UNIVERSITY OF STAVANGER, NORWAY

DAT240 Project Report: Image Guessing Game

Malin Larsson, Matias Lyngnes Ramsland, Roger Bærheim,
Elen Pedersen, Siren Melkevig, Vebjørn Lia Riiser

November 17, 2022

CONTENTS

1 Introduction

The goal of this project was to develop an *Image Guessing Game* where the user should recognize images. The images that the player should recognize are divided into multiple segments where all except one starts as hidden. The objective of the game is to guess what is portrayed in the image with as few segments of the image revealed as possible. The two main roles of the game is "Guesser" and "Oracle". The "Guesser" will guess the name of the image that is shown on the screen, and "Oracle", which is a bot, will select which segments of the image that will be shown to the "Guesser".

We opted to allow the user to upload new images, which can be sliced manually by the user or automatically by our system. In addition we started implementing multiplayer, but only had time to lay the foundation for it.

In this report we will give a more detailed explanation on how we chose to create the game, as well as a reflection of our work.

2 Working process

In this chapter, we will explain the process we used for organizing and working on this project.

We started the project by voting for the task we wanted to do. The majority of the group voted for the *Image Guessing Game*, where most of us felt like a new challenge would be interesting. After deciding which project to work on, we set fixed times for our group meetings. We agreed to meet every Monday and Thursday.

2.1 Organization of our collaboration

For this project, we chose to use GitHub as our main collaboration tool. GitHub gave us access to multiple tools that helped us track and plan our work efficiently, such as GitHub Projects, GitHub Issues, and Kanban Board.

We chose to assign one group member the role of Scrum Master. The Scrum Master was in charge of assigning issues, creating new detailed issues in the group meetings, and having a general overview of every group member's progress.

2.2 Split of work

Before we started the implementation, we divided the game into different domains which in turn were split into smaller tasks that were needed to create the domains. While shaping these tasks, we aimed to create a minimum viable product (MVP). A minimum viable product means that we wanted to get a working product before we added more advanced features.

2.3 Track of work

To keep track of our work, we used GitHub Issues combined with Kanban Board. All issues were put on to our Kanban Board. The board had different columns to express the status of every issue. Our Kanban Board had the following columns:

- Backlog
- Priority
- In Progress
- On Hold
- Testing
- Awaiting Review
- Done

Expressing the status in this way gave us an organized overview of all the tasks, made it easier to understand what needed to be done, and gave us the ability to keep track of what each group member was working with. Every issue had its own branch, which made it easier to merge changes and discard unneeded or wrong changes to the code. Merges into the main branch were made by submitting a pull request, which had to be reviewed and approved by at least one other group member. We tried keeping commits small to make the pull requests easier to review and to minimize the risk of changing the same piece of code.

2.4 Technical Decisions

2.4.1 Languages

We wanted to use the languages that all group members were familiar with. Therefore, we opted to use C# for backend with the ASP.NET RazorPages library to create the dynamic web pages. We mainly used CSHTML for our frontend, as well as a small amount of JavaScript for some of the more complex frontend aspects of the game.
We used Bootstrap for the CSS styling of our game.

2.5 Communication

While working on this project, we used multiple forms of communication. The most valuable form of communication we have been using is face-to-face communication, such as in our weekly meetings. This made it possible to cooperate, share ideas, and make decisions together as a group. Essentially, these meetings gave us a way to build team spirit and motivate each other.

Between the agreed meetings we have used social media applications such as Discord and Messenger. Here we have had the opportunity to help each other and discuss small cases. If more complex problems or questions arose, we naturally saved them for the meetings.

2.6 Evaluating during the project

2.6.1 How?

To analyze the progress of our project, we have continuously factored in the remaining time and resources. When we started the project, we set our own deadlines. Having clear deadlines was a great way to determine whether we were on track with the project, or not.

2.6.2 What worked?

During the period of creating this project, our communication abilities made it easier for us to avoid creating similar codes and replicating other group members' work.

Our weekly meetings helped us stay on track with the project's progress. In these meetings, we also set weekly goals, which made it easy for us to structure our work for the week. If a task turned out to be very time-consuming, complex, or urgent for future progress, we would assign more resources, such as group members, to the task.

By using Domain Driven Design we were able to create well-detailed and smaller issues, which made it easier to keep tasks separate and avoid code conflicts. In addition, smaller issues helped motivate developers when a task was completed by giving a dopamine rush.

2.6.3 What did not work?

Throughout the project different kinds of complications occurred and some unproductive choices were made.

The first mistake we did was during our first meeting in the process of making issues. We created issues in pairs, instead of together as a complete group. This complicated the issues, due to not considering the other issues made by other pairs. This resulted in issues and tasks overlapping. We solved this problem by creating new issues that were smaller and more specific, all together as a group. At this point we also assigned one group member as the Scrum Master, see section 2.1.

One of the most inefficient choices that were made was not writing tests simultaneously to the code being implemented, despite our lecturer strongly recommending it. We started to think about and plan the testing from the very beginning, but did not put enough resources into this before the last part of the project. This made it more difficult and time-consuming to ensure that our code were performing as expected. Looking back, we would have definitely written tests while implementing code.

We planned on implementing two of the advanced features; multiplayer and uploading new images. We started implementing the multiplayer feature about halfway through the project. We changed the class that kept track of the game state to support multiple players, and then successfully added SignalR to our project. We assumed that it would be quick to replace the post and get methods with SignalR. However, it was not easy to transfer the data with SignalR because of some logic being done in the frontend. For instance, we had to use

JavaScript instead of razor templates when displaying the overlapping segments.

(See 2.1).

```
@for (var i = 0; i < Model.Images.Count; i++)
{
    @if (i == 0)
    {
        <img style="position: relative; top: 0; left: 0;" src=" @Model.Images[i]">
    }
    else
    {
        <img style="position: absolute; top: 0; left: 0;" src=" @Model.Images[i]">
    }
}
```

Figure 2.1: Code that show how the segments are placed on top of each other using a for loop in the cshtml file and styling with css

As soon as the backend part of the multiplayer feature was completed, we realized that this feature would be more time-consuming than expected. We would have to reimplement and test a lot of logic in JavaScript instead of using razor templates. We also would have had to find another way of showing the segments, taking in user guesses and determining what should be displayed to the user.

We also tried using Razor and Blazor together. Since Blazor's and Razor's templates are very similar we would not have to change any logic, other than using SignalR to send the objects to Blazor. This is because Blazor handles the updates on the client, and not in backend. However this was to time consuming regarding the projects deadline.
We should have investigated better how SignalR can be integrated with Webassembly. We also should have put more of the logic of what should be displayed on the page in the backend. If we had done this from the very beginning, we would have been able to present our project containing the multiplayer.

3 Design

3.1 Domain Driven Design

Before starting the implementation of the project, we as a group decided to use Domain Driven Design for our project's structure. The reasoning for this decision is the experience we all had from the previous lab assignments of this course. We decided that we wanted to divide our project into four different domains, which were History, Image, Oracle, and User. This structure made the project well organized, helped separate concerns, and simplified the planning and implementation of the different functionalities of the game.

The History domain was in charge of retrieving history information, such as our leader board and recent games for a specific user. The Image domain was in charge of functionalities for handling and processing an image. The Oracle domain was in charge of different functionalities regarding the actual game, such as checking whether the user's guess is correct or not.

3.2 FLOWCHART

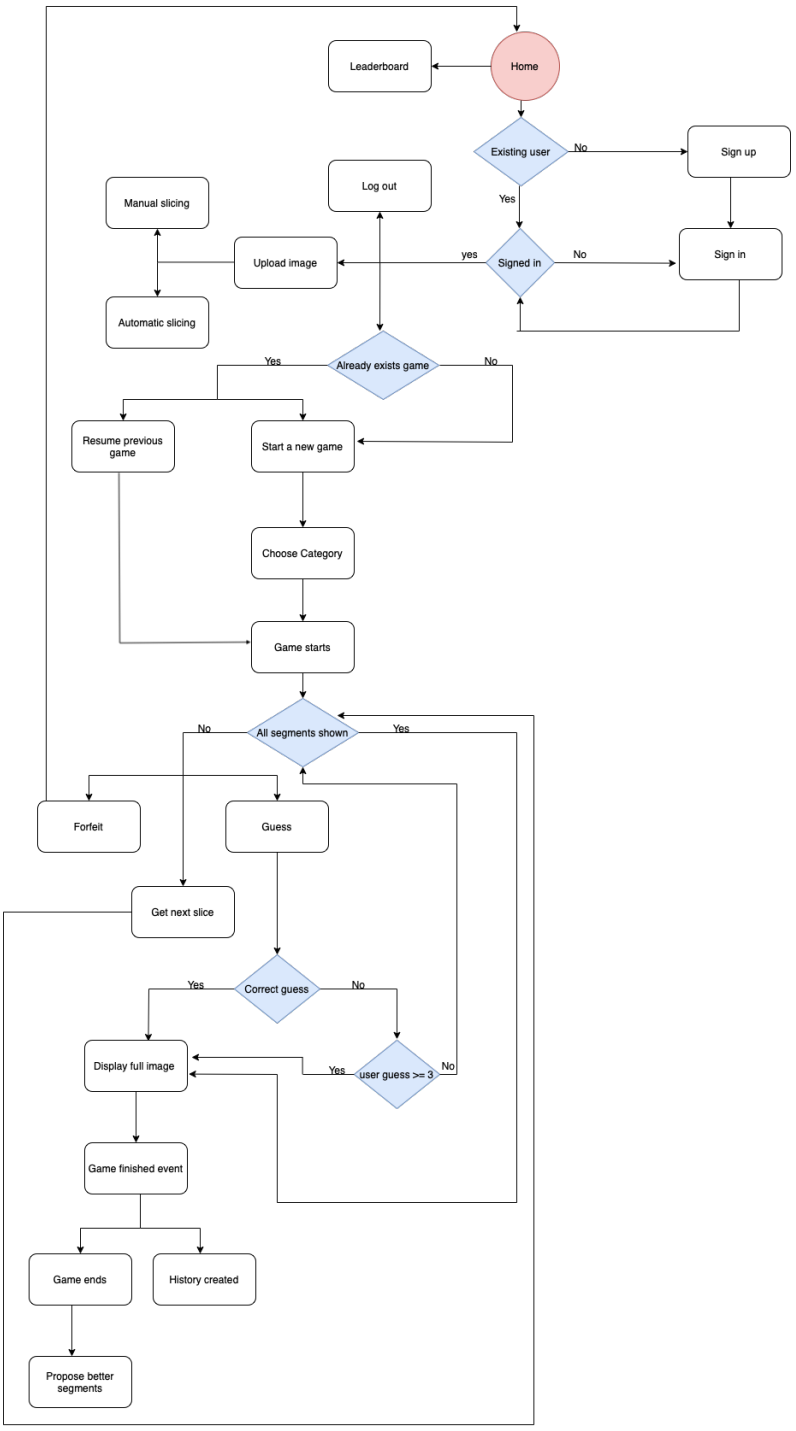

Figure 3.1: Flowchart

3.3 Block Diagram

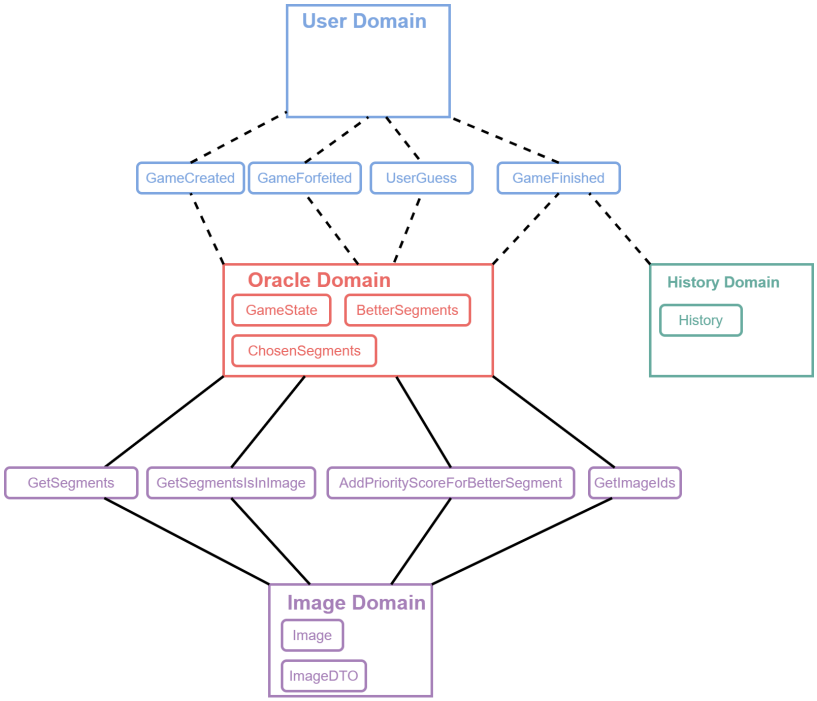

Figure 3.2: Flowchart

4 Backend

4.1 Dependency injection

We have used dependency injection which is a design pattern that allows an object to get the services and objects that it depends on. We added the identity service, oracle service, history service, and image service to our dependency container.

4.2 MediatR

To maintain Domain Driven Design for our project structure, we have used a mediator, which is in charge of the interactions between the domains. The goal of the mediator is to let the domains communicate without a direct dependency on each other.

4.3 ImageSharp

Imagesharp is a cross-platform .NET library for manipulating 2d graphics[2] (images). We used this library to split the images into segments both automatically and manually. Im-

ageSharp was used as an alternative to the system.drawing library that is only supported on Windows, since we wanted to run our project in a docker container with Ubuntu.

4.4 EntityFramework Core

Entityframework Core (EF) is a library for interacting with databases using an "object-relational mapper"[1]. This means that the developer does not need to write SQL queries or know the details of how objects are retrieved. The details of retrieving data are hidden by EF Core's abstractions so that the developer only needs to interact with native .NET classes. EF Core also simplifies the creation of the database for developers. EF Core is "code first", which means that EF translates the class definitions that developers create into actual databases. A big advantage of this is that it is easy to create reproducible databases since it is always created directly from code, and not added one SQL statement at a time.

4.5 Testing

Testing is important mainly because it makes developers able to refactor code without noticing that something else breaks. Tests are usually separated into unit tests which handle one feature or function, and integration tests which handle the interaction between units. As mentioned in 2.6.3, we did not put enough resources into testing before the last part of the project. We decided to test the most important pipelines and methods in the services of the different domains. We structured the test file by dividing the different tests for each domain into their own folder, which makes it organized and user-friendly. However, we encountered a redundancy error while trying to run all tests simultaneously. This was solved by putting all tests in the same collection.

5 Frontend

5.1 JavaScript

We used JavaScript to let the user paint on a canvas for the manual slicing and to let the user click on image-segments to propose better, more helpful segments for future "guessers". We used the jQuery library to simplify sending the mouse position to the backend. jQuery was also used when trying to implement SignalR.

5.2 CSHTML

As explained earlier in the rapport, we should have chosen to do a proper frontend. However, from the beginning we decided to only use CSHTML, with the exception of a little implementation in JavaScript, see section 5.1. The reason we had for making this decision was our lecturer specifying that this course was not a frontend-course, but a backend-course. We also

made this decision due to the fact that the only frontend experience we had from the previous labs in this course, were CSHTML. However, if we were to do this project all over again we would have used VUE as our frontend.

6 Summary and Contributions

6.1 Summary of the project

This project has been a valuable experience for all of us in this group. We have experienced the positive parts, as well as the challenging parts of working in a group. We saw this as a great opportunity to experience and learn what it is like to work in a team, and felt as if this project will give us an advantage in future work affairs.

One of the advantages of our group is that most group members attend school every single day. Due to this we have had access to help, advice, and ideas every day, even though we did not have meetings that day. It also made the communication easier, because in-person communication reduces misunderstandings on different choices that had to be made throughout the process.

Communication was one of the biggest challenges for our group, especially in the beginning. We had some overlap incidents and disagreements. This is something we worked on, and was able to succeed on before the project was over. This is a challenge we expected to experience, due to being a group of six individuals with different strengths and weaknesses. However, we now feel satisfied with the knowledge we gained about communication, and feel like this is a good experience to bring into our future professional lives.

6.2 Contributions

Every group member worked with both backend and frontend. We wanted everyone to feel as if they had the opportunity to do something they enjoyed doing. Group members were able to show their knowledge by doing issues and tasks they were very experienced in. Group members were also able to gain knowledge by doing issues and tasks they were inexperienced in.

References

[1] Microsoft. Entity Framework Core. Hentet fra: `https://learn.microsoft.com/en-us/ef/core/`. (Lastet ned: 16.11.2022).

[2] Six Labors. Imagesharp. Hentet fra: `https://sixlabors.com/products/imagesharp/`. (Lastet ned: 16.11.2022).