

VILNIAUS UNIVERSITATAS
MATEMATIKOS IR INFORMATIKOS FAKULTETAS
PROGRAMŲ SISTEMŲ KATEDRA

PROGRAMŲ SISTEMŲ INŽINERIJA II
KETVIRTASIS LABORATORINIS DARBAS
CHEAP EFFORT

Tedas Grinkevičius
Klaidas Sinkevičius
Pijus Zlatkus
Vadim Čeremisinov

Vilnius, 2023

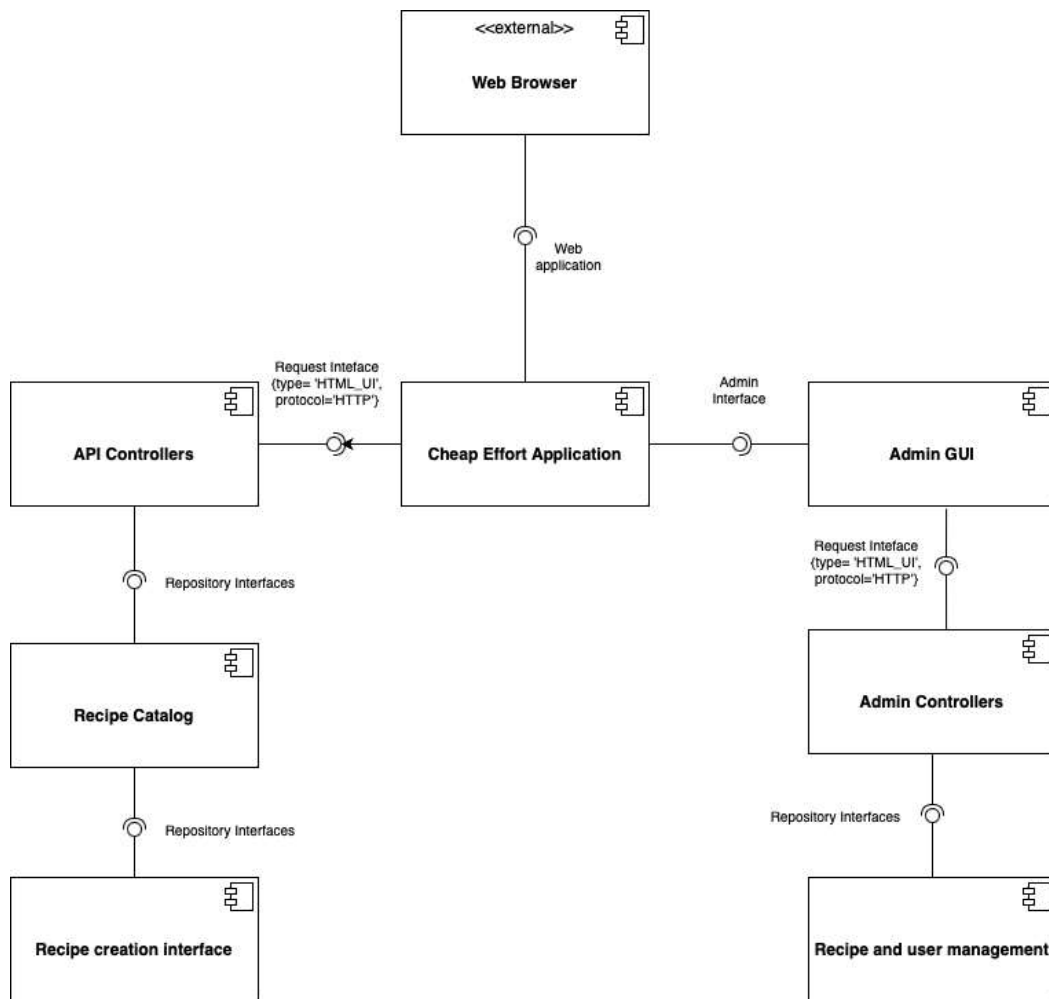
Contents

1	Viewpoints.....	3
1.1	Functional viewpoint.....	3
1.2	Information viewpoint.....	4
1.3	Development viewpoint	5
1.4	Context viewpoint	6
1.5	Deployment viewpoint	7
2	Perspectives.....	8
2.1	Security Perspective	8
2.1.1	Authorization and Access Control	8
2.1.2	Protection of Sensitive Information	8
2.1.3	Auditing and Accountability.....	8
2.1.4	Strengthening Database Security	9
2.1.5	Data Backup and Recovery.....	9
2.1.6	Protection of Private Information	9
2.2	Availability perspective.....	10
2.2.1	Fault Detection.....	11
2.2.2	Crash Recovery	11
2.2.3	Failure prevention	12
2.3	Internationalization perspective	13
2.4	Performance and extensibility perspective.....	13
2.4.1	Response time	13
2.4.2	Scalability	14
3	Architecture styles and patterns.....	15
3.1	Layered Architecture.....	15
3.2	Model-View-Controller (MVC).....	15
3.3	Repository Pattern	16
3.4	RESTful Design	16
4	Testing	17
4.1	Testing Environment.....	17
4.2	Unit and integration tests.....	17
4.3	CI/CD	18
5	Traceability.....	20

1 Viewpoints

We chose to represent only those viewpoints which were relevant for Cheap Effort system: functional, information, development, context and deployment. Concurrency viewpoint was left out, since the system does not actually have any processes that are happening simultaneously. The operational viewpoint was also ignored, since Cheap Effort does not have enough not only financial, but also work force resources for administrating and overseeing.

1.1 Functional viewpoint

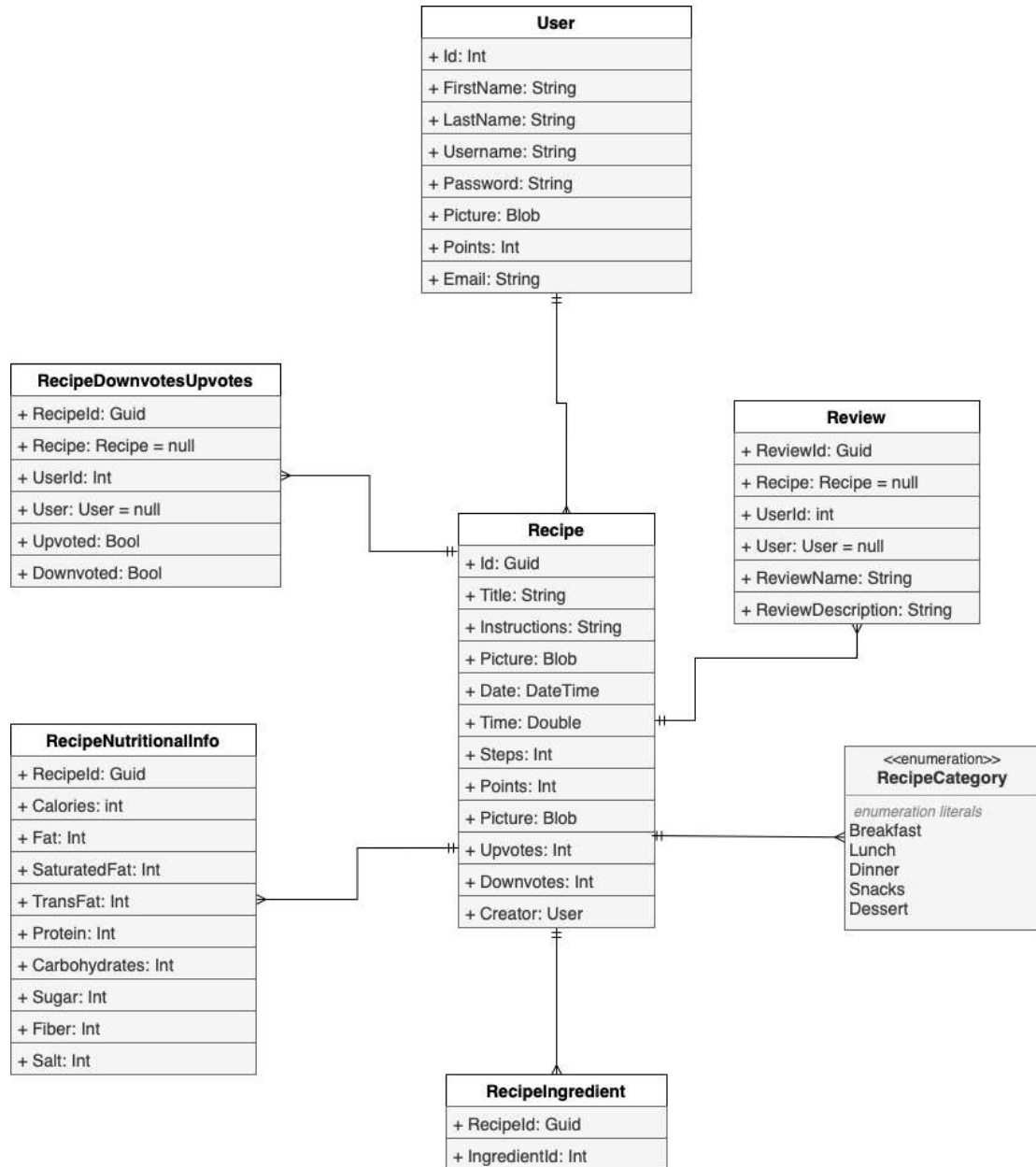


1 Fig. Functional viewpoint diagram

Functional viewpoint shows system's functional elements, interfaces and interactions. In this diagram we can see that a user via his or her web browser is communicating and working with Cheap Effort Application. Admins can access Admin GUI, in which he or she has a

possibility to oversee other users' information and manage recipes. By using API controllers (inbuilt into Pages in Razor Pages), Recipe catalogue and a recipe creation interface can be reached.

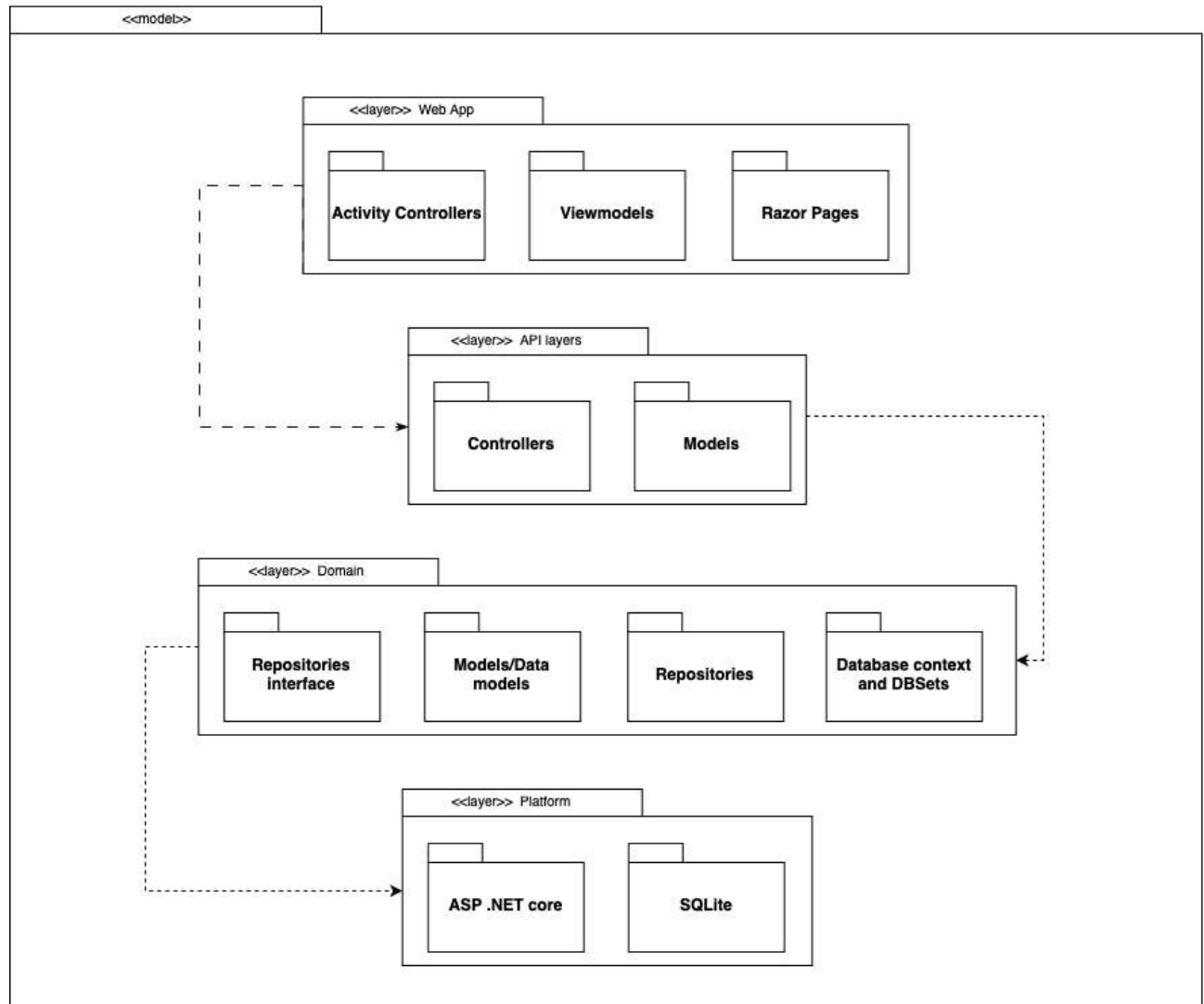
1.2 Information viewpoint



2 Fig. Information viewpoint diagram

Every recipe has its ingredients, nutritional information, reviews, a category and a number of upvotes or downvotes. A user, who could be a chef or an admin, may have his or her own recipes.

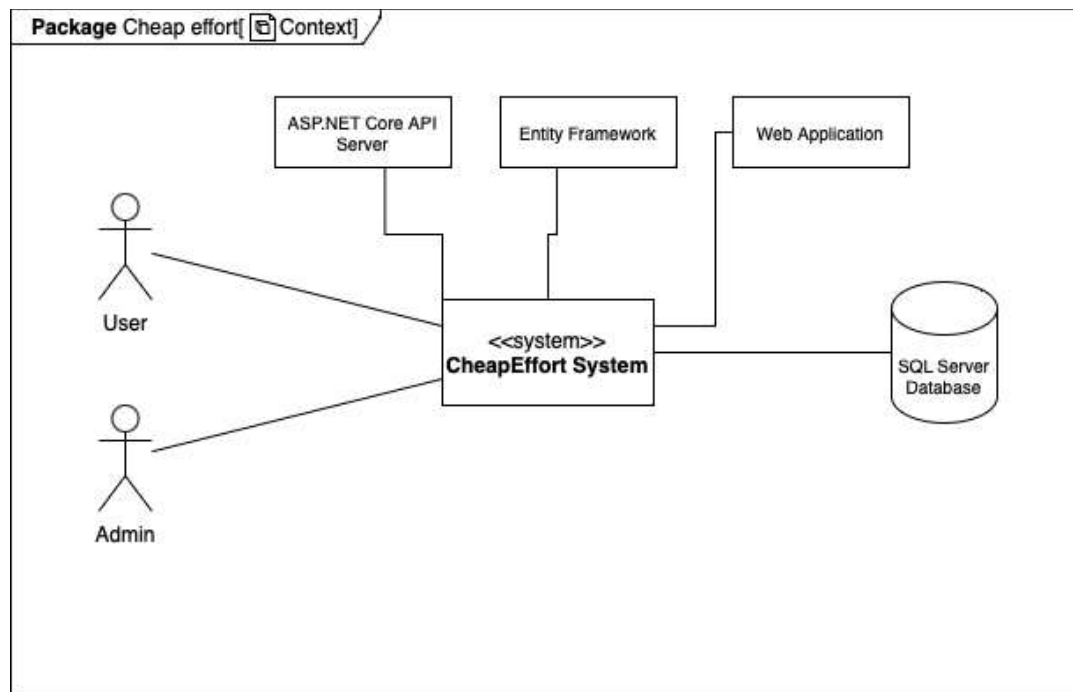
1.3 Development viewpoint



3 Fig. Development viewpoint diagram

This development viewpoint diagram shows Cheap Effort system layers and connections between them. The web app layer communicates with API layers, which then communicates with Domain, which in the end communicates with platform.

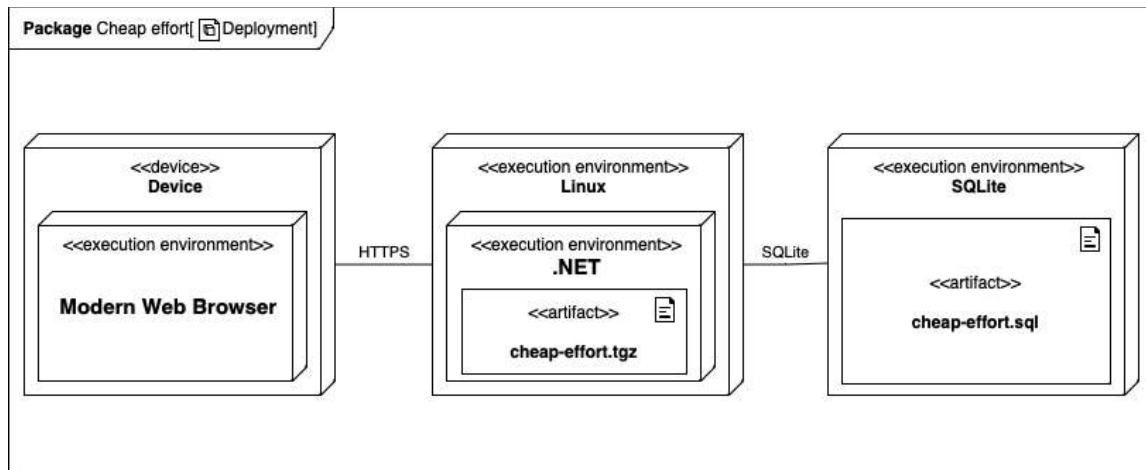
1.4 Context viewpoint



4 Fig. Context viewpoint diagram

This context viewpoint diagram represents all the internal or external systems and data sources that Cheap Effort is made from. Since our system does not use any external APIs, they are not seen in the diagram. The internal systems are: ASP.NET Core Api Server and a Web Application, while the data source is SQL Server Database. Cheap Effort System is accessible and usable for all users and admins who opens the URL.

1.5 Deployment viewpoint



5 Fig. Deployment viewpoint diagram

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 can run 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

2 Perspectives

2.1 Security Perspective

Security is a critical aspect of any online platform, especially when it involves handling sensitive user data and personal information. In the case of "Cheap Effort" an online web-based recipe platform, ensuring data security is paramount to protect user privacy, prevent unauthorized access, and maintain the integrity of the platform. This document explores the security measures and countermeasures that should be implemented to safeguard the "Cheap Effort" platform and its users.

2.1.1 Authorization and Access Control

To maintain data security, the platform should enforce strict authorization and access control mechanisms. Only authorized users, such as registered members and administrators, should have appropriate access levels. This can be achieved through robust authentication methods, such as username and password combinations, two-factor authentication, or even more advanced techniques like biometric authentication.

2.1.2 Protection of Sensitive Information

"Cheap Effort" handles sensitive user information, including emails, names, and surnames. To protect this private data, encryption techniques should be employed both for data at rest (stored on disk) and data in transit (exchanged over the network). Encryption ensures that even if unauthorized access occurs, the data remains unreadable and unusable.

2.1.3 Auditing and Accountability

To counter the threat of data bribery or unauthorized access by administrators, an audit log should be implemented to track data access and modifications. This log should record details such as who accessed the data, when, and what changes were made. Additionally, strict controls and limited privileges should be enforced for administrators to ensure they can only access the necessary data and functions.

2.1.4 Strengthening Database Security

To mitigate online attacks and protect against unauthorized access, the database should be secured using robust security measures. This can include implementing secure coding practices, regularly patching and updating software, and conducting security audits and vulnerability assessments. Furthermore, firewalls should be employed to safeguard the platform against external threats.

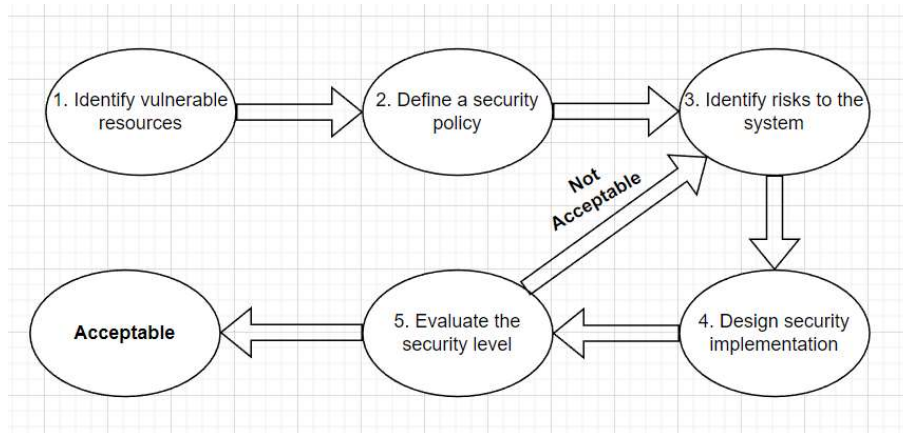
2.1.5 Data Backup and Recovery

Backups are crucial for ensuring data availability and protection against data loss. It is recommended to encrypt the data in the database backups to maintain their confidentiality. However, careful consideration should be given to the performance impact of encryption and the maintenance of availability during the backup and recovery processes.

2.1.6 Protection of Private Information

To conceal private information, such as usernames and surnames, techniques like hashing can be utilized. Hashing transforms the original information into irreversible, unique values, making it extremely difficult to reverse-engineer the original data. Additionally, other security measures, such as implementing proper access controls, can further enhance the protection of private information.

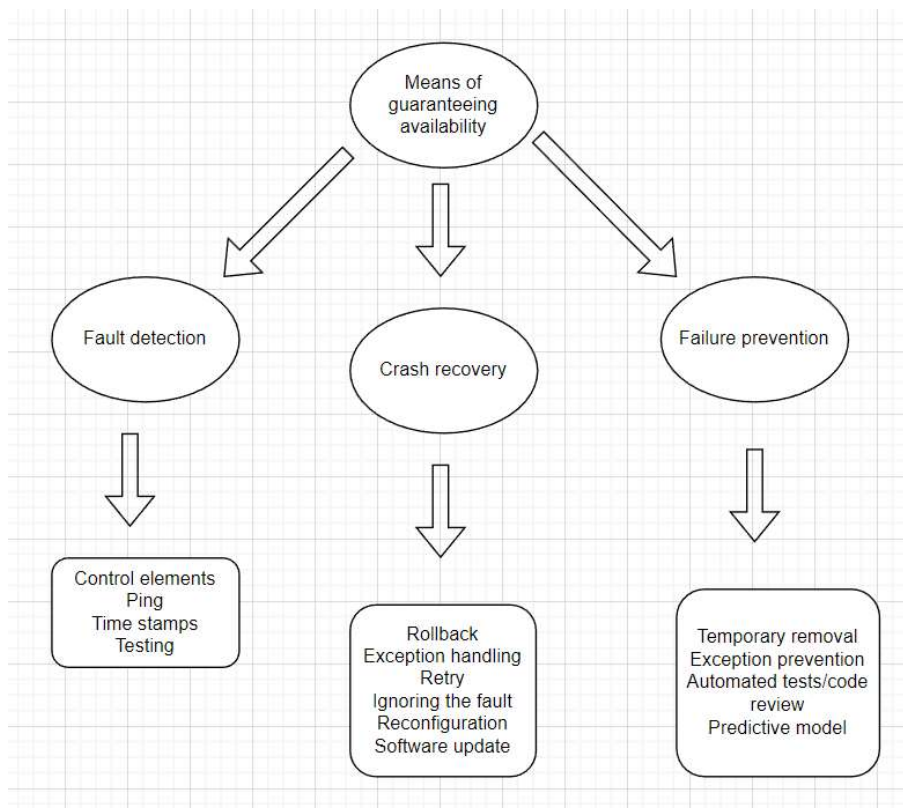
Data security is of paramount importance for the "Cheap Effort" recipe platform to ensure the protection of user information, prevent unauthorized access, and maintain the integrity of the system. By implementing robust authentication, encryption, access controls, auditing mechanisms and other security measures, the platform can provide a secure environment for users to explore recipes, share their own, and interact with the community, instilling trust and confidence in the platform's security practices.



6 Fig. Cybersecurity audit chart

2.2 Availability perspective

In the context of the "Cheap Effort" recipe platform, the availability perspective plays a crucial role in ensuring that users can access and utilize the platform without interruptions. Failures and downtime can greatly impact user experience and satisfaction, so it is essential to employ strategies to detect faults, prevent failures, and minimize system downtime.



7 Fig. Means guaranteeing availability chart

2.2.1 Fault Detection

To detect faults and potential failures, the "Cheap Effort" platform implements various mechanisms:

2.2.1.1 Control Elements

These elements actively monitor critical processes, such as CPU usage, memory utilization, and I/O operations, to identify any anomalies or performance issues.

2.2.1.2 Ping

Asynchronous requests and response messages are used within the system to evaluate accessibility and identify potential connectivity or availability problems.

2.2.1.3 Time Stamps

When an incorrect sequence of events is detected, time stamps are used to measure the duration between these events, aiding in identifying the root cause of failures.

2.2.1.4 Testing

Team members create comprehensive tests to detect possible failures and ensure the robustness of the platform.

2.2.2 Crash Recovery

In the event of a failure, the "Cheap Effort" platform employs several strategies for crash recovery.

2.2.2.1 Rollback

The system can be restored to the last known stable state by reverting to a previous backup.

2.2.2.2 Exception Handling

Important information about the failure is captured, allowing the program to be restored to a working state while preserving user data.

2.2.2.3 Retry

Assuming the failure is temporary or isolated, failed operations can be retried to achieve successful execution.

2.2.2.4 Ignoring the Failure

In cases where the failure is due to a known and verified source, it can be blocked, preventing further impact on the system (a proven approach against DDoS attacks).

2.2.2.5 Reconfiguration

Responsibilities can be redistributed among operating resources to ensure continuity and minimize the impact of failures.

2.2.2.6 Software Update

Regular software updates can address identified vulnerabilities, enhance system stability, and improve overall availability.

2.2.3 Failure prevention

The "Cheap Effort" platform focuses on proactive measures to prevent failures and maximize availability:

2.2.3.1 Temporary Removal

Perishable resources, such as unstable or problematic components, can be temporarily removed from the system to prevent cascading failures.

2.2.3.2 Continuous Automated Tests and Code Review

Ongoing automated tests and rigorous code review processes help identify potential failures and vulnerabilities early in the development lifecycle.

2.2.3.3 Exception Prevention

Best practices, such as avoiding null pointer exceptions and out-of-bound access, are implemented to prevent common programming errors that can lead to failures.

2.2.3.4 Predictive Model

By analyzing various metrics, such as program performance speed and process status, a predictive model can be used to anticipate impending failures and take proactive measures to mitigate their impact.

By considering fault detection, crash recovery, and failure prevention from an availability perspective, the "Cheap Effort" recipe platform aims to provide a reliable and uninterrupted experience for its users. These strategies ensure that potential failures are addressed promptly, downtime is minimized, and the platform remains accessible and responsive for users to view, review, and share recipes effortlessly.

2.3 Internationalization perspective

The "Cheap Effort" recipe platform plans to expand language support beyond English, catering to popular and widely used languages. It will also ensure compatibility with various foreign character sets. The platform's development timeline allows for adequate resource allocation to support internationalization efforts. Regular updates will be made to maintain and improve language support. The platform will utilize widely available technologies and standard practices without requiring specialized technical skills or software. The goal is to make the platform accessible to a global user base and foster inclusivity.

2.4 Performance and extensibility perspective

2.4.1 Response time

Response time is the time required for a specific interaction with the system to occur. It takes into account how quickly the system responds to daily workloads such as interactive user requests. At a load of 1000 users, certain processes may take:

1. Registration, login - 2 seconds.
2. Viewing a recipe - 6 seconds.
3. Recipe creation - 3 seconds.
4. Leaving feedback - 2 seconds.

The time in seconds is calculated based on the number of requests and the complexity of a given one during the process. This time can be improved by properly optimizing the database and its queries, also with better indexing.

2.4.2 Scalability

Most systems can experience some form of increased load. Scalability is the system's ability to handle this increased workload, which may be caused by an increase in the number of requests, transactions, messages, or jobs that the system needs to process per unit of time, or an increase in complexity. One of the main objectives of the system should be to maintain maximum performance at the highest possible load. Our goal is to maintain the maximum performance of the application for at least 100 users who use the application at the same time.

3 Architecture styles and patterns

In software engineering, architectural styles and patterns are a common solution to problems that can arise due to messy code logic. The main goal of the architecture style is to understand how the main components of the application interact with each other and how data flows through the system. A single application under development can have multiple architectural styles or patterns to optimize the development process and facilitate future code base growth. These styles help to use popular principles such as YAGNI, KISS, DRY, etc. In addition, it enables the entire team to work smoothly towards the same goal without breaking each other's changes. Now, let's dive into the architectural styles of our application.

3.1 Layered Architecture

Layered architecture pattern is one of the most common architectural styles. The idea behind it is that modules or components with similar functionalities are organized into horizontal layers. Therefore, each layer performs a specific role within the application. This architecture style does not have a restriction on the number of layers the application can have. Typically, there are 4 layers: presentation, business, persistence, and database layers. In our case, different folders represent different layers: Models for database models, Services for business logic and persistence layers, Pages for the presentation layer.

3.2 Model-View-Controller (MVC)

It is a software design pattern that divides the related program logic into three interconnected elements: Model, View, and Controller.

Model is the central component of the pattern. It defines the data structure that the user interacts with. This data is flowing through views and controllers.

Views are used for presenting the user interface. Users interact with views and then trigger the controllers to manipulate the models accordingly.

Controllers receive input and convert it to commands for the models or views. They manipulate the data through the models and provide results through views for further interaction.

This pattern is popular for designing web applications. That's why it is utilized in our application. However, the model and controller code are embedded within the page itself due to the nature of Razor pages technology.

3.3 Repository Pattern

It is a design pattern that isolates the data layer from the rest of the application. It is a technique for achieving Inversion of Control between classes and their dependencies and is used for data access. In our application Repositories are Services that contain interfaces and methods that interact with the data storage. In our startup file the dependency injection for repository pattern is configured where the services are registered with their corresponding interfaces. By registering these services, the DI container will be able to resolve and inject the required implementation whenever they are required throughout the application.

3.4 RESTful Design

REST (Representational State Transfer) is an architectural style commonly used for designing web services, it uses HTTP protocol. In our application there are no external clients since we use Razor Pages. However, Pages folder contain methods that adhere to RESTful principles. In our razor pages there are http methods such as POST and GET.

4 Testing

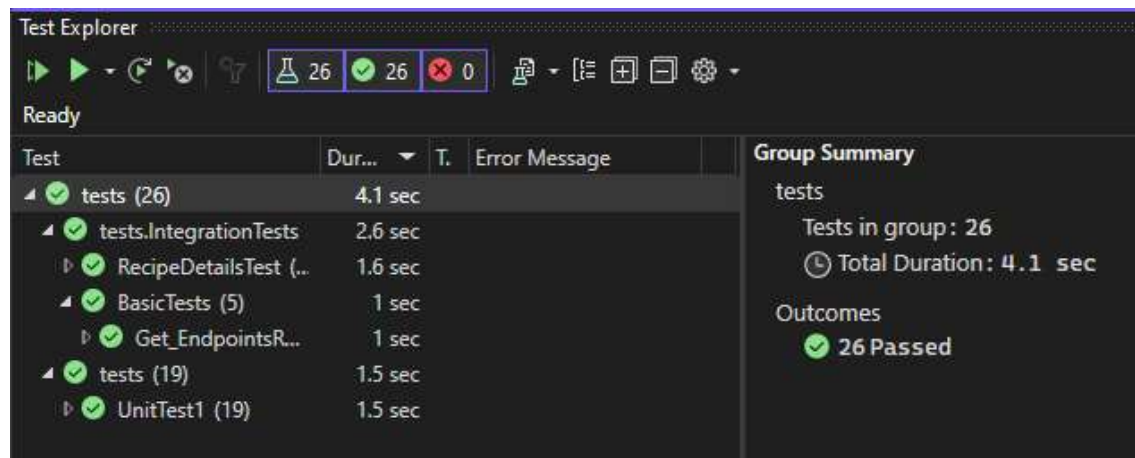
We have incorporated unit and integration tests into our program to improve its quality, lower the number of problems released to production, and speed up software development.

4.1 Testing Environment

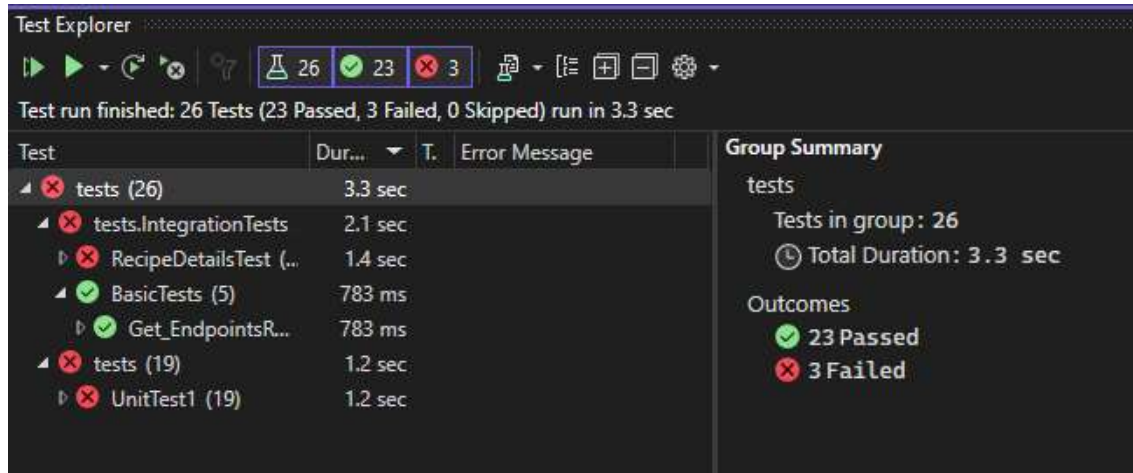
We have used the xUnit package to implement automated testing because it allows for a wider range of functions and methods of testing software than the vanilla.NET framework does. Using NuGet package manager, we can add the necessary package and begin testing.

4.2 Unit and integration tests

To make software development more reliable and feel more confident about the written code, we have implemented unit and integration tests. Unit tests allow us to test and examine how every small function works independently. However, since everything is interconnected and dependent upon one another, unit tests cannot guarantee that our software will operate as intended. Integration tests are helpful in preventing this issue.



8 Fig. Passed tests view in Visual Studio IDE

































9 Fig. Failed and passed tests view in Visual Studio IDE

Figures 8 and 9 above show that if our application performs as expected by the unit and integration tests, we get a message that the tests passed. However, if, in the future, when changing the logic of the code, accidentally a part of the application breaks, the tests would indicate this by redirecting to the test that has failed.

4.3 CI/CD

To make automated testing and building for every code change, we have implemented CI/CD pipeline.

For Continuous Integration (CI) and Continuous Deployment (CD), we used the GitLab CI/CD tool. The pipeline is set up to build code and execute tests for the application after each new merge request. The third release stage currently only notifies if all tests pass and the code compiles. The change cannot be merged with the main code if any process, such as the tests or the application code, fails.

Status	Pipeline	Triggerer	Commit	Stages	
passed	#6090 latest		Pmaster → dc898d0d Merge branch 'feature' into 'mas...	  	17 hours ago
failed	#6089		Pfeature → 3d038eea merge conflict fix	 	3 weeks ago
failed	#6086		Pgitlabssh/ma... → 9b1d7c49 merge feature	 	3 weeks ago
passed	#5952		Pmaster → 20d59fbb Update .gitlab-ci.yml	  	00:03:12 1 month ago
failed	#5951		Pmaster → 20d59fbb Update .gitlab-ci.yml	  	00:00:10 1 month ago
failed	#5935		Pmaster → 20d59fbb Update .gitlab-ci.yml	  	1 month ago
passed	#5931		Pmaster → 20d59fbb Update .gitlab-ci.yml	  	00:01:36 1 month ago
passed	#5929		Pmaster → 20d59fbb Update .gitlab-ci.yml	  	00:01:38 1 month ago

10 Fig. CI/CD pipelines progress view in GitLab

```

306 info: Microsoft.EntityFrameworkCore.Update[30100]
307     Saved 2 entities to in-memory store.
308 info: Cheaper_Effort.Middlewares.DateLogMiddleware[0]
309     Date: Tuesday, 23 May 2023
310 info: Cheaper_Effort.Middlewares.DateLogMiddleware[0]
311     userAgent: not found
312 info: Cheaper_Effort.Middlewares.DateLogMiddleware[0]
313     ipAddress: not found
314 info: Cheaper_Effort.Middlewares.DateLogMiddleware[0]
315     url: /RecipePages/RecipeDetails
316 Passed! - Failed:    0, Passed:    26, Skipped:    0, Total:    26, Duration: 728 ms
318 Cleaning up project directory and file based variables
320 Job succeeded

```

11 Fig. CI/CD testing stage output view in GitLab

Figure 11 above shows the output after a pipeline finishes the testing stage. It provides information about passed and failed tests and duration testing.

5 Traceability

This is traceability matrix for Cheap Effort application.

Viewpoints	Perspectives			
	Security	Availability	Internationalization	Performance
Functional				
Information				
Development				
Context				
Deployment				