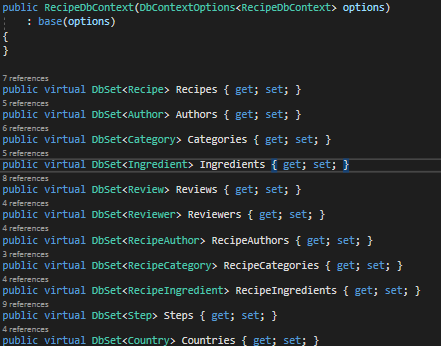


Figure 8 -Db Sets

After creating all of the required models the next step was to ensure that, once with the navigational proprieties, the proper constraints would be also set in the database, so a custom “OnModelCreating” method was created to ensure that the Model Builder that guides our migrations would create the proper connections. In this case connections mainly refer to primary and foreign keys in the database.This method resides in the RecipeDbContext class.



Figure 9 - Model Builder

With this step being done, the database was tested by checking every key that was created, but also by autogenerating the ER diagram of the database. This was done by changing the connection string from a MSSQL server to a local Db, enabling auto migrations and then opening the database in Microsoft SQL Server Management Studio 17 were the diagram could be finally checked.

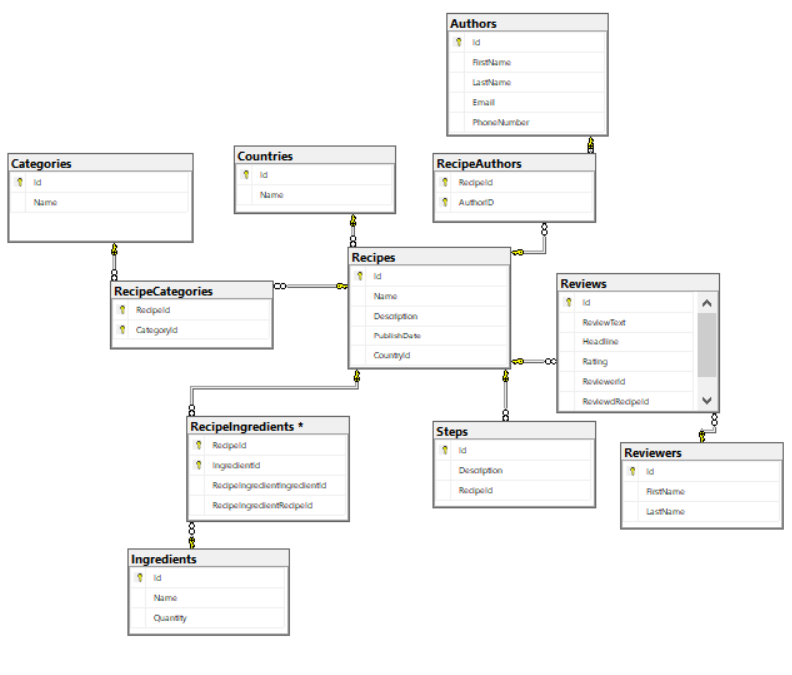


Figure 10 - Database Diagram

The result fell perfectly in line with already set expectations. But to be entirely sure that nothing could be forgotten the database got data added to it by seeding it through a class that has this sole purpose. The class is then called once in the startup and then commented out, to avoid adding the same data in the database every time the program would run.

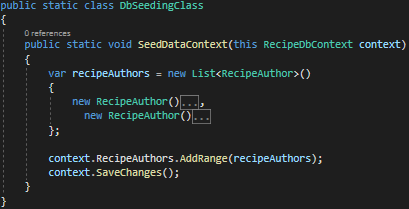


Figure 11 -Seeding

On of the greatest thing of .net Core is it’s moduled state, that different components can be brought to a system as services. This is where Interfaces do their part. Connecting a concrete class through it’s interface makes a system more maintainable and scalable.

For each of the model classes a IRepository and a concreate Repository class were created to exchange data with our database.

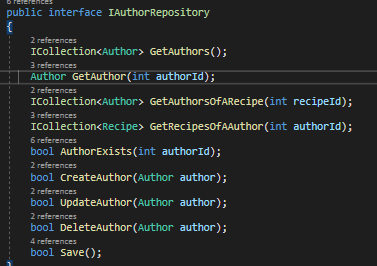


Figure 12 - IRepository

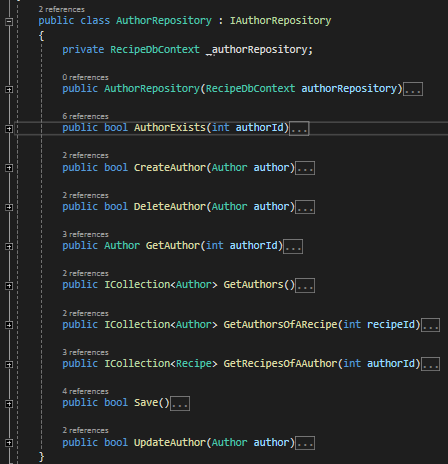


Figure 13 - Repository

A repository class implements a interface class. The reference to the repositories is made in the startup class.

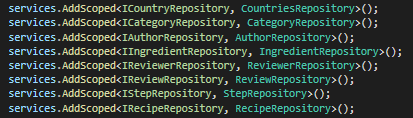


Figure 14 - Referencing In Startup

On top of the Repositories, Controller were built so that different endpoint would be exposed. With these endpoints, communication with other tiers of the system would be possible. But data cannot be sent in the form of the existing models to other system tiers, because these models contain navigational proprieties that only make sense to have in the Persistence tier.

And so, Data Transfer Objects had to be created, one for each model class we have.

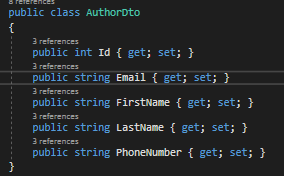


Figure 15 - DTO

These objects only contain the information that makes sense to have and use in the other tiers of our system.

The last step was to create the Controllers that would expose the needed endpoints, one controller for each model. The controllers work using HTTP requests and they were designed keeping in mind REST principles.



Figure 16 - Controller

### 5.2 Business Logic Tier

### 5.3 Presentation Tier

The presentation tier is created with the purpose of displaying data to the use and should be connected to the Business Logic Tier of the system. Since the Business Tier of the system was not read in due time, the Presentation Tier, for now, is connected straight to the Persistence Tier of the system. This is done by consuming the API expose by the Persistence Tier of the system and sending HTTP request to the endpoints.

Information is displayed with the help of ASP.NET core Razor pages. Minor styling has been added to the project. This was achieved by using Bootstrap.

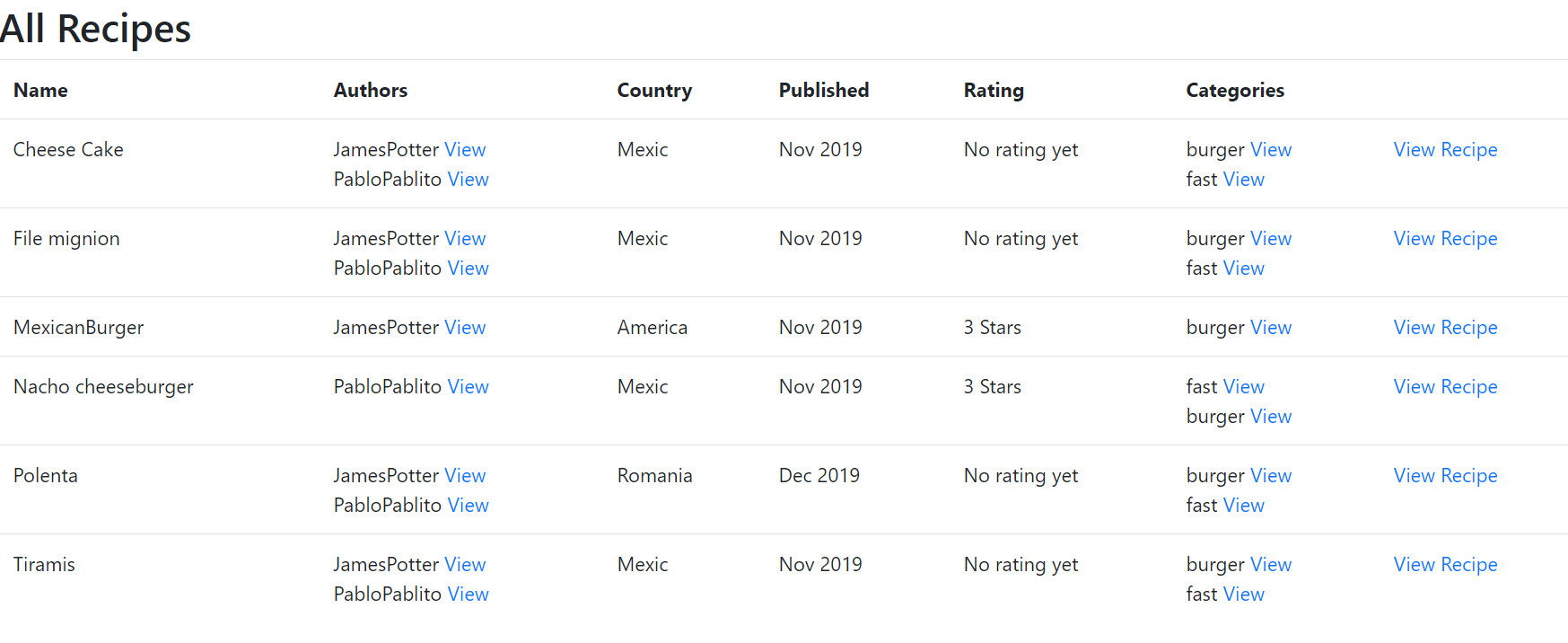


Figure 17 - Home Page

For the data transfer to be consistent equivalent DTO objects had to be implemented in this tier.

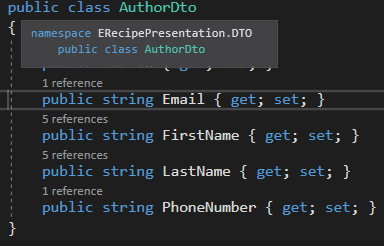


Figure 18 -DTO

The connection between these tiers is made again using IRepositories and Repositories classes. The difference is that in this case, the Repositories are aiming to our Persistence Tier and not to a database.

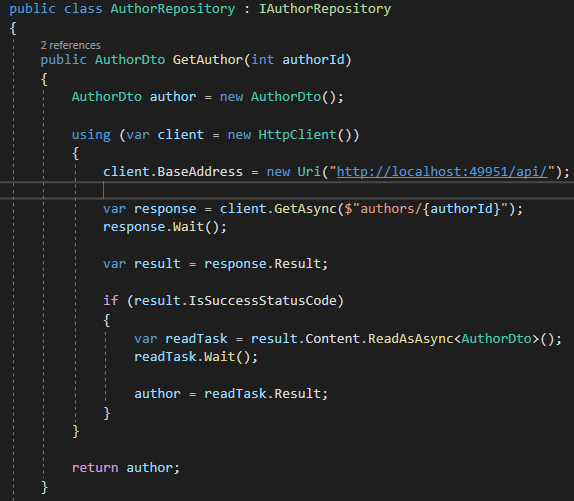


Figure 19 - Repository

The IRepositories and Repositories have to be registered in the Startup class along with other components to make routing possible.



Figure 20 - Startup

If the Repositories are responsible of retrieving data, Controllers are created to send if further up the system, in this case toward the Views. Each data transfer object has is represented by it’s own controller, but in this case they are not enough for displaying required data for each of our web page. This because on each Razor Page you can only use one model, but for example on a recipe page, next to the recipe itself, you also need to display the authors, ingredients, steps, categories, reviews and linked reviewers. So to make this possible, View Models are created according to the needs of each page.

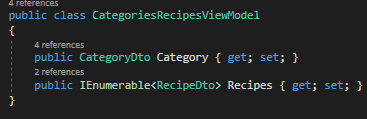


Figure 21 - ViewModel

This view model is created so that on a category page, created for the GetCategotyById, recipes that belong to that specific category can also be displayed.

In our context, for each action in a controller, a Razor Page is created.

## 6 Testing

### 6.1 Persistence Tier

With the project being forced to advance at a vey fast pace, for the presentation and persistence layers no unit tests have been done. This was a risk and a sacrifice that had to be made.

With the persistence layer being finished, a suite of tests for the controllers was created. The implementation itself was extremely helpful with testing, because every controller method specifies all the possible response types and the url format that needed to be accessed.

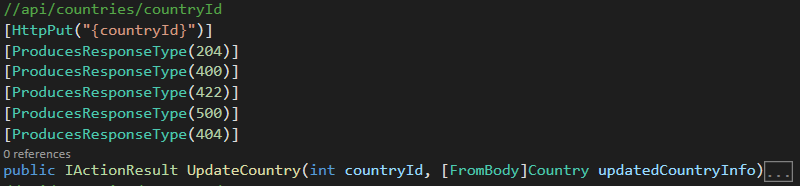


Figure 22 - Response Types

The test suite that was prepared for the controllers contains tests for all the possible scenarios that can be encountered in our current business context.

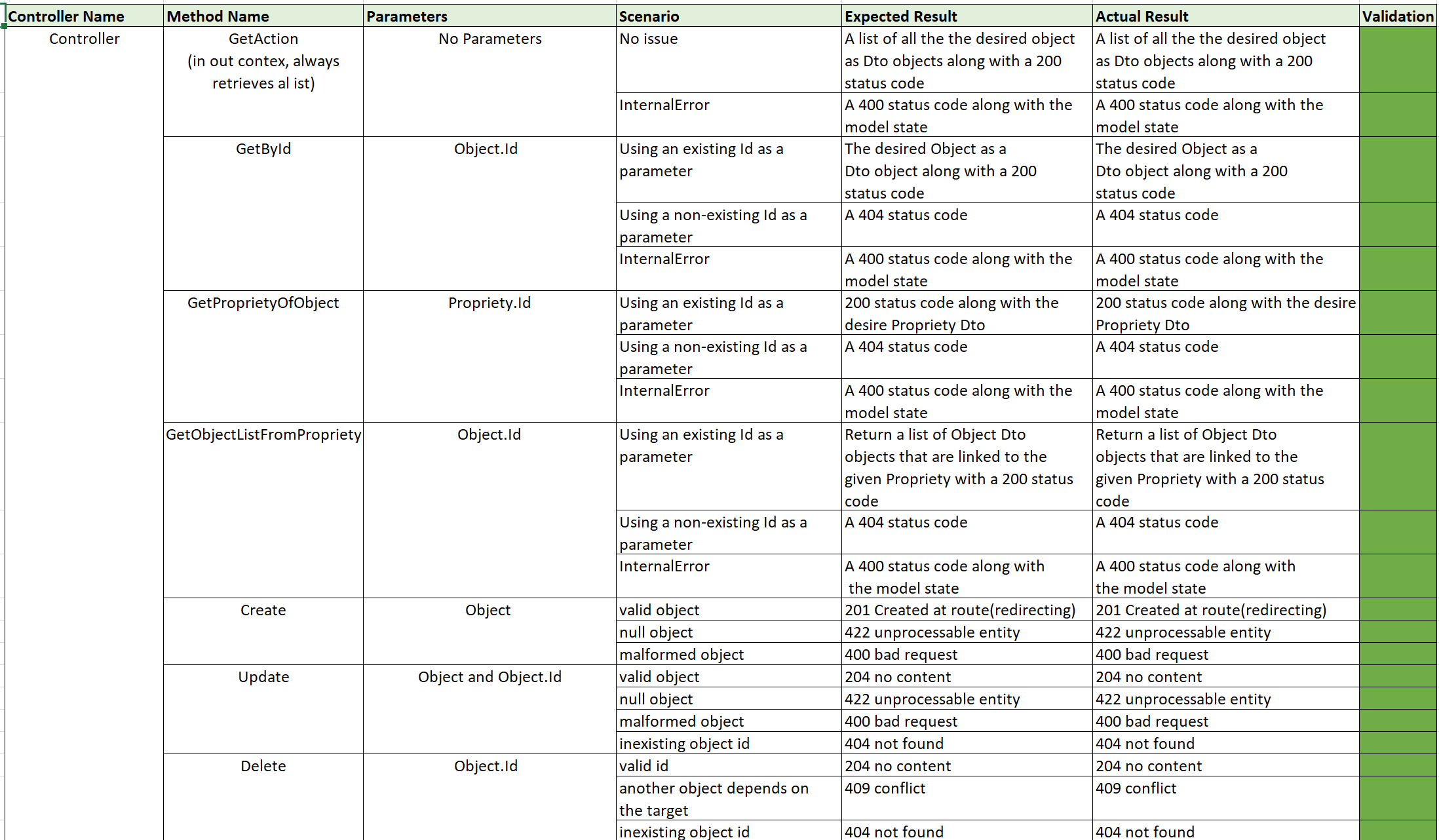
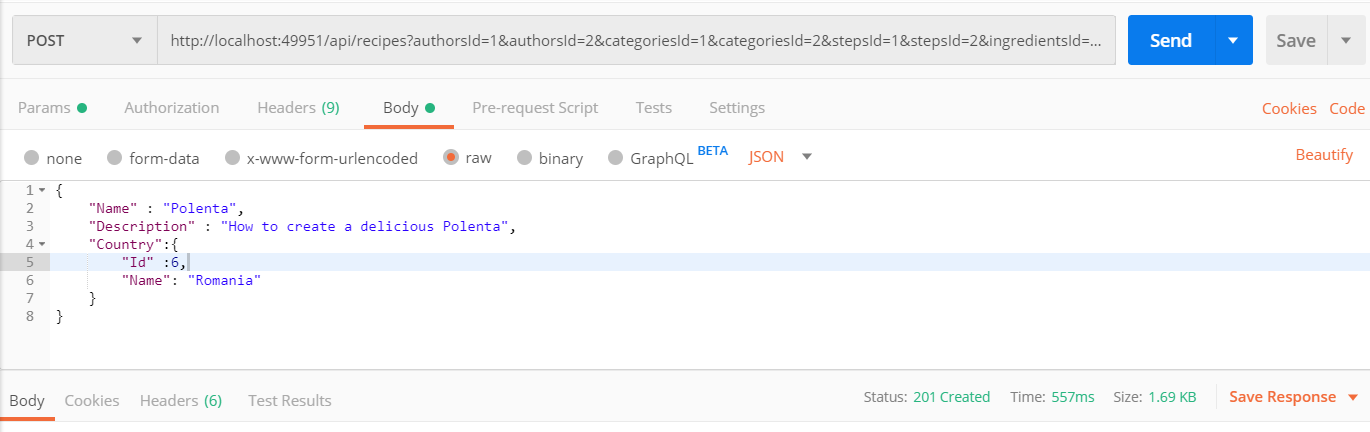


Figure 23 - Test Suite

The test suite was a success and it proved that it tested every piece of functionality that the Persistence Tier had to offer.

It is true that one minor flaw was discovered during tests. That is that while trying to update a “Recipe”, the response would consist in a 400 Bad Request status code, although the item in discussion would still be updated.

The suite was tested using Postman.



### 6.2 Business Logic Tier

### 6.3 Presentation Tier

Since time was still pressuring down on the project, no unit tests were written for the presentation tier, but since the result is a simple website that covers most of the functionality implemented in the first two tires, white box testing was considered sufficient at the time.

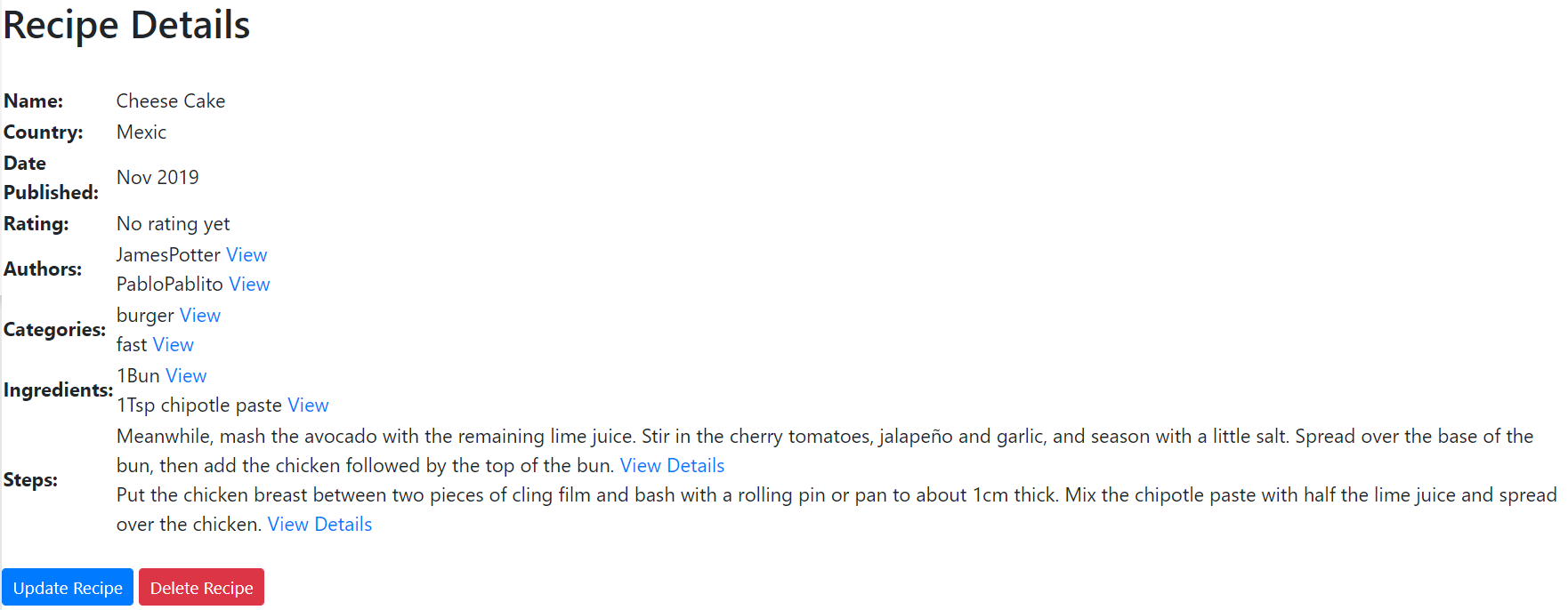


Figure 24 - Layout

## 7 Results and Discussions

The purpose of this project was to create an electronic book of recipes, called E-Recipe, where user could inspire themselves and try new recipes, as well as post their recipe. The group succeeded in creating the system, accomplished the requirements of having a layered architecture, consume and expose a web service as well as having a GUI for each client. However, there were some issues with creating the Business Logic Tier, written in Java programming language.

In the end, this is a functional project of two-tiers with a friendly user interface, written in C# programming language. The graphical design of the user interface is as user friendly as possible, however, there could have been implemented some more graphical features.

Overall there is room for future improvements and building more features.

## 8 Conclusion

This semester project was challenging from the beginning till the end. The final idea has come a long way till it was developed in a functional and usable application.

A rocky overview of the system is first made in the analysis part, described in the section above. It illustrates system’s behavior as well as how it will communicate with the user. Furthermore, the analysis part has taken the team to the opinion of not implementing any security features yet, as it requires lots of time the team didn’t have.

In the design section, there can be seen all the diagrams created and a brief explanation of every piece. The diagrams help the team to have a better understanding of how the system will exchange information between the tiers. Also, diagrams sharped the need of using design patterns in order to have a more readable and maintainable project.

The project implementation section took most of the time as it was a huge system to put into action. However, it was successful implemented along with some White Box testing described in the section above.

As a result, the group members have concluded that a good part of the requirements were accomplished. The system still needs some future improvements, but overall it is great project in the existing timespan.

## 9 Project Future

For future updates of the system, there are a few important functions to be implemented. First, for security features of the system, to have a login functionality in order to personalize users and distinguish between one another. Also, there could be implemented two actors, this being the user and the administrator – who will be responsible for managing users and will have access to more features than the user. Second, implementation of the middle tire, or the business logic tier, written in Java programming language. Third, there could be implemented a more eye-catching user interface, so the user will have a better experience with the application.

# Sources of information

Britannica, T. E. o. E., 2019. *Encyclopaedia Britannica.* [Online]   
Available at: https://www.britannica.com/topic/food  
[Accessed 2 October 2019].

## Appendices

Appendix A: Project Description

Appendix B: Use Cases

Appendix C: Test Cases

Appendix D: ERecipe Diagrams