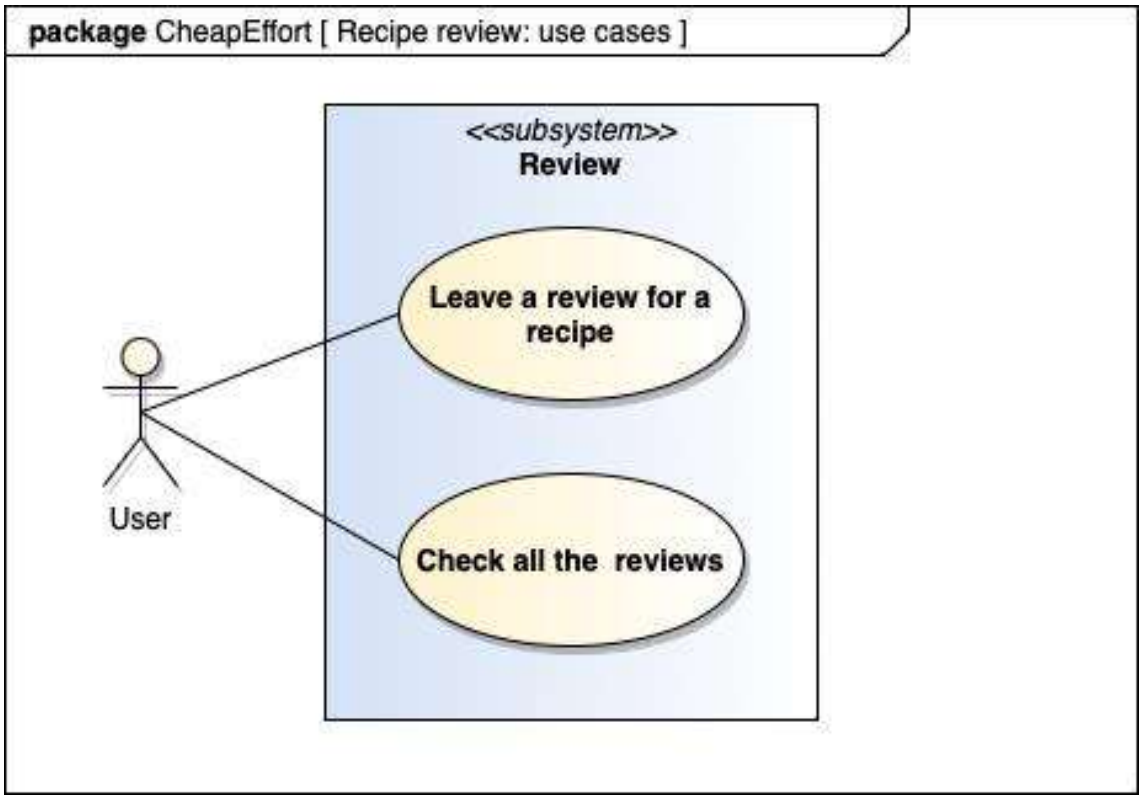
VILNIAUS UNIVERSITATAS
MATEMATIKOS IR INFORMATIKOS FAKULTETAS
PROGRAMŲ SISTEMŲ KATEDRA

PROGRAMŲ SISTEMŲ INŽINERIJA II

ANTRASIS LABORATORINIS DARBAS
CHEAP EFFORT

Tedas Grinkevičius
Klaidas Sinkevičius
Pijus Zlatkus
Vadim Čeremisinov

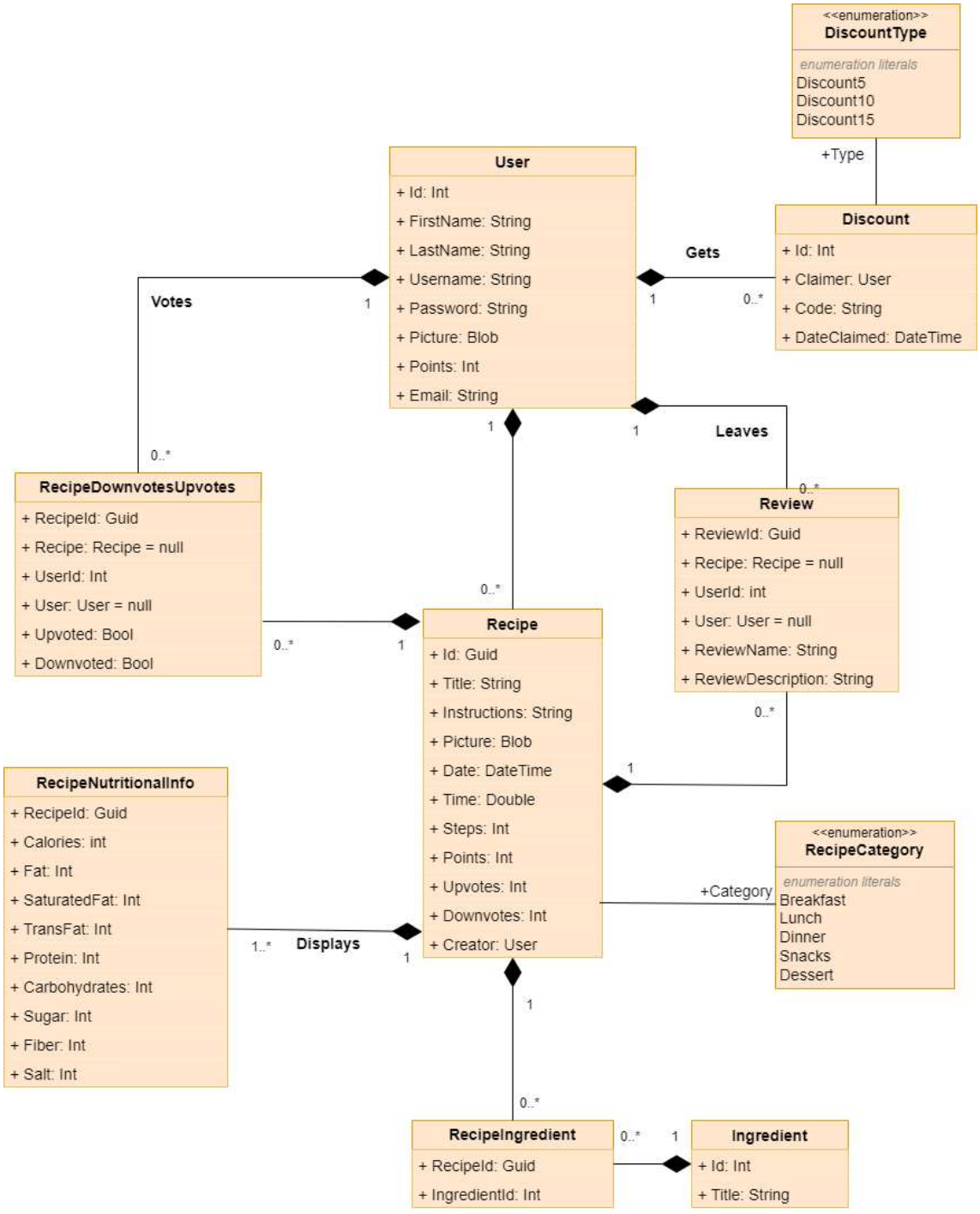
Vilnius, 2023



Introduction

In the initial laboratory work, we have identified the user needs and requirements for our software program. This document aims to present the architecture of the system and its new features. To analyze these features, we will use the UML 4+1 framework, which comprises five views: Logical, Development, Process, Physical, and Use Cases. We have structured this document to analyze each feature individually, providing a comprehensive analysis before moving on to the next one. Additionally, we will describe the CI/CD process that we have implemented for this project, followed by a traceability matrix that maps features to system requirements.

Information Model



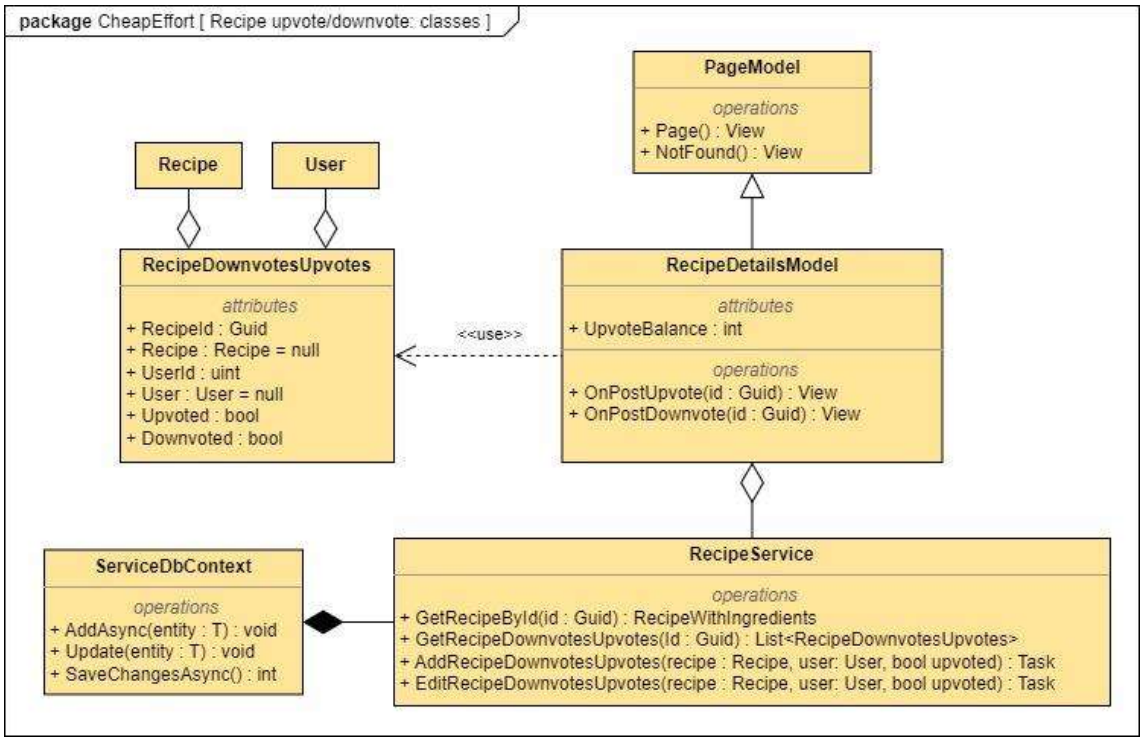
Information model demonstrates the components of our application's entities and their relationships. There are two main entities that everything revolves around: User and Recipe. The User entity depicts any user of the platform. Recipe entity describes the recipe model having a recipe category. Discount and Discount Type are entities used to award users for their recipe creation. User can have many recipes and many discounts. RecipeDownvotesUpvotes is another entity that acts as a pivot table to keep the track of upvote/downvote system records of

users and recipes. Finally, RecipeIngredient is the pivot table for the recipe ingredients, depicted by the Ingredient entity and Recipe entity. In this document the terms “user” and “chef” are used interchangeably as a regular user can use chef functions in our platform.

Feature Analysis

Recipe upvote/downvote

Logical View



PageModel: a class provided by the ASP.NET Core Razor Pages framework. Must be inherited by all page models that handle Web requests and return HTML pages.

RecipeDetailsModel: resides in the presentation layer. It acts as a controller and is responsible for handling Web requests and rendering page models. It has two methods that return View, a type that returns HTML pages.

RecipeService: resides in the service layer. It is responsible for implementing the business logic required to handle the request received from RecipeDetailsModel.

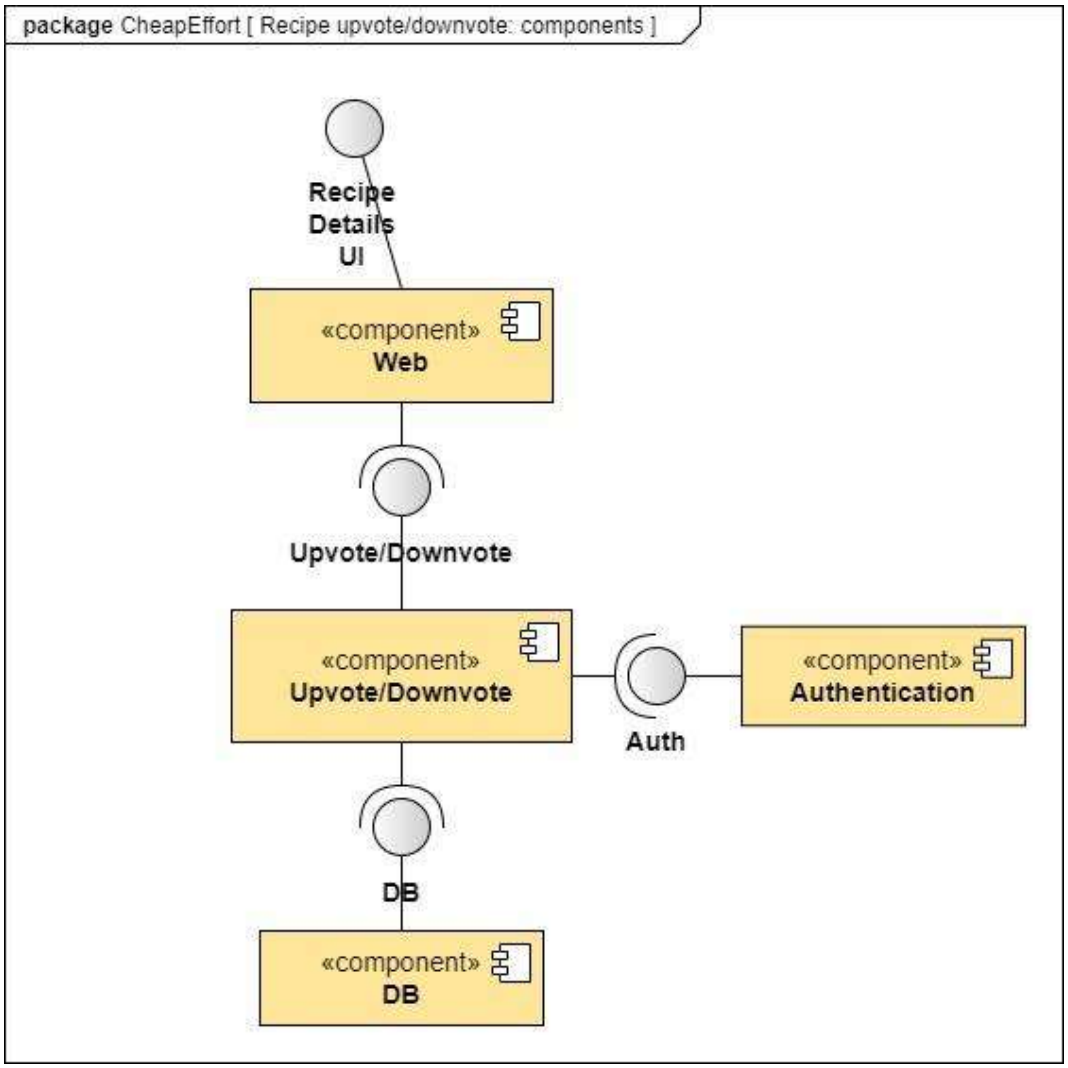
RecipeDownvotesUpvotes: entity representing the recipe's upvotes and downvotes in the database. It is used to identify what users upvoted and downvoted the recipe.

Recipe: an entity that represents a Recipe in the database and is used to identify what recipe is upvoted or downvoted by a user.

User: an entity that represents a User in the database and is used to identify what user upvoted or downvoted a recipe.

ServiceDbContext: a class provided by the Entity Framework Core that defines how a database communicates with the application and maps database objects to application entities.

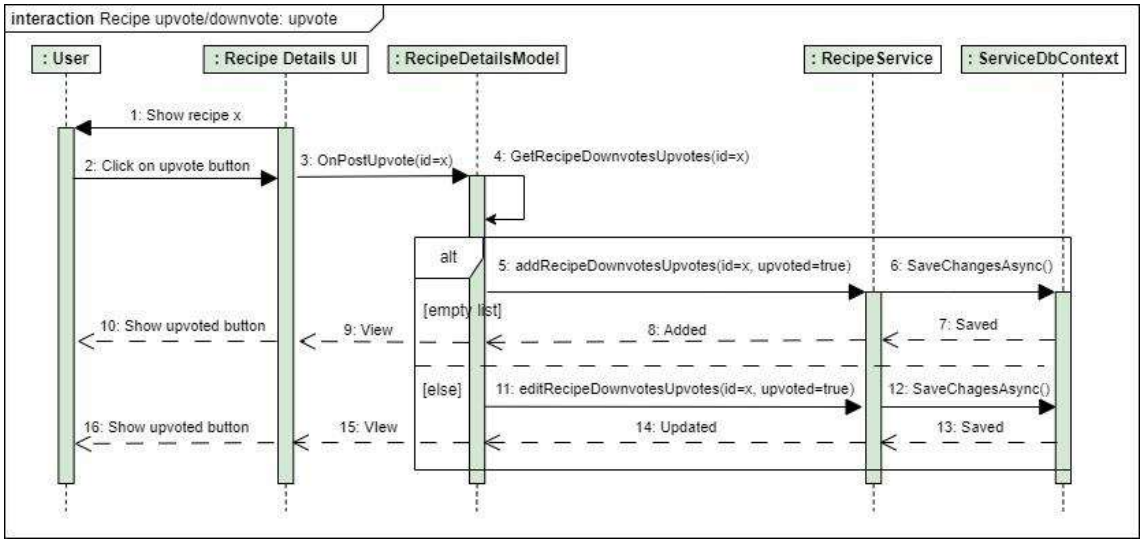
Development View



The upvote/downvote component diagram has four components. Web component provides the Recipe Details UI. To provide upvotes and downvotes to Recipe Details UI, Upvote/Downvote interface is used that is provided by the corresponding component. Chat component uses an Auth interface for Authentication and DB to transfer upvotes and downvotes to the database.

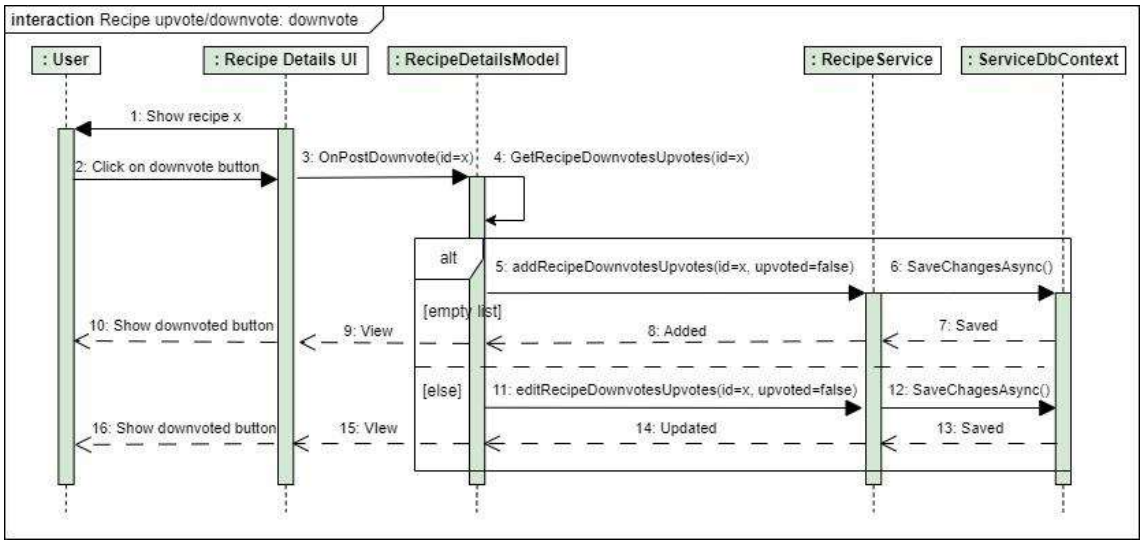
Process View

Upvote



The sequence diagram above describes the recipe upvote system. At first, the sequence starts when a user checks a recipe, and an upvote button is shown to the user. To upvote the recipe, the user has to click the button. Then from UI, an upvote request is sent to the page model that acts like a controller and takes action. The request is examined from the upvotes/downvotes list and whether the user has already upvoted or downvoted the recipe. In case the returned list is empty and the user is upvoting for the first time, the recipe service is called with an upvoted parameter set to true to add a new upvote to the recipe. Afterward, the recipe service calls the service database to save made changes, and the user is informed when the recipe is upvoted. Otherwise, if the user has already upvoted or downvoted the recipe, then the recipe service is called to edit recipe's upvotes/downvotes with upvoted parameter set to true instead of adding, and the remaining sequence completes respectively.

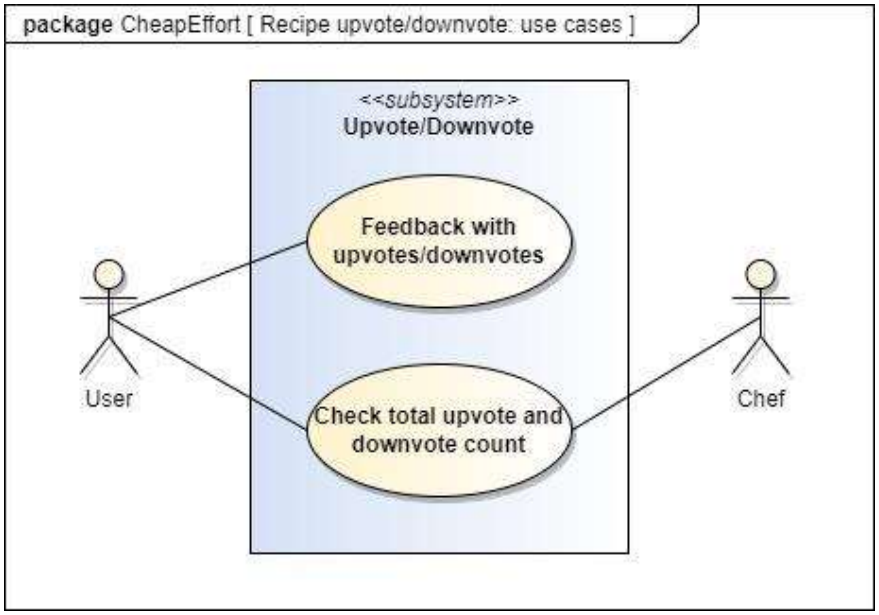
Downvote



The sequence diagram above describes the recipe downvote system. The system works similarly to the recipe upvote system. However, it starts when a user clicks the downvote button instead of the upvote button. The recipe details UI then sends a downvote request to the recipe page model to check if the user is already upvoted or downvoted. Similarly, like in the upvote

sequence diagram, a request is sent to the recipe service but with upvoted parameter set to false. The remaining sequence runs the same.

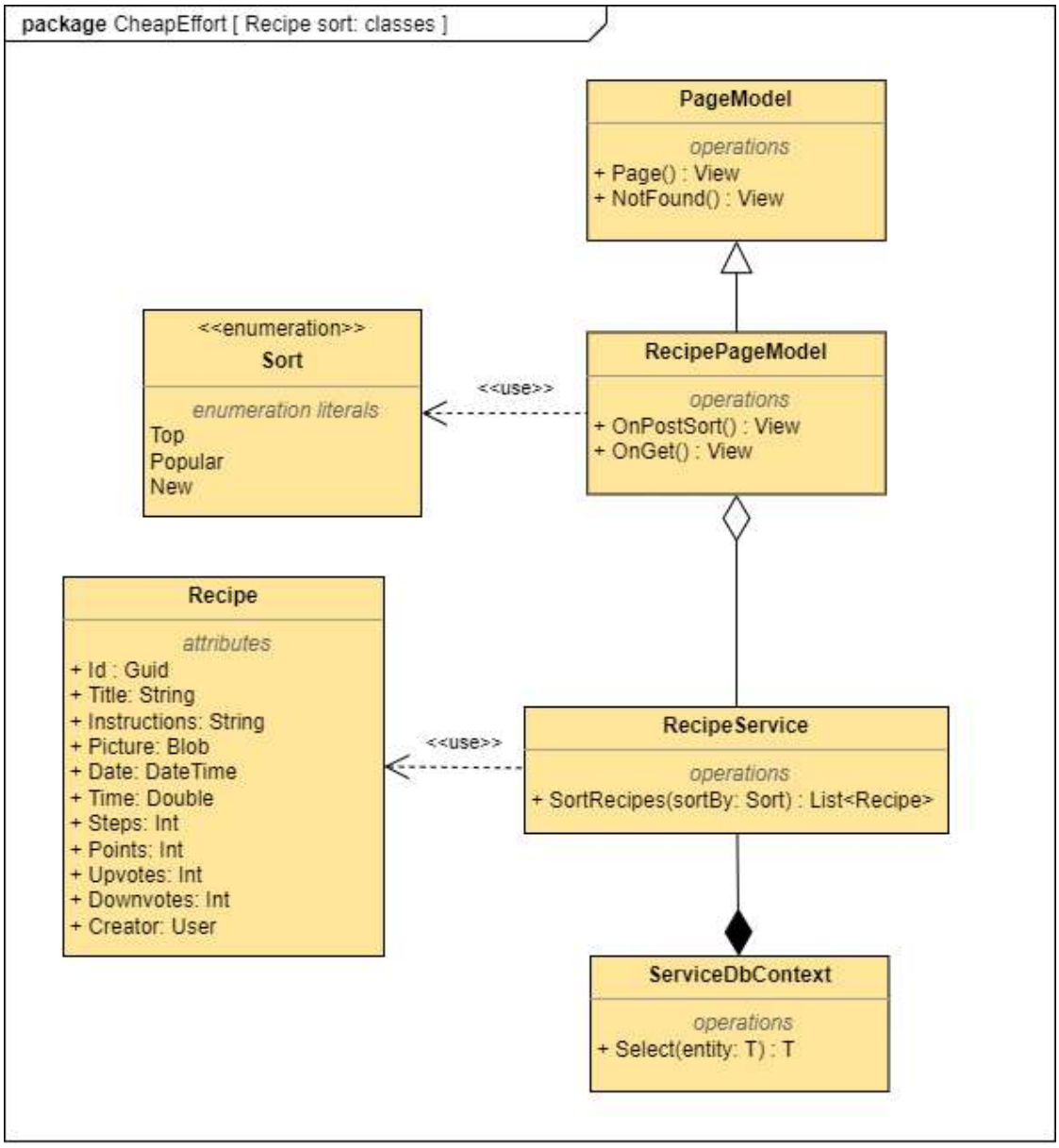
Use Cases



The main goal of upvote/downvote functionality is to provide feedback between users and chefs. The user can upvote or downvote to express his opinion about the recipe, while the chef and the user can check how many other users have given feedback about the recipe.

Recipes Sort

Logical View



Sort enum object: depicts an enumeration object that defines all the ways to sort recipes.

PageModel: a class provided by the ASP.NET Core Razor Pages framework. Must be inherited by all page models that handle Web requests and return HTML pages.

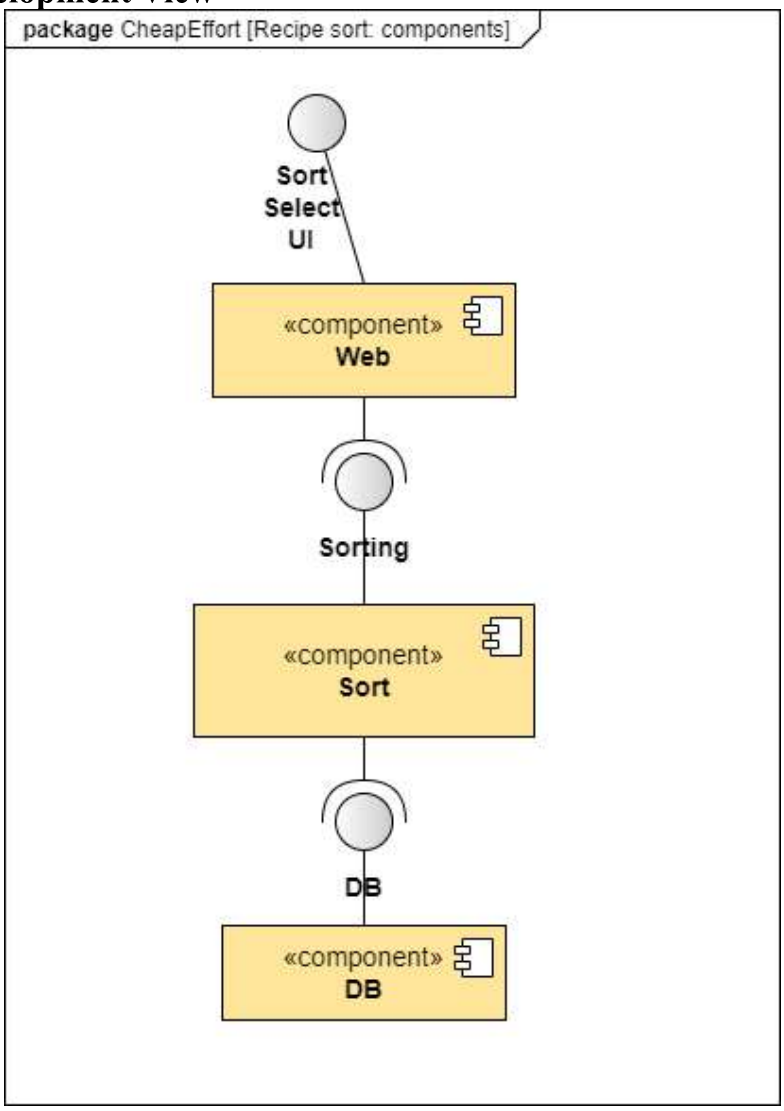
RecipePageModel: resides in the presentation layer. It uses a sort object to retrieve the value of the chosen sort option. It has two methods for displaying and sorting the recipes. It acts as a controller that returns View objects that render pages.

RecipeService: resides in the service layer and is in charge of executing the business logic necessary to properly handle the requests received through the controller.

ServiceDbContext: resides in the data access layer, it is the database context where the data is stored and retrieved.

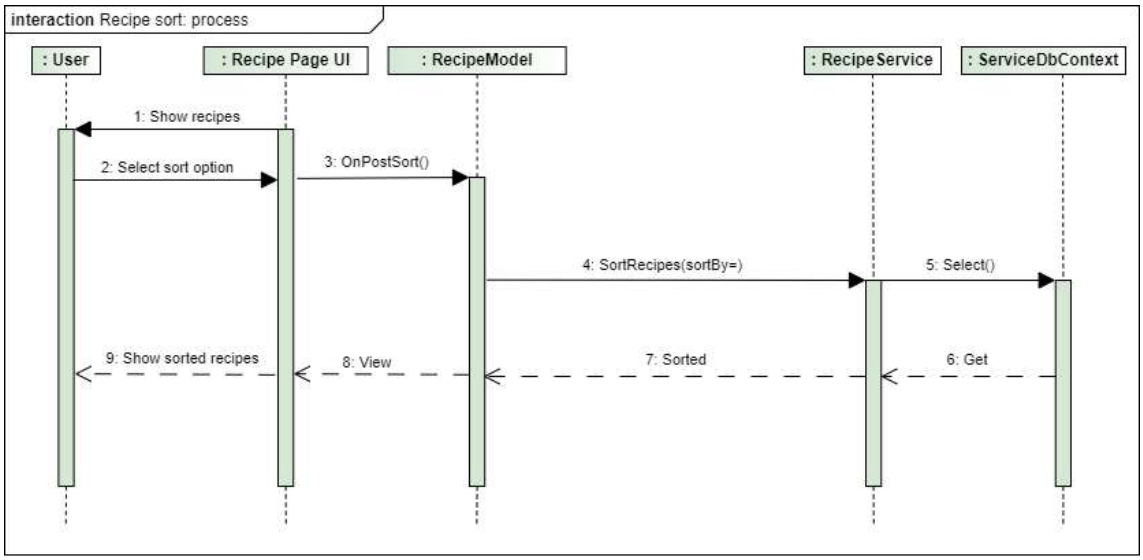
RecipeObject: represents a Recipe entity in the database. It is used by RecipeService to access and retrieve recipe data.

Development View



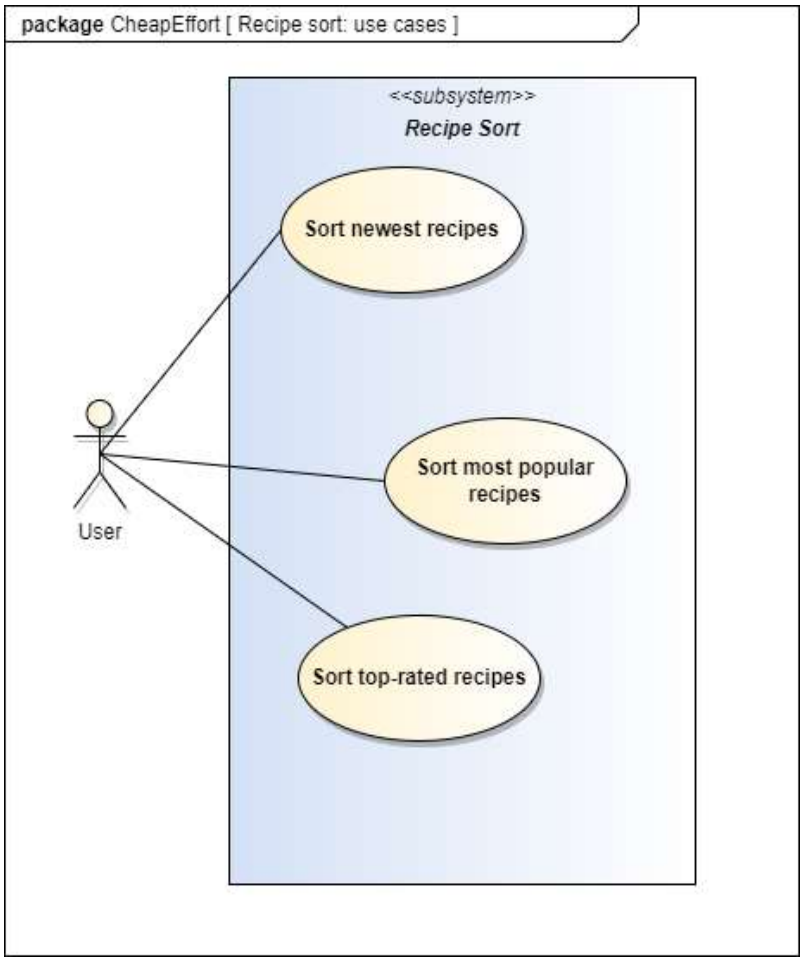
The recipe sort component diagram has 3 components. Web component provides Sort Select UI. Sort component uses DB interface to get the data and provides Sorting interface to the Web which requires it.

Process View



The recipe sort process view can be represented as follows. User is shown many recipes. User can sort recipes through the select button, which sends request to RecipeModel. Then, synchronously, RecipeService method is called which retrieves the displayable recipes and sends them back asynchronously. Next, the list is sorted and sent back to the UI.

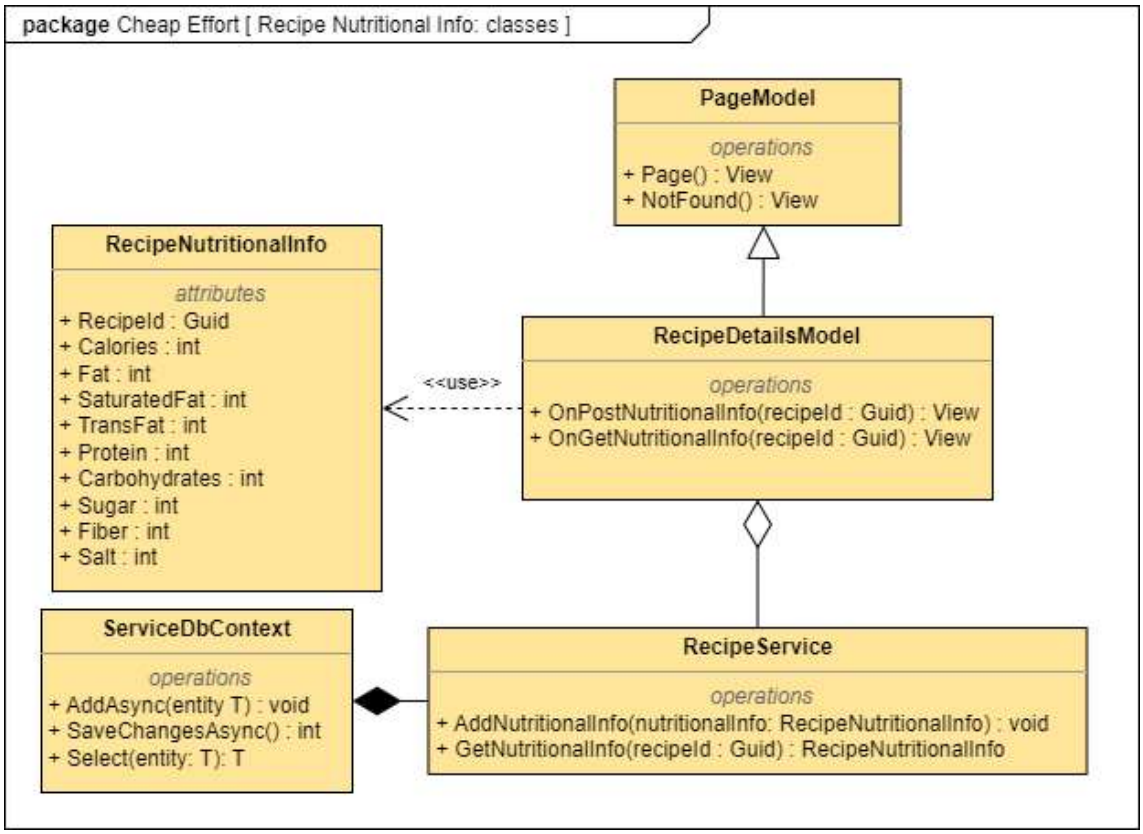
Use Cases



This use cases diagram shows the main functionality of recipe sort select. It is an important feature that lets users select content based on date, popularity and appreciability. The user can sort the content any way he wants.

Recipe Nutritional Info

Logical View



PageModel: a class provided by the ASP.NET Core Razor Pages framework. Must be inherited by all page models that handle Web requests and return HTML pages.

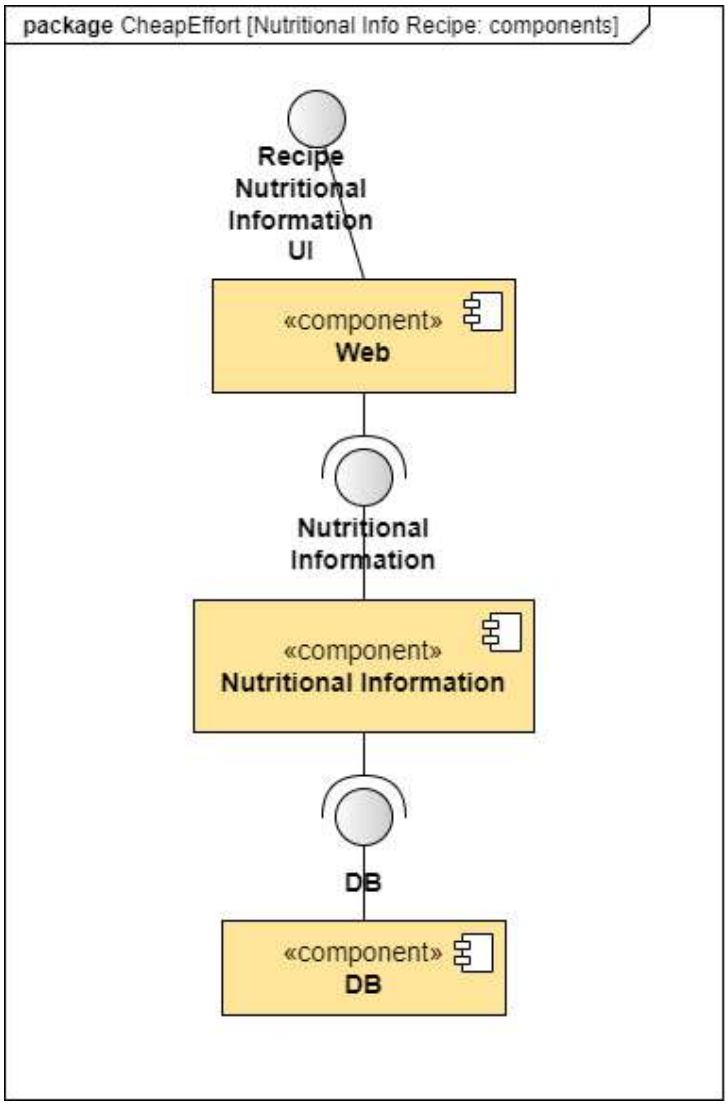
RecipeDetailsModel: resides in the presentation layer. It acts as a controller and is responsible for handling Web requests and rendering page models. It has two methods that return View, a type that returns HTML pages.

RecipeNutritionalInfo: entity that describes nutritional recipe information in the database. It is used to show the nutritional information about the recipe.

RecipeService: resides in the service layer. It is responsible for implementing the business logic required to handle the request received from RecipeDetailsModel.

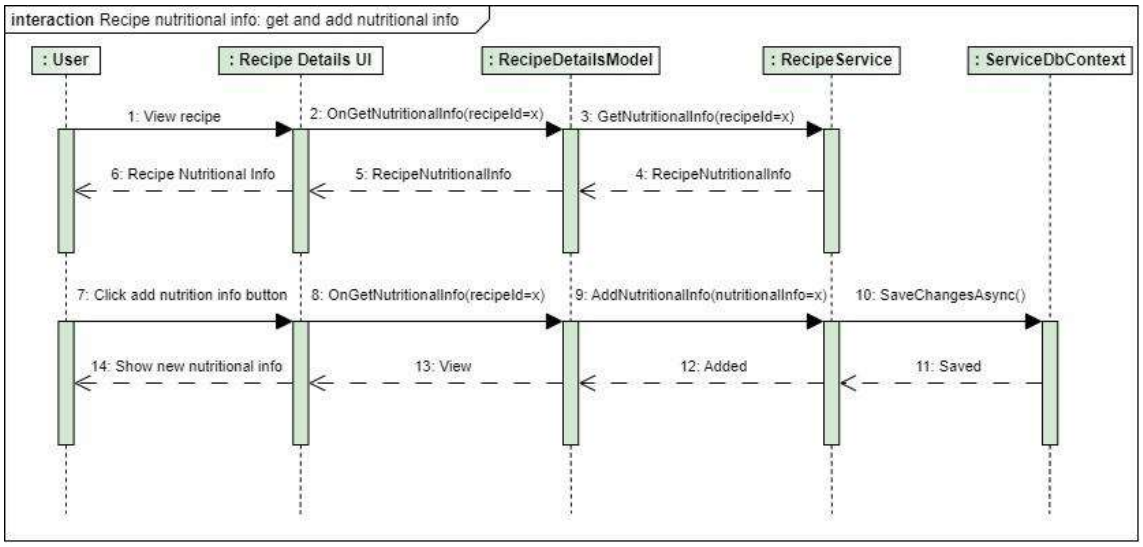
ServiceDbContext: resides in the data access layer, it is the database context where the data is stored and retrieved.

Development View



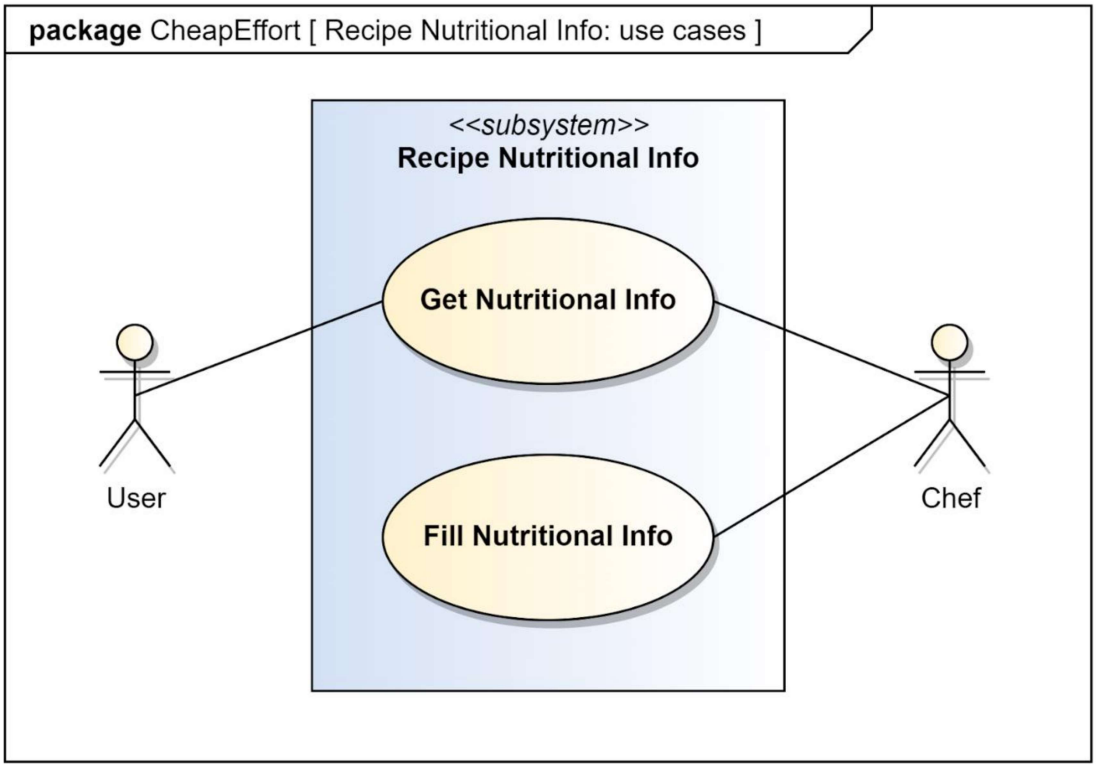
The Nutritional Info Recipe component diagram has three components. Web component provides nutritional information UI. The Nutritional Information component uses DB to query and access the data.

Process View



The sequence diagram above describes the process of adding the recipe's nutritional info. At first, the sequence starts when a user views a recipe. To get all nutritional info, the UI communicates with the recipe details model that acts as a controller. The request is further sent to the recipe service and the user synchronically gets all nutritional info. Afterwards, the user can click the add nutritional info button to add more nutritional data to the recipe. The request from UI goes through the recipe details model and recipe service to the service database context to add new nutritional info to the database. Then all data asynchronously is added to the database and is shown to the user.

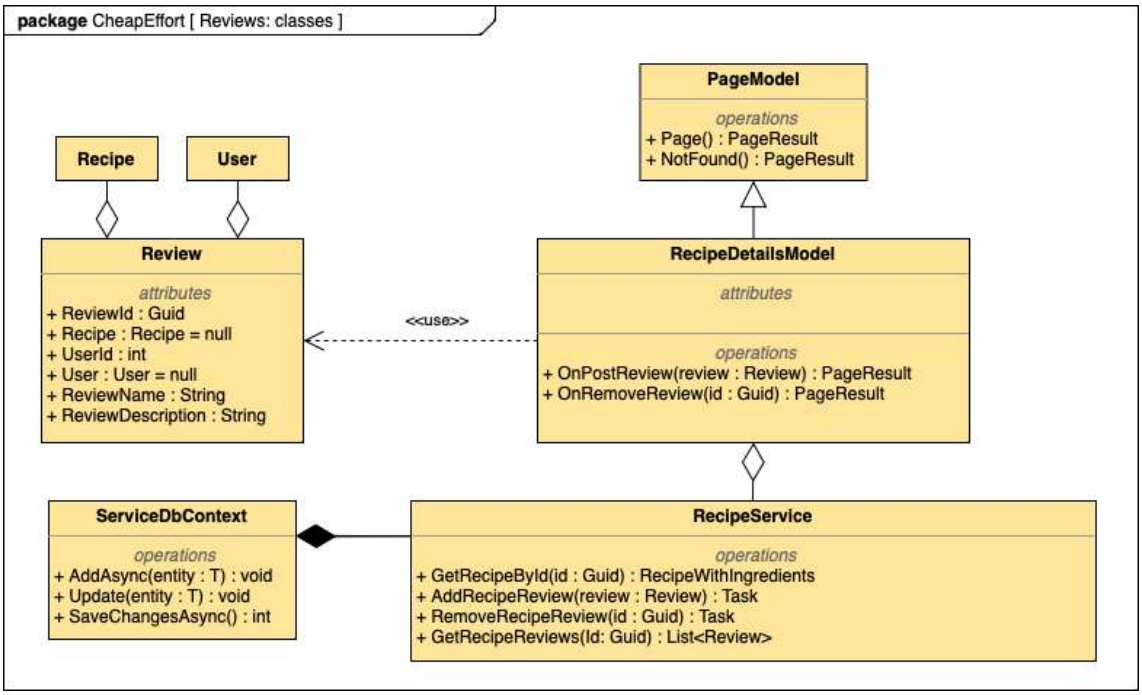
Use Cases



The above use cases diagram represents the main functionality of the recipe nutritional information feature. This significant feature allows users to get and read nutritional information of a recipe that they are viewing. The chef, meaning the author of a recipe, has a possibility to fill the nutritional information while adding the recipe.

Reviews

Logical View



PageModel: a class provided by the ASP.NET Core Razor Pages framework. Must be inherited by all page models that handle Web requests and return HTML pages.

RecipeDetailsModel: resides in the presentation layer. It acts as a controller and is responsible for handling Web requests and rendering page models. It has two methods that return View, a type that returns HTML pages.

RecipeService: resides in the service layer. It is responsible for implementing the business logic required to handle the request received from RecipeDetailsModel.

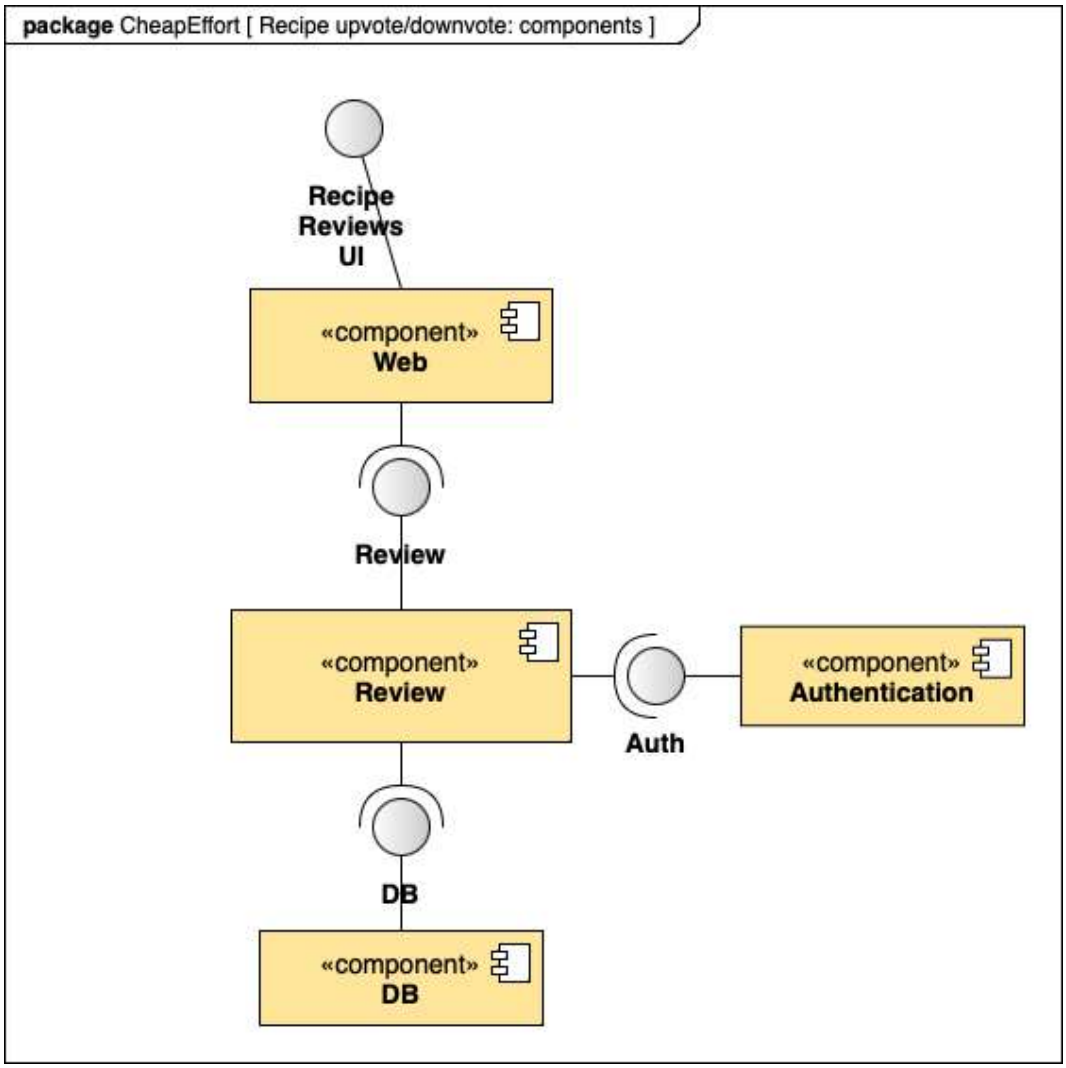
Review: entity representing the recipe's review in the database. It is used to identify what users upvoted and downvoted the recipe.

Recipe: an entity that represents a Recipe in the database and is used to identify what recipe is upvoted or downvoted by a user.

User: an entity that represents a User in the database and is used to identify what user upvoted or downvoted a recipe.

ServiceDbContext: a class provided by the Entity Framework Core that defines how a database communicates with the application and maps database objects to application entities.

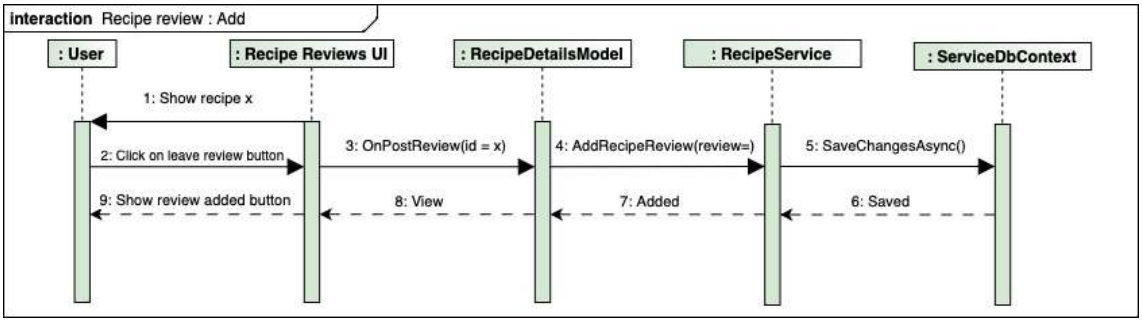
Development View



The review component diagram has four components. Web component provides the Recipe Reviews UI. To provide reviews to Recipe Reviews UI, Review interface is used that is provided by the corresponding component. Chat component uses an Auth interface for Authentication and DB to transfer reviews to the database

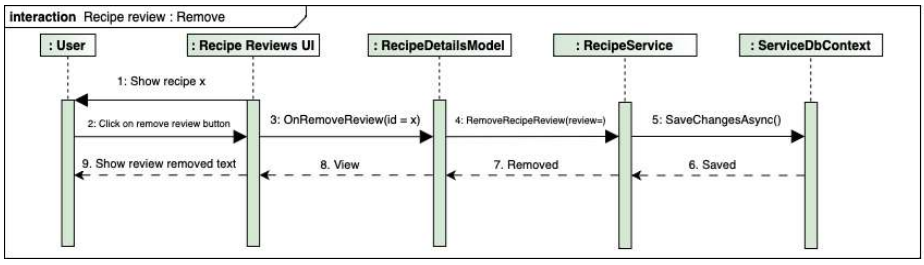
Process View

Adding Review



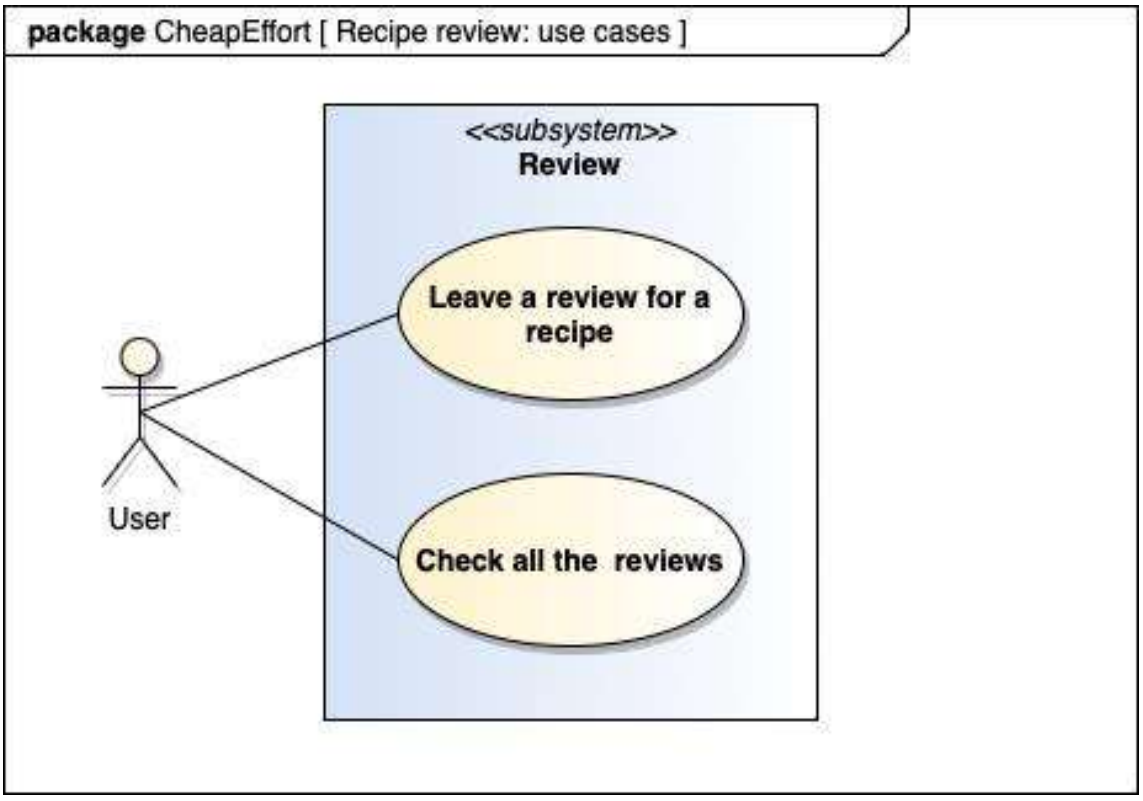
The depicted sequence diagram represents a system for adding a review to a recipe. The sequence initiates when a user opens a recipe page, where they can view previously posted reviews and a form to add their own. Upon filling out the necessary fields, including their name and review description, the user clicks on the 'POST' button to submit their review request. The page model, functioning as a controller, receives the request and takes appropriate action. Next, the review service contacts the database service to save the newly added review. Finally, the user is able to view their posted review.

Removing Review



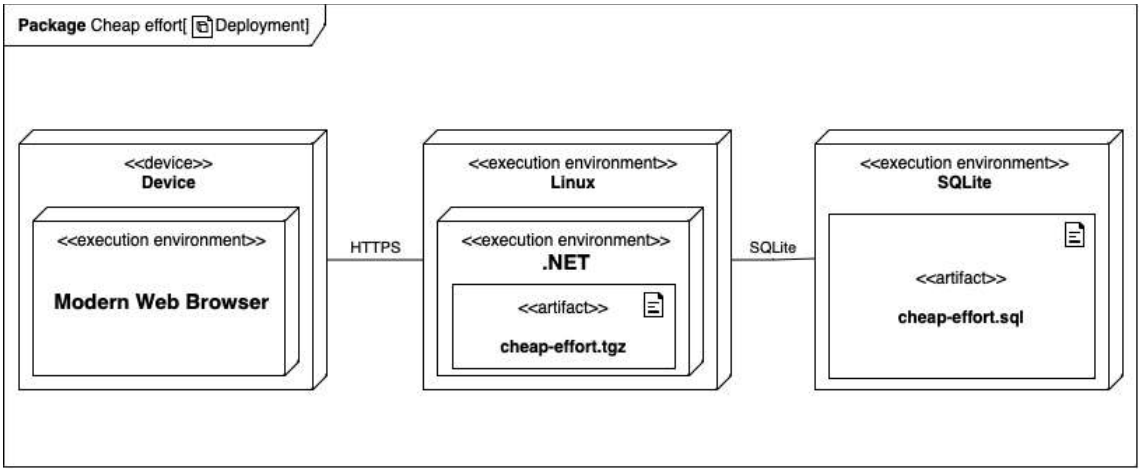
The depicted sequence diagram represents a system for removing a review from a recipe. The sequence initiates when a user opens a recipe page, where they can view previously posted reviews . Upon filling out the necessary fields, including their name and review description, the user clicks on the 'POST' button to submit their review request. The page model, functioning as a controller, receives the request and takes appropriate action. Next, the review service contacts the database service to save the newly added review. Finally, the user is able to view their posted review.

Use Cases



This use cases diagram shows the main functionality of recipe review. It is an important feature that lets users select content based on date, popularity and appreciability. The user can sort the content which way he wants.

Deployment



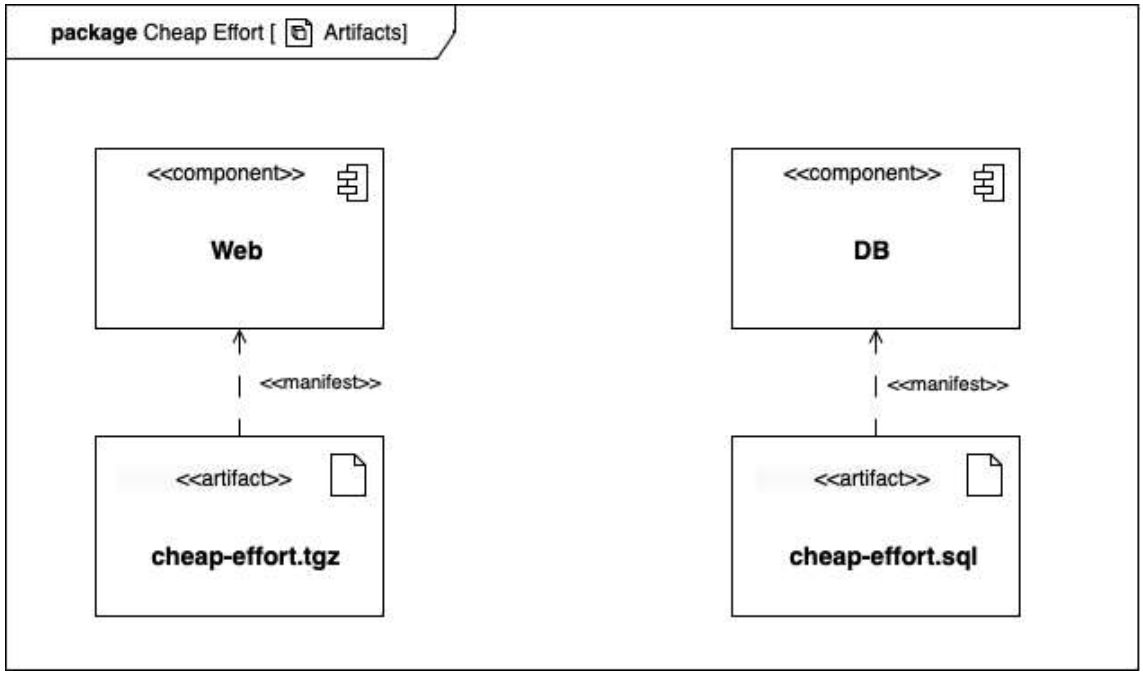
Smart Devices communicate with the application execution environment via HTTPS and the .NET application uses SQL over TCP to query the remote SQLite database. Requirements for the devices:

- Any OS that is capable of running a modern web browser (personal computers: Windows, Linux, MacOS. Mobile phones: Android, iOS)
- Supported web browsers: Google Chrome version 54 or later, Microsoft Edge version 79 or later, Mozilla Firefox version 68 or later, Apple Safari version 14 or later. These versions are chosen as the lowest possible versions as it is recommended to use the latest version of the supported browsers for optimal performance and security.

Minimum requirements for the server:

- OS: Linux (Debian / Ubuntu)
- Dual-core 2GHz CPU
- 2 GB ram
- 32 GB of free storage

Artifacts



The two main components that can be mapped to artifacts are Web and Database. Web component is mapped to an artifact “care-web.tgz” which is an archive that contains the entire static website. Database component is mapped to the “care.sql” file that contains the SQL statements used to populate the relational database with data.

CI/CD

To create and ensure steady, fast and reliable ways of maintaining the project and releasing new updates, Continuous Integrity and Continuous Delivery processes are being used. After each branch merge or code change in VCS (Version Control System), an automatic pipeline process is started, all code changes and components are checked for any issues. If the newly added changes are safe and don't cause any issues, the changes will be deployed.

Tools

We've set up a streamlined CI/CD pipeline using Gitlab's tools. The pipeline runs on a Debian 11 Linux server hosted on a virtual machine within Vilnius University's secure network. In addition to running the pipeline, the server also hosts our web server, which is enabled through Docker images. Whenever a new image is built, the web server automatically restarts to ensure uninterrupted service for our users.

Building stage

Building stage involves only one job: to compile/build the application from source code. .NET CLI command is executed, which ensures that the source code compiles, does not contain any left errors and is ready for the next stage: testing. Since the same server is used for hosting, building does not use Docker, as running the same container multiple times can cause issues.

Testing stage

After building is done, the same artifacts from the building stage are used to run automated tests that are present in the project. This is also done using .NET CLI commands to avoid any issues.

Deployment stage

If all the tests pass, the deployment stage notifies that the new application version is fit for deployment. After that, the application is deployed to the cloud.

Traceability

Functional Requirements

Nr.	Requirements
FR 1	Whenever a user makes a mistake while entering data and clicks proceed, a pop-up warning window should be shown with information of what is wrong and a button to close it.
FR 2	All newest recipes should be displayed in the home page with a picture, which on click routes the user to the recipe's page.
FR 3	A chef should be able to create an infinite amount of recipes.
FR 3.1	Every recipe should have a description of up to 500 words and 50 as a minimum.
FR 3.2	Every recipe should have up to 7 images and 2 as a minimum.
FR 3.3	Every recipe should have a name and tags assigned to it.
FR 3.4	Every recipe should have a list of required products up to 50 and 1 as a minimum.
FR 3.5	Every recipe should be rated on a scale of 5 stars determining how hard it is to make it.
FR 4	A chef should be able to edit or delete any created recipe at any time.
FR 5	A user should be able to rate recipes.
FR 6	A chef and a customer should be able to collect Kudos points for doing certain tasks.
FR 6.1	1 point of Kudos is added for leaving a review on a recipe.
FR 6.2	3 points of Kudos are added after subscribing to a chef
FR 6.3	5 points of Kudos are added after receiving a subscriber
FR 7	Points could always be removed from a user
FR 7.1	Points are reduced for posting fake recipes.

FR 7.2	Points are reduced for commenting or reviewing inappropriately.

Table 1: Functional requirements

The table above shows the list of requirements that were gathered during the first laboratory work.

Mapping to Features

	Recipe upvote/downvote	Reviews	Recipes sort	Recipe nutritional info
FR 1				+
FR 2				
FR 3				+
FR 3.1				
FR 3.2				
FR 3.3			+	
FR 3.4				+
FR 3.5		+		
FR 4				
FR 5	+	+		
FR 6		+		
FR 6.1		+		
FR 6.2				
FR 6.3				
FR 7				
FR 7.1				

FR 7.2		+		

Table 2: Mapping features with requirements

- Symbol '+' in this table represents that the function has a relation to the requirement.
- Several functional requirements are not mapped with any feature as in this laboratory work we are analysing only the new features. uploading process.
- Recipe nutritional information feature is mapped with FR 1 as we aim to avoid user generated errors made while entering nutritional information of a recipe, such as entering letters instead of numbers while filling calories field. Also, this feature is linked to FR 3 because the possibility to create an infinite number of recipes directly affects recipes' nutritional informations as it is a part of it. Lastly, the feature is covered by FR 3.4 as the products needed for the recipe directly affect the content of recipe's nutritional information.
- Recipe sort feature is mapped with FR 3.3 because the name and tags of the recipe are essential in order to have the possibility to sort the recipes.
- Reviews feature is linked to FR 3.5, FR 5, FR 6 and FR 7.2 as these functional requirements are regulating the logical use of possibility to review the content.
- Recipe upvote and downvote feature is linked to FR 5 because the user can choose to rate the recipe by upvoting or alternatively - downvoting the recipe, by doing so he expresses his opinion on the content he has just viewed.