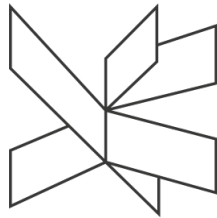


Game Renting Applicaiton



**VIA University
College**

By

Audrius Sauciunas

293156

Philip Philev

293735

Levente Nagy

293115

Eva Nikolaeva

293164

Supervisors

Troels Mortensen

Joseph Okika

Software Technology Engineering

3rd Semester

Horsens, Denmark

December 2020

This report is dedicated to *Everyone reading*

Number of Pages: 42

Abstract

The purpose of this report is to document the development of a game rental system. The system was developed as an idea to assist people who are interested in acquiring physical copies of games. The focus was directed towards creating a way for users to either post games available for renting or rent already existing elements.

The documentation presents all stages of system development. The functionality of the system was decided after analysing the main problem with already existing systems, followed by design and implementation of the system.

The software was developed to be a high availability distributed system; the development process followed the kanban framework. The result of the project is a game rental system that follows 3 tier system architecture, has heterogenous nodes, follows SOLID principles and well-known design patterns.

Keywords: *3-Tier, Board Game, Blazor, Spring Boot*

Contents

1	Introduction	1
2	Analysis	2
2.1	Requirements	2
2.1.1	Functional Requirements	2
2.1.2	Non-Functional Requirements	4
2.2	Use Case Diagram	5
2.3	Domain Model	9
2.4	Security	10
3	Design	16
3.1	Overview	16
3.1.1	Presentation Tier	17
3.1.2	Application Tier	17
3.1.3	Data Tier	17
3.1.4	Discussion on 3-tier Architecture	18
3.1.5	Summary	20
3.2	Database	20
3.3	Tier 3	21
3.4	Tier 2	21
3.5	State Diagram	24
3.6	Tier 1	24

3.7	Sequence Diagram	25
3.8	Security	27
4	Implementation	28
4.1	JavaDoc	28
4.2	Points of interest	28
4.2.1	Back-end	28
4.2.2	Frontend Implementation	30
5	Testing	35
5.1	Functional Testing	35
5.1.1	Unit testing	35
5.1.2	System testing	36
5.2	Non-functional testing	37
6	Results and Discussion	38
7	Conclusion	39
8	Future Development	41
9	References	42

List of Figures

2.1	Use Case Diagram	5
2.2	Register Use Case	6
2.3	Add Game Use Case	7
2.4	Search Use Case	8
2.5	Request Trade Activity Diagram	9
2.6	Domain Model	10
2.7	STRIDE threat model	11
2.8	Threat model	11
3.1	3-Layer Architecture	17
3.2	System Architecture based on 3-tiers	18
3.3	System Architecture Discussion 1	19
3.4	System Architecture Discussion 2	19
3.5	System Architecture Discussion 3	20
3.6	ER Diagram	21
3.7	Database class diagram	22
3.8	Class diagram of Tier-2	23
3.9	Client Class Diagram	24
3.10	Client Class Diagram	25
3.11	Sequence Diagram	25
4.1	Send Content	29

4.2	Frontend Snippet 1	30
4.3	Frontend Snippet 2	31
4.4	Frontend Snippet 3	31
4.5	Frontend Snippet 4	32
4.6	Frontend Snippet 5	32
4.7	Frontend Snippet 6	33
4.8	Frontend Snippet 7	34
5.1	Nunit test for Web Services	36
5.2	Register and Login Use Cases	37

List of Tables

CHAPTER 1

Introduction

The media's attention on board and video games has done nothing less, but grow rapidly in the last decade, as new platforms, apps and technologies come to the market. With so much product on the market, people can't help but look for new and innovative alternatives to the main methods, such as trading games or renting them from gaming cafes, subscription-based sites or friends and personal connections. Although people like to exchange games of any category, there are also people looking to trade or rent out-of-stock games, games for rare consoles or even first editions of famous board games, such as Dungeons and Dragons or Warhammer. The growing community of rent-or-trade a game can be seen all over the world with Facebook groups, High School and University clubs, which is why the necessity for a Game Rental System is the purpose of this project.

The existing platforms, which fulfill these needs are, as mentioned, subscription-based websites, which trade digital copies of video games and sell used board games, thus deeming a system for local trading and renting a useful and innovative solution to the growing communities.

The conclusion is that a distributed system, that inherits the three-tier architecture is to be made for the use of people with the common passion of playing new, old and collectible video and board games with the intent to bring them closer together in an environment, that is easy-to-use and convenient.

Due to the scope of the project, there are a few things, which the system will not cover. One of them being, that currency-based payment methods will not be implemented nor condoned. In this document, the phases, which will be covered are Analysis, Design, Implementation and Testing. Each of these phases will have its own chapter, where it will be delved into a detailed explanation of what each phase consists of and how this project was realized.

CHAPTER 2

Analysis

The analysis chapter of this project delves with the establishment of a coherent problem to be solved. The first step is to present the requirements necessary to describe the system. This leads to the formulation of the use case diagrams that define the actors and actions necessary to ensure the functionality of the system. Furthermore, use case descriptions corresponding to each use case is presented with a corresponding activity diagram representing the flow of activities. The domain model subsection provides an overview of the classes in the system based on the previously described data. The chapter is concluded with an estimation of the security risks a system such as this might be exposed to and tries to find ways to mitigate such risks.

2.1 Requirements

Going through the necessary steps of the Requirement Engineering Process based on the scale of this project yields the **Functional** and **Non-Functional** Requirements for the system.

2.1.1 Functional Requirements

The requirements related to the functional aspects of the software are separated based on priority and each corresponds to certain functionality to be included in further sections. They are as follows:

Critical priority:

1. As a user I want to create an account, in order to have my information personalized.
2. As a user I want to be able to login, in order to enter my account.

3. As a user I want to add a posting for a game I own, in order to make it available for other users to see.
4. As a user I want to edit a posting for a game I own, in order to update information and/or status.

High priority:

5. As a user I want to be able to search for active listings, in order to find specific games.
6. As a user I want to update my account, in order to edit personal information.
7. As a user I want to delete a posting for a game I own, in order to remove the game from active listings.

Medium Priority:

8. As a user I want to be able to search for other users, in order to find users with similar interests.
9. As a user I want to update my account, in order to edit personal information.
10. As a user I want to rent games from other users, in order to play them.
11. As an admin, I want to review listings, in order to verify the information provided.
12. As an admin, I want to remove listings in conflict with the platform, in order to prevent malicious intent.

Low Priority:

13. As a user I want to find statistics, in order to see the most played games.
14. As a user I want to view articles, in order to get the latest information.
15. As a user I want to filter games, in order to view them based on preferences.
16. As a user, I want to rate games, in order to provide statistical information.
17. As a user, I want to rate users, in order to give feedback.

2.1.2 Non-Functional Requirements

The Requirements not related to the functional aspects of the system, but instead describes the properties of the system are the non-functional requirements as shown:

1. The system must be based on a three-tiered architecture.
2. The system will take use of the PostgreSQL, Java and CS programming languages.
3. The system will use a Graphic User Interface for each client.
4. The system will utilize SSL sockets
5. The system will utilize RESTful web services

2.2 Use Case Diagram

The Use case diagram in **Figure 2.1** is continuation of the established in previous section requirements in graphical form and includes the actors and extensions on certain use cases. Logically the diagram represents the core functionality each actor has access (but not limited) to.

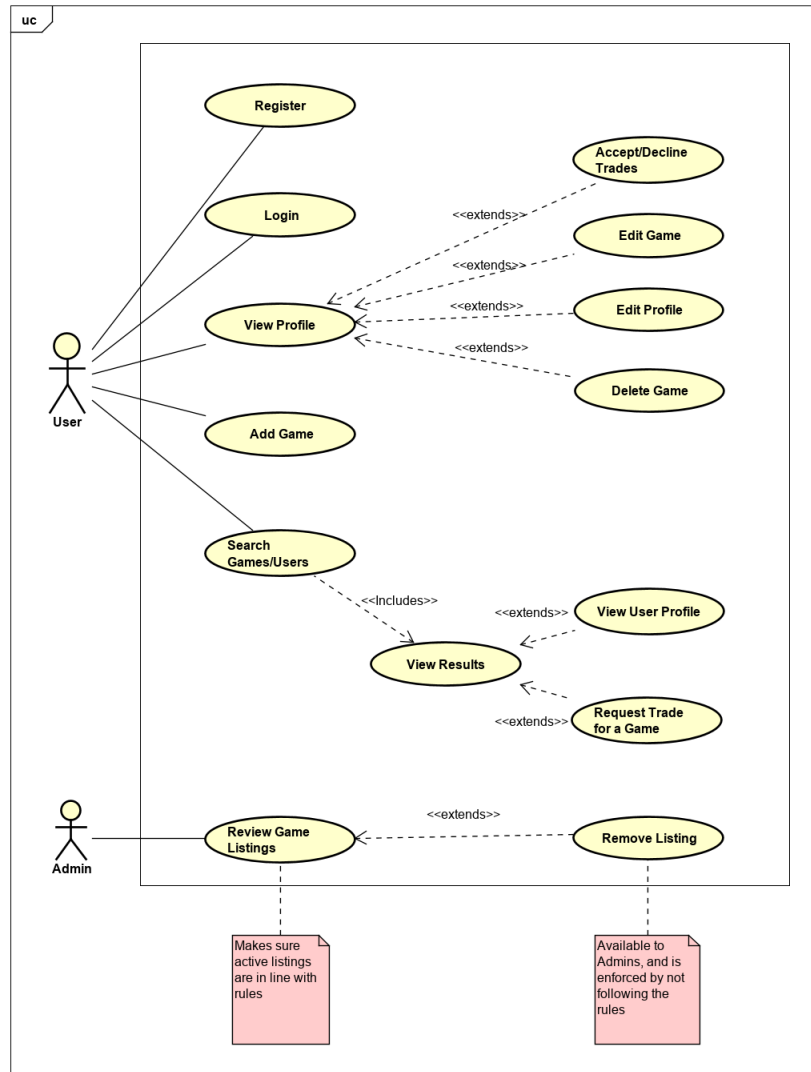
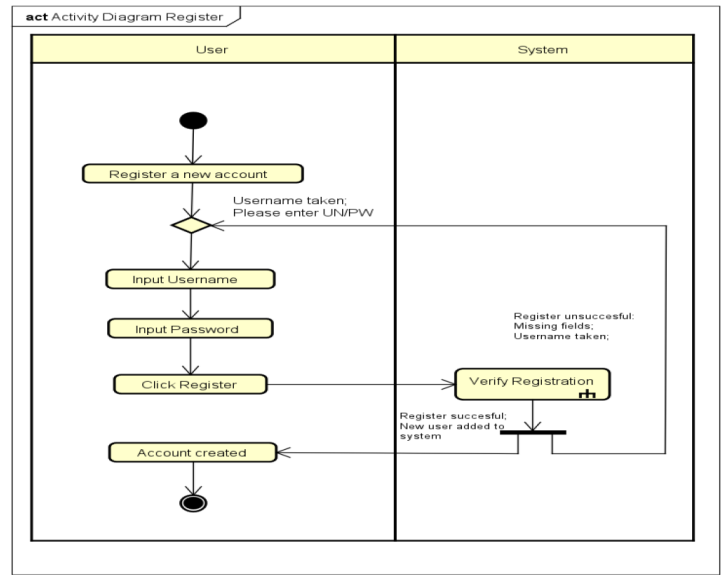


Figure 2.1: Use Case Diagram

There is a use case description corresponding to each use case that allows for a closer look into its functionality. The use case description portrays each functionality and indicates the steps the user goes through while using the system. Every use case description also has an activity diagram which represents the flow of activities during the process. In this section we describe some of the core use cases in addition to their descriptions and activity diagrams. For every use case description and corresponding diagram refer to the appendixes.

ITEM	VALUE
UseCase	Register
Summary	The User wants to register his/her own account.
Actor	User
Precondition	-
Postcondition	The User has created an account and can now login.
Base Sequence	1. User clicks Register 2. User inputs Username 3. User inputs Password 4. User clicks Register 5. Register successful
Branch Sequence	
Exception Sequence	2. a User does not input Username 3. a User does not input Password 5. a Register unsuccessful - Missing fields
Sub UseCase	
Note	The Username should not be taken

(a) UC Description for Register



(b) Activity Diagram for Register

Figure 2.2: Register Use Case

The first Use case to be discussed is the Register. An important aspect of the system is for users to create their own profile in order to trade and browse games. In **Figure 2.2a** the use case description is provided with the active steps in creating a profile.

The activity diagram associated with the registration use case offers a more detailed view on the interaction between the user and the system and potential expected scenarios. The diagram is present in **Figure 2.2b** and includes the to be realized "Verify Registration" response by the system.

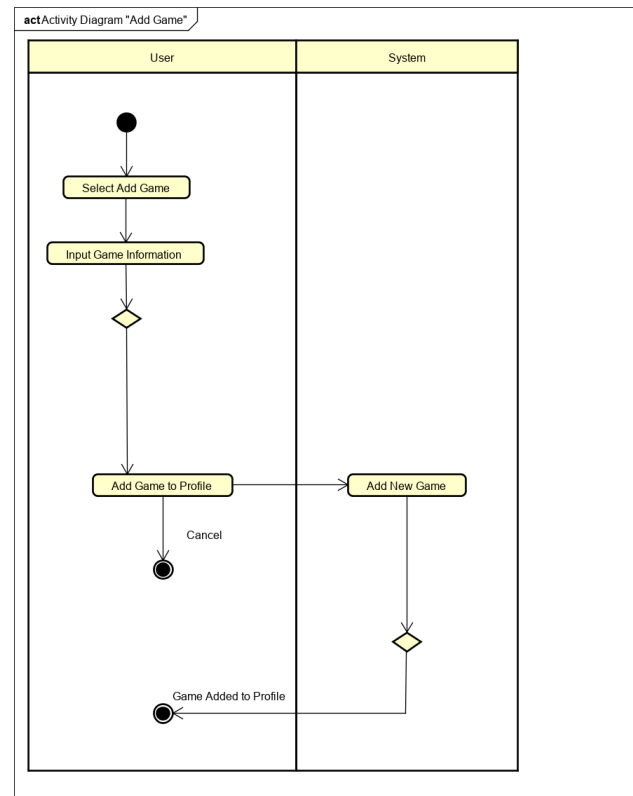
The step after creating a profile is to log in the system. This again includes some verification in order to confirm the existence of the username and a matching password. The use case description and its matching activity diagram are available in the appendixes due to their similarity to the Registration use case provided in **Figure 2.2**.

After the user has logged in the system, they are given the opportunity to select from a number of options that would provide them with the desired effect. We chose to discuss the more important Use case descriptions and Activity diagrams that would provide the core functionality of the system. As the aim is to develop a game rental system, the core options recognized are the addition of games, the ability to browse and search for a specific preferences and the capability to review, accept or decline offers for listings.

Clearly, to add a game and/or make a listing certain criteria have to be met. Firstly, the option has to be available for selection by users. Secondly, they have to be able to provide relevant information about

Use Case	Add Game
Summary	The User wants to add a game to his/her collection.
Actor	User
Precondition	The User must have an account.
Postcondition	The User has added a game to his/her collection.
Main Scenario	<ol style="list-style-type: none"> 1. User clicks on Add Game. 2. User inputs game name, genre, type, description, price 3. User clicks Add Game Button.
Alternative Scenario	3.a User clicks Cancel.

(a) UC Description for Add Game



(b) Activity Diagram for Add game

Figure 2.3: Add Game Use Case

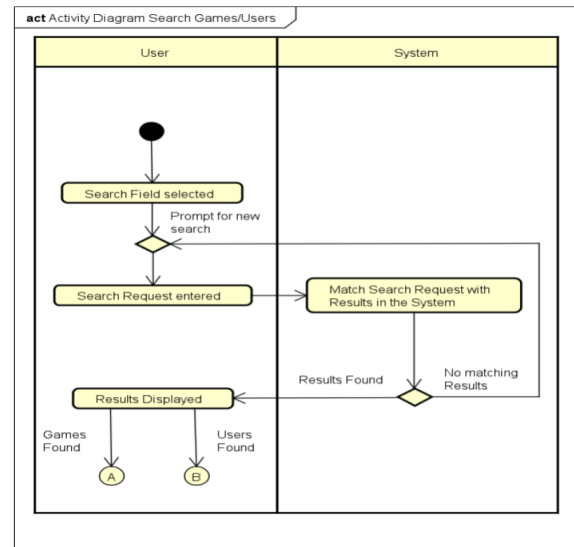
the game (f.x name, genre, type etc.). Finally they should be able to make it an active listing that could be discovered or keep simply appended to their profile information. This can be seen in **Figure 2.3**. More specifically the Use case description in 2.3a and the Activity diagram in 2.3b.

It can be seen in **Figure 2.3b** that depending on the status of the listing, the system will allow the game to be discovered and add it to the user's profile.

In order to discover games, a search function has to be available to the users. When a user performs a search matching or similar results should be provided. The results can be game listings, which can be modified to, for example, search games for a specific console. Clarification is provided in **Figure 2.4**.

ITEM	VALUE
UseCase	Search Games/Users
Summary	The User wants to search an existing game or user account
Actor	User
Precondition	The User must have an account
Postcondition	A list of wanted results is visible
Base Sequence	1. User clicks on the "Search" field 2. User inputs wanted results 3. User clicks "Search" 4. The search list view is visible.
Branch Sequence	
Exception Sequence	3. a User does not click Search 4. a List unavailable - Search data does not match any results
Sub UseCase	
Note	

(a) UC Description for Search



(b) Activity Diagram for Search

Figure 2.4: Search Use Case

The Activity Diagram in **Figure 2.4b** further expands the concept. If no matching searches are present in the system's data, the list would appear empty, prompting the user to search with different criteria or make a listing of his own for desired category. If any matches are found the user can select one and proceed to either view a the game's detailed information or request a trade for a game.

The case where the user desires to request a trade is elaborated in **Figure 2.5**. As it can be seen, once a user selects a game from the available searches, a "Request Trade" option should be available. What follows is a navigation to the user's rented games, where the rent dates and more information can be seen.

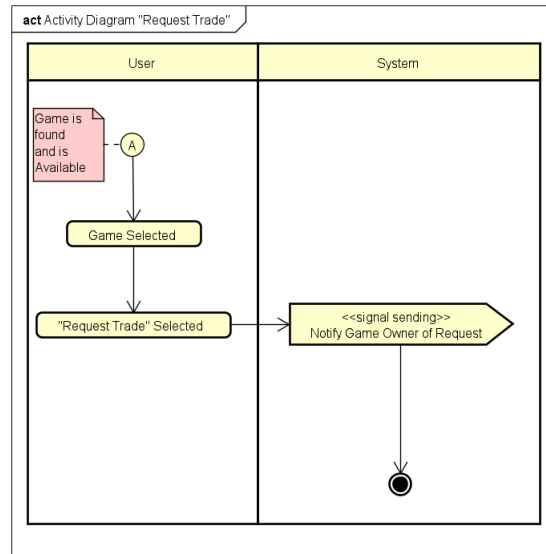


Figure 2.5: Request Trade Activity Diagram

In this section we described our initial approach to structuring the system using the Use Case Diagram based on the provided requirements. The focus has been on the more relevant parts of the core functionality. For more information regarding the remaining requirements and use cases refer to the materials present in the Appendix G.

2.3 Domain Model

As the main points of the functionality were covered in the previous section, we could move forward and establish the conceptual model and visual representation of the objects in our domain of interest. Deriving from the information of the Use Case diagram we can establish the core objects necessary to achieve such a system. The core objects can easily be identified as the **Users** the main actor of the core functionality and **Games** as the subject of interest and exchange. Nevertheless, both require additional objects to cover the previously discussed functionalities. **Figure 2.6** provides a visual representation of the Domain of interest.

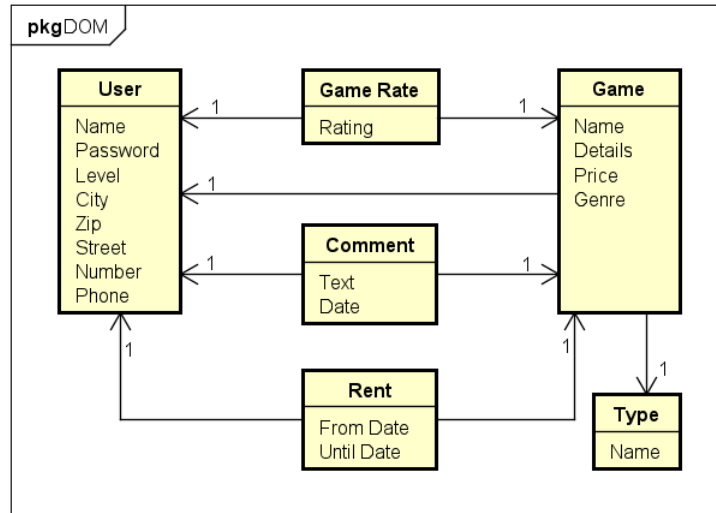


Figure 2.6: Domain Model

2.4 Security

Security is a vital part in any application. The internet has always been a place where users could exchange information and ideas, but in today's internet where treats and malicious users lurk everywhere we need to be sure that each bit exchanged with our servers is authentic. Thus authentication, authorization and encryption are crucial for the longevity and peace of mind of our users. In this Section the concept of security and potential risk will be further expanded, while in the end of Chapter ?? a more focused approach on the available security mechanisms will be discussed. In order to analyze the system for potential security treats, an abstract tread model should be created. That is, the focus will be on identifying and addressing potential vulnerabilities assosicated with Web Applications.

Threat modeling is a way of viewing not just software but the world. It is ever changing and allows to explore what our systems are and how they work Web applications have inherent vulnerabilities due to the surface of distribution fx. the web server, network infrastructure, data storage, web services etc. Formalized by Microsoft in the late 90s, as part of the Security Development Lifecycle, STRIDE (Spoofing, Tampering, Repudiation, Information disclosure, Denial of Service, Elevation of privileges) is a threat mitigating strategy that asseses risk in the system.

Property	Threat	Definition	Example
Authentication	Spoofing	Impersonating something or someone else.	Pretending to be any of billg, microsoft.com or ntdll.dll
Integrity	Tampering	Modifying data or code	Modifying a DLL on disk or DVD, or a packet as it traverses the LAN.
Non-repudiation	Repudiation	Claiming to have not performed an action.	"I didn't send that email," "I didn't modify that file," "I <i>certainly</i> didn't visit that web site, dear!"
Confidentiality	Information Disclosure	Exposing information to someone not authorized to see it	Allowing someone to read the Windows source code; publishing a list of customers to a web site.
Availability	Denial of Service	Deny or degrade service to users	Crashing Windows or a web site, sending a packet and absorbing seconds of CPU time, or routing packets into a black hole.
Authorization	Elevation of Privilege	Gain capabilities without proper authorization	Allowing a remote internet user to run commands is the classic example, but going from a limited user to admin is also EoP.

Figure 2.7: STRIDE threat model

In **Figure 2.7** the matching security property is described by a certain security threat in the STRIDE model. In order to apply the model to the specific system, software such as the Microsoft Threat Modeling Tool could be used once the structure of the system is known. It can point potential threat based on the servers, clients and services in order to implement a potential mitigation strategy. [1]

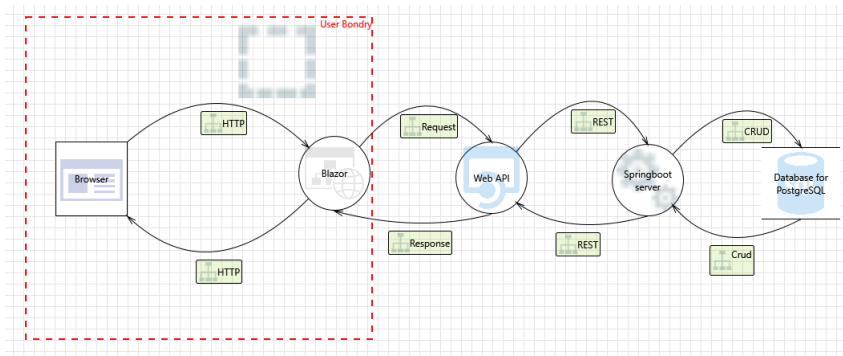


Figure 2.8: Threat model

With the current analysis of the system we can derive the existence of both external and internal risks. External threats based on the STRIDE, such as SQL injections or DoS attacks are a potential risk. Furthermore, internal threat can pose a risk as password can be compromised or an admin might expose information not meant to be accessed by users. Shown in Figure 2.8 is the mentioned model. It can be seen that due to the infrastructure of the system several common threats can affect the operation of the

system based on the STRIDE categories. Analysis on the said threats is performed in below:

1. Tampering: Integrity

- (a) SQL injection attack in which malicious code is inserted into strings that are later passed to an instance of SQL Server for parsing and execution. The primary form of SQL injection consists of direct insertion of code into user-input variables that are concatenated with SQL commands and executed. A less direct attack injects malicious code into strings that are destined for storage in a table or as metadata. When the stored strings are subsequently concatenated into a dynamic SQL command, the malicious code is executed
 - i. Mitigation can be achieved through ensuring that type-safe parameters are used in Web API for data access.
- (b) An adversary may inject malicious inputs into an API and affect downstream processes
 - i. Mitigation can be achieved through implement input validation on all string type parameters accepted by Web API methods.
- (c) An adversary can gain access to the config files. and if sensitive data is stored in it, it would be compromised.
 - i. Mitigation can be achieved through encrypting sections of Web App's configuration files that contain sensitive data.

2. Elevation of Privileges: Authorization

- (a) Failure to restrict the privileges and access rights to the application to individuals who require the privileges or access rights may result into unauthorized use of data due to inappropriate rights settings and validation.
 - i. Ensure that administrative interfaces are appropriately locked down.
 - ii. Ensure that proper authorization is in place and principle of least privileges is followed.
 - iii. Business logic and resource access authorization decisions should not be based on incoming request parameters.
 - iv. Ensure that content and resources are not enumerable or accessible via forceful browsing.
- (b) An adversary may gain unauthorized access to Web API due to poor access control checks
 - i. Implement proper authorization mechanism in ASP.NET Web API.
- (c) An adversary can gain long term, persistent access to Database for PostgreSQL instance through the compromise of local user account password(s).

- i. It is recommended to rotate user account passwords (e.g. those used in connection strings) regularly.

3. Spoofing: Authentication

- (a) The session cookies is the identifier by which the server knows the identity of current user for each incoming request. If the attacker is able to steal the user token he would be able to access all user data and perform all actions on behalf of user.
 - i. Set up session for inactivity lifetime.
 - ii. Implement proper logout from the application.
 - iii. Enable ValidateRequest attribute on ASP.NET Pages.
 - iv. Encode untrusted web output prior to rendering.
 - v. Applications available over HTTPS must use secure cookies.
- (b) Ensure that TLS certificate parameters are configured with correct values
 - i. Verify X.509 certificates used to authenticate SSL, TLS, and DTLS connections.
 - ii. Third level item
- (c) Attackers can exploit weaknesses in system to steal user credentials. Downstream and upstream components are often accessed by using credentials stored in configuration stores. Attackers may steal the upstream or downstream component credentials. Attackers may steal credentials if, Credentials are stored and sent in clear text, Weak input validation coupled with dynamic sql queries, Password retrieval mechanism are poor.
 - i. Explicitly disable the autocomplete HTML attribute in sensitive forms and inputs.
 - ii. Validate all redirects within the application are closed or done safely.
 - iii. Enable step up or adaptive authentication.
 - iv. Implement forgot password functionalities securely.
 - v. Ensure that password and account policy are implemented.
 - vi. Implement input validation on all string type parameters accepted by Controller methods.
- (d) If proper authentication is not in place, an adversary can spoof a source process or external entity and gain unauthorized access to the Web Application
 - i. Consider using a standard authentication mechanism to authenticate to Web Application.
 - ii. Ensure that standard authentication techniques are used to secure Web APIs.
 - iii. Consider using a standard authentication mechanism to authenticate to Web Application.
 - iv. Implement forgot password functionalities securely.

- v. Ensure that password and account policy are implemented.
 - vi. Implement input validation on all string type parameters accepted by Controller methods.
- (e) Phishing is attempted to obtain sensitive information such as usernames, passwords, and credit card details (and sometimes, indirectly, money), often for malicious reasons, by masquerading as a Web Server which is a trustworthy entity in electronic communication
 - i. Ensure that authenticated ASP.NET pages incorporate UI Redressing or clickjacking defences.
 - ii. Validate all redirects within the application are closed or done safely.
- 4. Denial of service: Availability
 - (a) Failure to restrict requests originating from third party domains may result in unauthorized actions or access of data
 - i. Ensure that authenticated ASP.NET pages incorporate UI Redressing or clickjacking defences.
 - ii. Mitigate against Cross-Site Request Forgery (CSRF) attacks on ASP.NET web pages.
- 5. Repudiation: Confirmation
 - (a) Proper logging of all security events and user actions builds traceability in a system and denies any possible repudiation issues. In the absence of proper auditing and logging controls, it would become impossible to implement any accountability in a system
 - i. Ensure that auditing and logging is enforced on the application.
 - ii. Ensure that log rotation and separation are in place.
 - iii. Ensure that Audit and Log Files have Restricted Access.
 - iv. Ensure that User Management Events are Logged.
 - (b) Attacker can deny a malicious act on an API leading to repudiation issues
 - i. Ensure that auditing and logging is enforced on Web API.
- 6. Information Disclosure: Confidentiality
 - (a) An adversary can reverse weakly encrypted or hashed content
 - i. Do not expose security details in error messages.
 - ii. Implement Default error handling page.
 - iii. Use only approved symmetric block ciphers and key lengths.

- iv. Use approved asymmetric algorithms, key lengths, and padding.
- (b) An adversary may gain access to sensitive data from log files
 - i. Ensure that the application does not log sensitive user data.
 - ii. Ensure that Audit and Log Files have Restricted Access.
- (c) An adversary may conduct man in the middle attack and downgrade TLS connection to clear text protocol, or forcing browser communication to pass through a proxy server that he controls. This may happen because the application may use mixed content or HTTP Strict Transport Security policy is not ensured.
 - i. Applications available over HTTPS must use secure cookies.
 - ii. Enable HTTP Strict Transport Security (HSTS).
- (d) An adversary may gain access to sensitive data from uncleared browser cache
 - i. Ensure that sensitive content is not cached on the browser.
- (e) An adversary can gain access to sensitive data by sniffing traffic to Web API
 - i. Force all traffic to Web APIs over HTTPS connection.

CHAPTER 3

Design

The design chapter inherits from the conclusions attained in the Analysis in order to establish the architecture of the system. This leads to the description of the overall architecture, and more in depth look into the different tiers (starting from the top) that are responsible for the Presentation, Application and Data.

3.1 Overview

The overarching architecture is composed of 3-layers (known as 3-tier architecture when layers are on different machines) shown in **Figure 3.1**. One big advantage of multi-layer architecture (3-tier for this specific project) is the modularization of different layers, typically the User Interface, Business logic and Data storage. This allows for flexibility by permitting changes to a layer independently from the others. Specifically, the existence of one-way dependencies allows for reusability fx. domain layer can be reused with a different presentation layers. When the layers are present on different computers, they can usually be pictured as Client, App Server and DB Server.

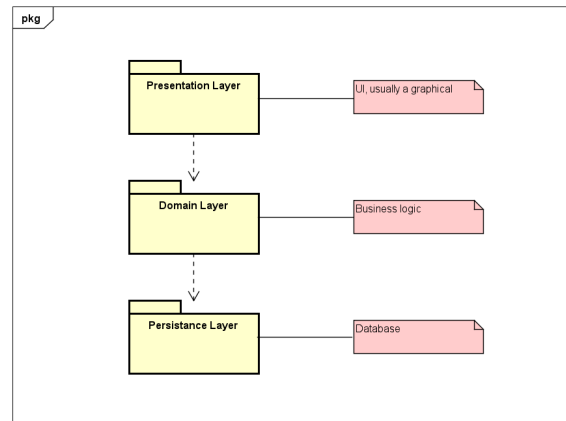


Figure 3.1: 3-Layer Architecture

3.1.1 Presentation Tier

The Presentation Tier represents the front end of a 3-tier system and is considered the top-level tier. It is responsible for displaying content or information requested by the user in graphical form usually through a web browser. The implementation is generally realized through HTML, CSS and JavaScript, however a plethora of modern frameworks can simplify the process by allowing more control and easier deployment. Communication with lower tiers is achieved through API calls. [2]

3.1.2 Application Tier

The Application tier consists of the core logic of the application. Information arriving from the Presentation Tier is handled and often compared with other information present in the data tier bound by specific business rules. When dealing with web applications, this tier processes user inputs and queries the data tier to request, update or delete information. The development is usually done through Java, Python, Ruby etc.; similarly to the presentation tier a multitude of frameworks exists to ease the process. [3]

3.1.3 Data Tier

The Data Tier includes the database in which the information is managed and stored, typically on a relational database management system. PostgreSQL, MySQL and Oracle are commonly implemented systems. As previously stated, in 3-tier systems, the data tier cannot communicate directly with the presentation tier, hence the existence of the application tier.

Deriving from the information discussed, the high level architecture of the system would resemble the

one shown in Figure 3.2. Based on this conclusive diagram, the underlying tiers would be formulated in subsequent chapters, as shown: Tier 1 (or the frontend) is going to be designed and implemented using ASP.NET Blazor ; Tier 2 using the Spring Boot Java Framework; Tier 3 will be separated and use SSL to communicate with the Tier above it; The Database will use PostresSQL relational database.

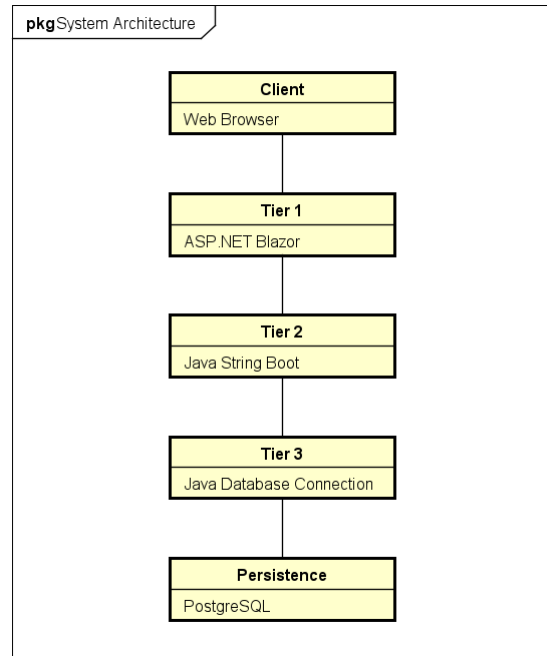


Figure 3.2: System Architecture based on 3-tiers

3.1.4 Discussion on 3-tier Architecture

It is important to discuss, that the architecture above is a demonstration of a final commercial design. Due to Non-Functional Requirements 4 and 5 presented by the client, we have to use RESTful webservice and SSL protocol in the development of the system. A choice was made to use RESTful services between Tier 1-2 and SSL between Tier 2-3. Due to the requirements a node has to be introduced above the Persistence and below Tier-2, which introduces redundant complexity to the system if we scale it horizontally shown in Figure 3.3

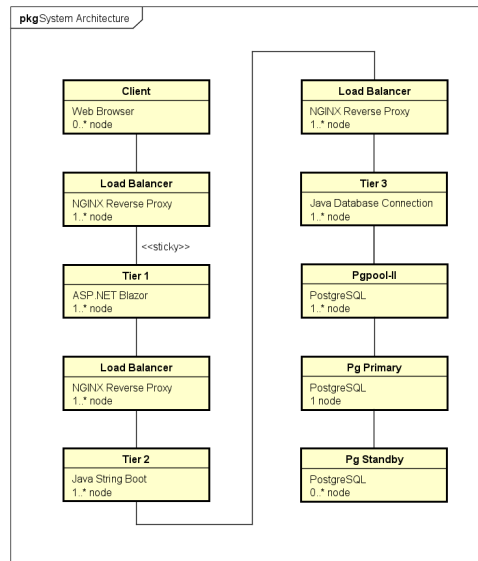


Figure 3.3: System Architecture Discussion 1

The design above is a High Availability distributed system, but the Tier 3 node and the Load Balancer above it adds a lot of redundant complexity, as the Tier 2 could be directly connected to the Persistence. It would remain, however, a 3 Tier architectural solution since the Database Engine would still run on a different machine. The only difference would be, that the queries would be formatted in Tier 2 which also would not violate the 3 Tier Architecture Pattern, as it can be considered part of the business logic (and internal logical separation can still be solved via packages). Also, another advantage would be the usage of the already existing, safe and unified communication protocol between Tier 2 and Persistence. Furthermore Spring Boot has native JDBC support built into the framework, since Spring Boot was designed to be the Middleware of a 3 Tier architecture.

It is debatable, that the complexity can be reduced by discarding the Load Balancer above Tier 3, but to keep the HA design we would need the same amount of Tier 3 and Tier 2 nodes to operate, like shown in Figure 3.4.

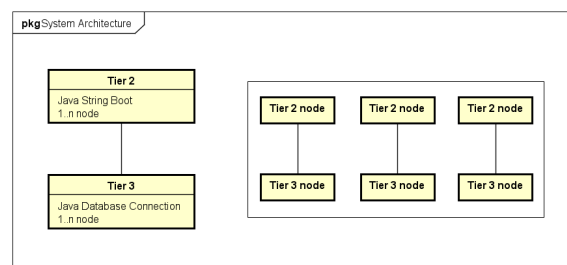


Figure 3.4: System Architecture Discussion 2

This Design clearly have some redundant complexity, as running the individual Tier 2 and 3 node pairs on different machines can be avoided. The HA design shown in Figure 3.5 below should be considered for future implementation under different circumstances as it keeps the 3 Tier architecture, has no single point failures and lacks unnecessary components.

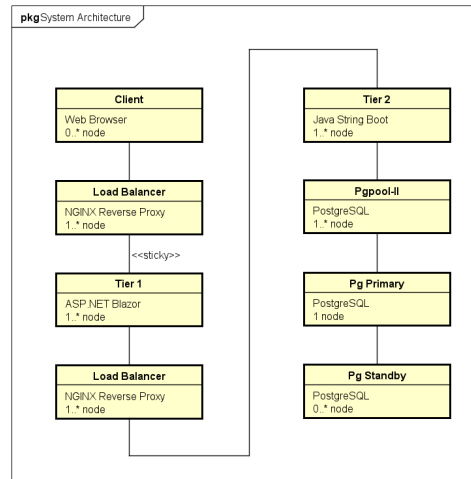


Figure 3.5: System Architecture Discussion 3

3.1.5 Summary

As the idea of this project is to construct a Web Application for Game Rental and Exchange, the 3-tier architecture allows for the usage of a variety of different languages and framework from which ASP.NET Blazor, Spring Boot Framework and PostgreSQL were chosen. In the following sections a structural bottom-up analysis will be performed on each tier describing the classes needed to implement the system.

3.2 Database

As previously mentioned the database is going to use the PostgreSQL relational database managements system. Having previously determined domain objects, allows for the creation of an ER diagram that describes the relationships between the objects in the system. The diagram is shown in Figure 3.6.

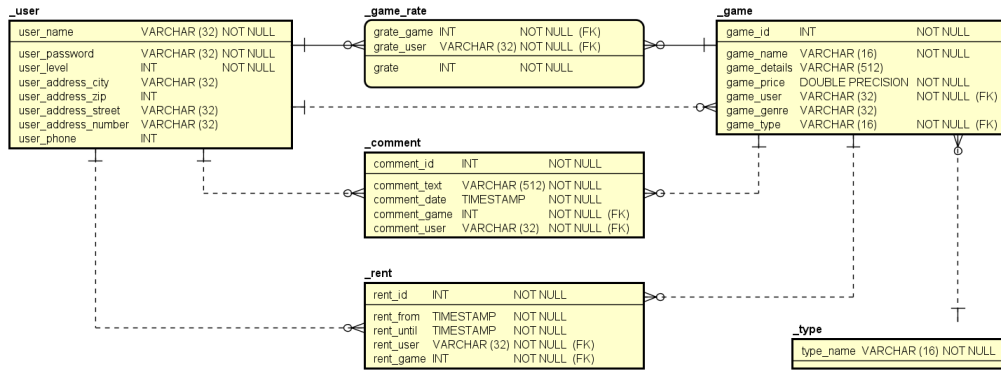


Figure 3.6: ER Diagram

User entity holds the relevant data necessary for a user with a primary key being the **user_name**. The User has a relationship of one to many with every other entity defined in the system. That is a user can have multiple game, provide multiple ratings etc. The other entity of interest is Game itself with a primary key **game_id**. It has one to many relationship with all other entities except User, as a user can own multiple games, and type as a game can be of multiple types (f.x Fantasy, Adventure, Strategy).

3.3 Tier 3

The Tier 3 of the system is directly coupled with the Database. It serves as a direct connection and provides the relevant classes shown in **Figure 3.7**.

As it can be seen, the database server consists of several parts: the data objects, the data access objects, and the mediator responsible for handling requests by the application tier.

The data objects are trivial as they match the ones from the database. The data access objects (DAO) are responsible for mapping the application calls to the persistence layer and are managed by Postgres-Manager, which is an implementation of the Persistence Interface. The isolation supports the single responsibility principle. The Mediator interprets and handles requests from the tier above it.

3.4 Tier 2

The middle tier of the design consists of the Server side of the application. A choice was made to design and implement the second tier in the Springboot Framework allowing for the usage of resources that

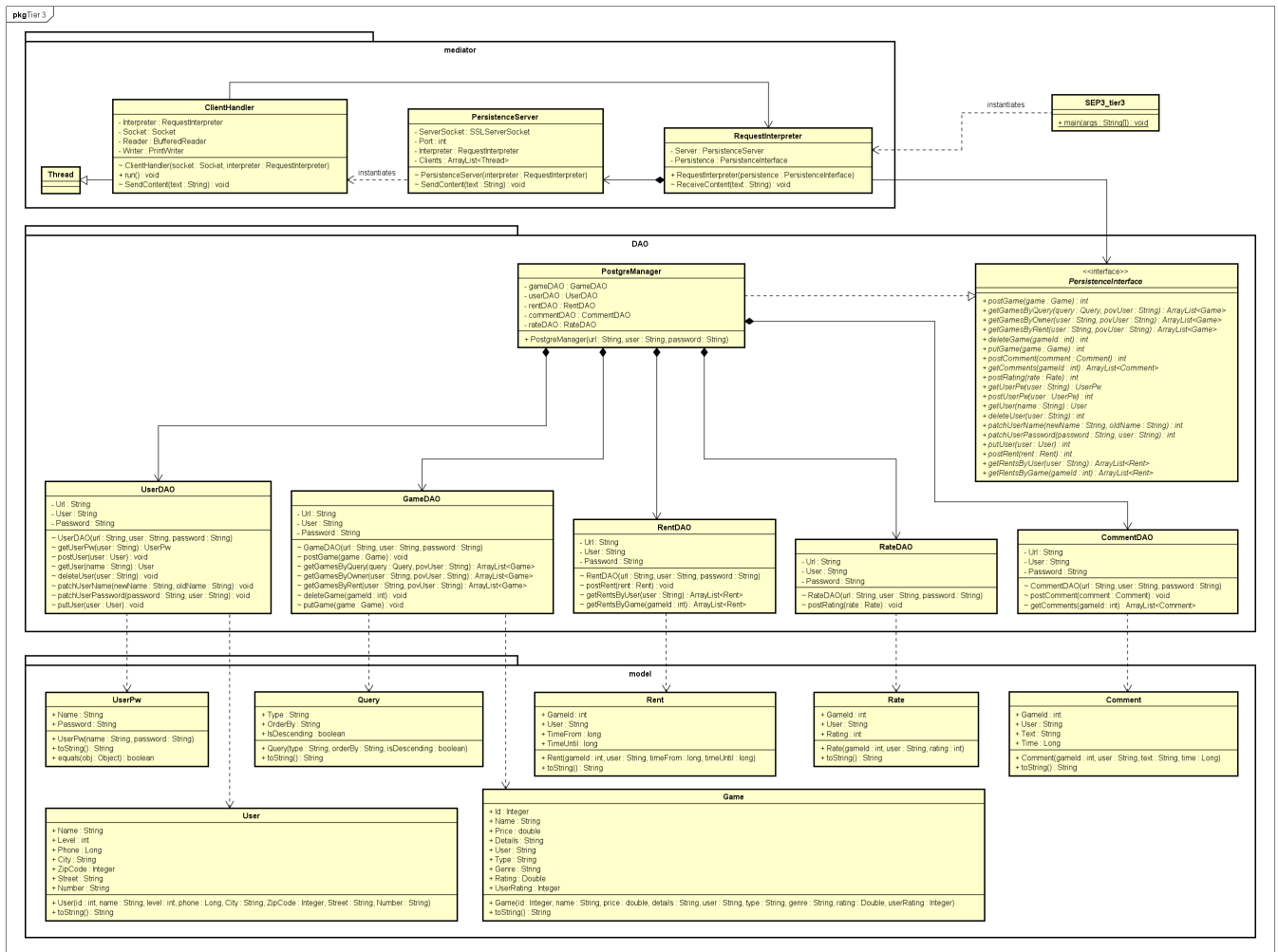


Figure 3.7: Database class diagram

would simplify the implementation process. **Figure 3.8** shows the class diagram of the tier-2 part of the application

As the second tier is a natural continuation of the domain model, the Model of the core logic, seen in the lower part of the diagram, contains the objects previously established in the domain model.

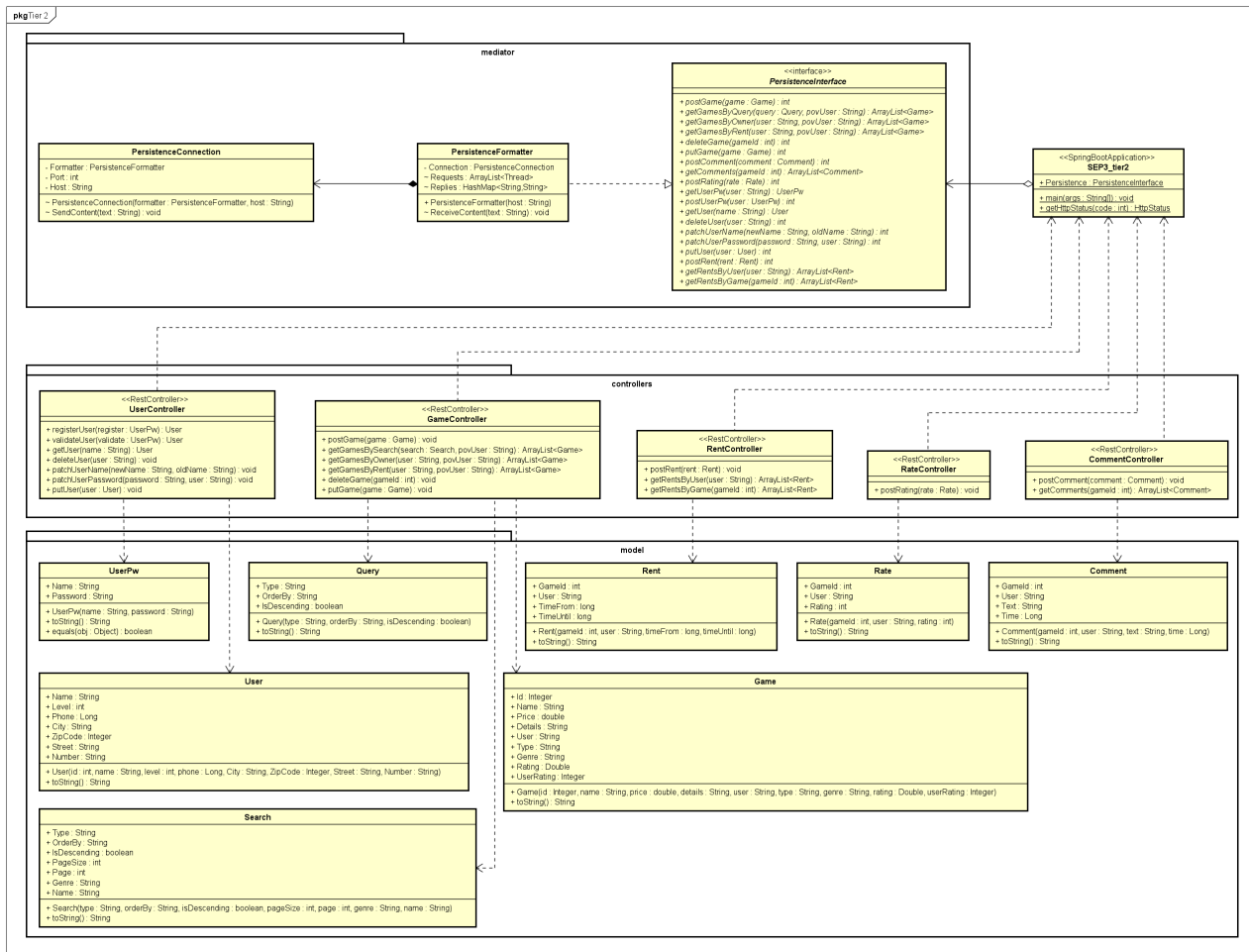


Figure 3.8: Class diagram of Tier-2

The objects present in the model are expanded with the inclusion of their attributes and relevant operations. The main classes Game and User are present as well as Rent, Comment and Game ranting (from the domain model). The representation is achieved by providing objects with relevant fields, constructors, and accessors. What follows are the controller classes shown in the middle section of the diagram, that handle HTTP requests (corresponding to RESTful web services). The last part of the Tier-2, consists of the mediator that would send queries to the database with specified information.

The middle tier of the architecture expands on the domain model objects and is designed to handle request from the top tier and make queries to the data tier below it.

3.5 State Diagram

Establishing the middleware and the backend design, allowed for a reflection on how to proceed with page navigation in order to simplify the design of the frontend. The navigation was established based on the higher priority use cases and supplementing activity diagrams. The state diagram is shown in Figure 3.9.

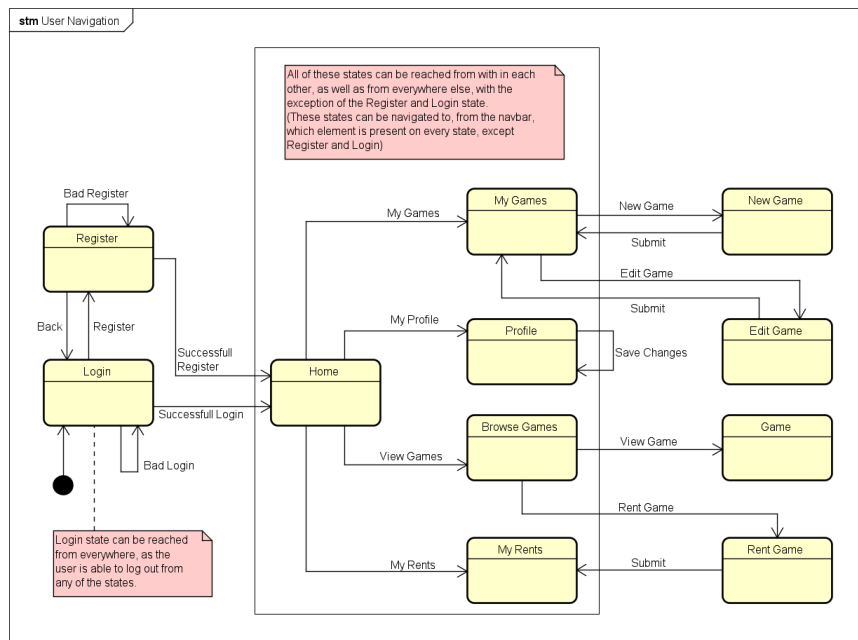


Figure 3.9: Client Class Diagram

3.6 Tier 1

The Tier-1 matches the frontend of the application. As described in the previous sections it is on top of the 3-tier architecture and is designed for ASP.NET Blazor to run the client logic independently. Furthermore having the relevant state diagram, allowed for a faster recognition of relevant classes and razor pages. **Figure 3.10** shows the class diagram. Also shown in the red area of the diagram are the relevant Services.

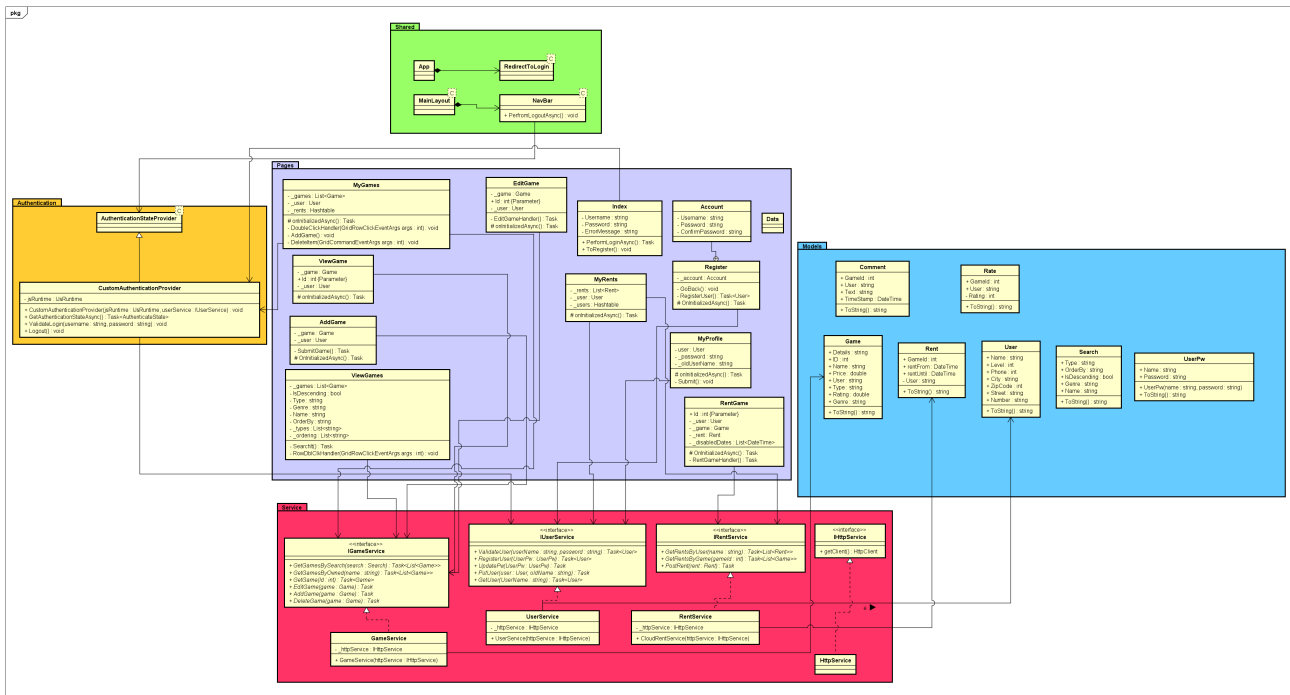


Figure 3.10: Client Class Diagram

3.7 Sequence Diagram

The Sequence Diagram in Figure 3.11 shows an example for password validation in order to demonstrate the object interactions and method ordered calls in the middleware and backend of the application. Although this is a specific example it demonstrates the behaviour of the REST calls. Under the figure is a description of each sequence in order shown in the diagram.

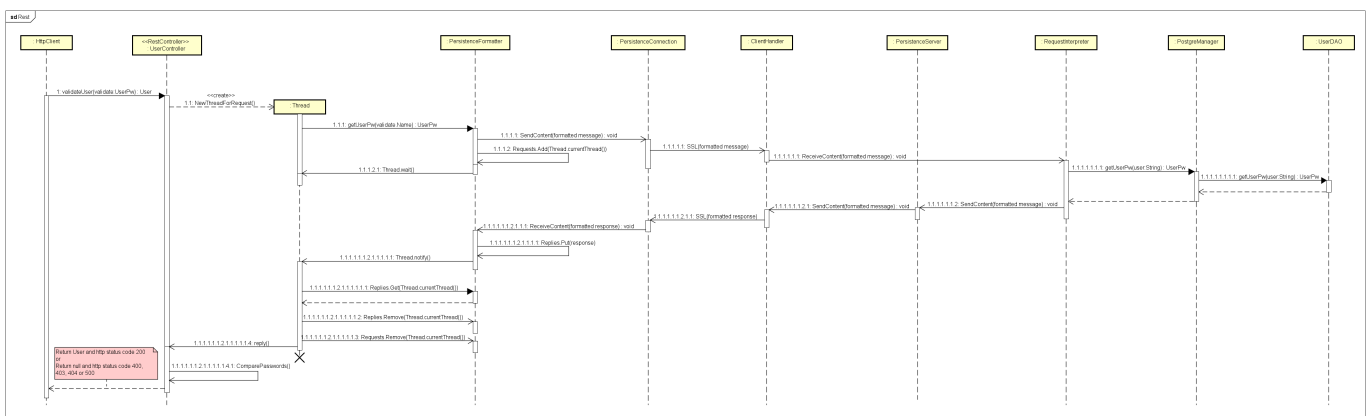


Figure 3.11: Sequence Diagram

- 1 the exposed validateUser RESTful service is called

- 1.1 the Spring framework instantiates a new Thread to handle the request
- 1.1.1 getUserPw is called from the PersistenceFormatter (which is behind a PersistenceInterface)
- 1.1.1.1 the PersistenceFormatter formulates a message which can be sent through an SSL connection, and passes it to the PersistenceConnection
- 1.1.1.2 the current Thread gets added to the Requests list
- 1.1.1.1.1 the message goes through the SSL connection
- 1.1.1.2.1 the current Thread is put on wait
- 1.1.1.1.1.1 the ClientHandler passes the message to the RequestInterpreter
- 1.1.1.1.1.1.1 the RequestInterpreter recognizes the message and calls the appropriate method (getUserPw) from the PostgreManager (which is behind a PersistenceInterface)
- 1.1.1.1.1.1.1.1 the PostgreManager uses the appropriate DAO to serve the request
- 1.1.1.1.1.1.1.2 the RequestInterpreter gets the requested Object or response code, and formulates an appropriate reply message, and passes it to the PersistenceServer
- 1.1.1.1.1.1.1.2.1 the PersistenceServer passes the reply to the same ClientHandler where it came from in the first place
- 1.1.1.1.1.1.1.2.1.1 the ClientHandler sends the reply through the SSL connection
- 1.1.1.1.1.1.1.2.1.1.1 the PersistenceConnection forwards the message to the PersistenceFormatter
- 1.1.1.1.1.1.1.2.1.1.1.1 the reply gets put into the Replies HashTable with the key being the Spring Thread's name that sent the request in the first place (the name is part of the reply message)
- 1.1.1.1.1.1.1.2.1.1.1.1.1 the Thread with same name as which was used as a key in the HashTable, gets notified (the Thread can be found in the Requests list, as it was added there in 1.1.1.2)
- 1.1.1.1.1.1.1.2.1.1.1.1.1.1 the Thread continues its execution and gets the appropriate reply from the Reply HashTable based on its name
- 1.1.1.1.1.1.1.2.1.1.1.1.1.2 the Thread removes itself from the Requests list, as it has been served
- 1.1.1.1.1.1.1.2.1.1.1.1.1.3 the Thread removes itself from the Replies HashTable, as it has been served
- 1.1.1.1.1.1.1.2.1.1.1.1.1.4 the Thread deserializes the reply and returns the requested Object or response code

- 1.1.1.1.1.1.2.1.1.1.4.1 the business logic is done here (in this example the passwords are getting compared, if they match) and an appropriate ResponseEntity serves the initial RESTful request, containing the http status code and an Object in some cases

3.8 Security

The Security Threat Model provided in the Analysis provides a detailed description of the possible threats in the system and some mitigation strategies. Since the scope of this project does not permit for extensive prevention, few possibilities will be mentioned with SSL being implemented in the system.

Firewalls could prevent unauthorized access in areas of higher vulnerabilities. It will filter traffic and protect against unauthorized access.

An antivirus software provides protection against viruses and other undesirable malware and trojan horses.

IPsec can provide authentication and encryption to information sent over IP network traffic. Provides encryption, decryption and key exchange

SSL can provide confidentiality and authenticity to information between the servers.

CHAPTER 4

Implementation

The chapter implementation follows the architecture established in Chapter ??: Design and logically follows the 3-tier architecture model. The frontend of the application is implemented using C# and ASP.NET Blazor with dedicated Web services to exchange data with the server side of the application. The middleware side of the application, implemented using Java Spring Boot, interprets and processes requests coming from the client and via an ssl connection queries the database. By the definition of 3-tier design lower tiers do not have outstanding knowledge about the tiers above them.

4.1 JavaDoc

A comprehensive java documentation was produced for the Tier 2 and Tier 3 where every attribute and method has their respective description. This JavaDoc, supported by the provided source code, assures that lengthy snippet of code can be avoided in the documentation. Nevertheless in the next section, a more detailed description is provided for specific methods.

4.2 Points of interest

4.2.1 Back-end

As previously mentioned the implementation precisely follows the established design. The most complex part regarding the Tier 2 and Tier 3 communication can be described through the Sequence Diagram in Chapter ??.

Nevertheless, a more detailed explanation can be provided for certain parts of the code that were not previously discussed.

The first point of interest is how the messages are formulated in the `PersistenceFormatter` (part of the Tier 2). The message is divided into separate parts via the special character `U+02DC`. The first separation is always the name of the Thread sending the request. It follows the Spring Boot's naming convention and is generally displayed as shown above.

- “http-nio-8080-exec-1” => “http-nio-port-exec-serial”

The part that follows is a special character and it is accompanied by the request's identifier, usually matching the method's name from `PersistenceInterface`, for better readability:

- “putUser” or “postGame”

Then comes another special character and it is followed by the request's attribute(s), this can be a serialized object or just a primitive type, or in some cases multiple primitive types, all separated by the special character.

The reply message is very similar. First part is unchanged and is the sender Thread's name, then comes, either a primitive type (can be a response status code as well), a serialized object or even null.

The following snippet shows us how does the `PersistenceServer` decides which `ClientHandler` has to send the reply:



```

void SendContent(String text)
{
    for (Thread client : Clients)
        if (client.equals(Thread.currentThread())) {
            ((ClientHandler) client).SendContent(text);
            System.out.println( "[SENT] " + text);
            break;
        }
}

```

Figure 4.1: Send Content

As all the `ClientHandler` Threads are stored in the `Clients` list, it finds the one which match the current Thread and sends it there. A more elegant solution would be to just cast the current Thread to a `ClientHandler` and send it there, but this offers more flexibility for future development, as well as better code readability, as it makes it clear what is it doing and why.

For further details please refer to the included `javaDoc`, or source code.

4.2.2 Frontend Implementation

The client side of the system is implemented using Blazor web framework, using C#, HTML, CSS, Telerik and other NuGets. At first, Syncfusion components were used in the system, but due to poor documentation and quirky way of working, Syncfusion was switched out for Telerik, as they provided much better documentation and smoother implementation. Due to the nature of Blazor compiling pages using C# and not JavaScript, it was decided that Bootstrap isn't a proper CSS framework as it uses JavaScript, instead Bulma, a pure css framework without any JavaScript was used. It provided a a clean, easy to make and good-looking design.

DataServices:

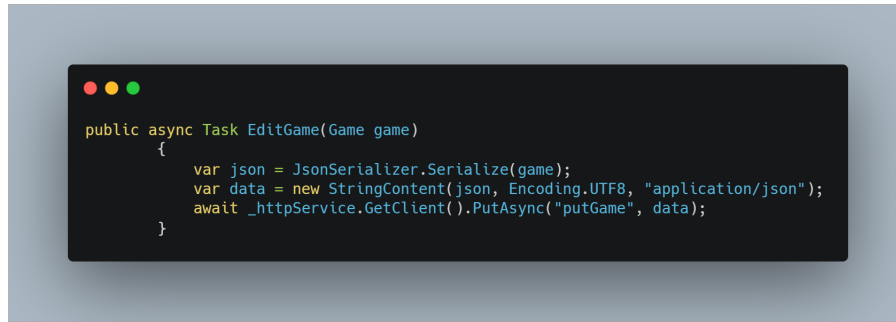
The client side has four different Data Services, which all followed the principles of CRUD – Create, Read, Update, Delete. Though these data services and CRUD operations the client could communicate and access items in the database through the 2nd tier.



```
public async Task<List<Game>> GetGamesOwned(string name)
{
    HttpResponseMessage response = await
_httpService.GetClient().GetAsync($"getGamesByOwner/{name}/{name}");
    if (response.IsSuccessStatusCode)
        return
        (List<Game>)JsonSerializer.Deserialize(response.Content.ReadAsStringAsync().Result,
        typeof(List<Game>));
    return new List<Game>();
}
```

Figure 4.2: Frontend Snippet 1

The crud operations all work in a similar fashion, for example the GetGamesByOwned method uses the httpService to get the client through which HTTPs method Get is used to get a representation of the target resource's state through the request URI that is exposed by tier 2. If the success status code OK 200 is received the response is deserialized and a list full of games that belong to specified owner is returned, otherwise because no games have been found, the deserialization results in an empty list and the status code 404 is received.



```

public async Task EditGame(Game game)
{
    var json = JsonSerializer.Serialize(game);
    var data = new StringContent(json, Encoding.UTF8, "application/json");
    await _httpClient.PutAsync("putGame", data);
}

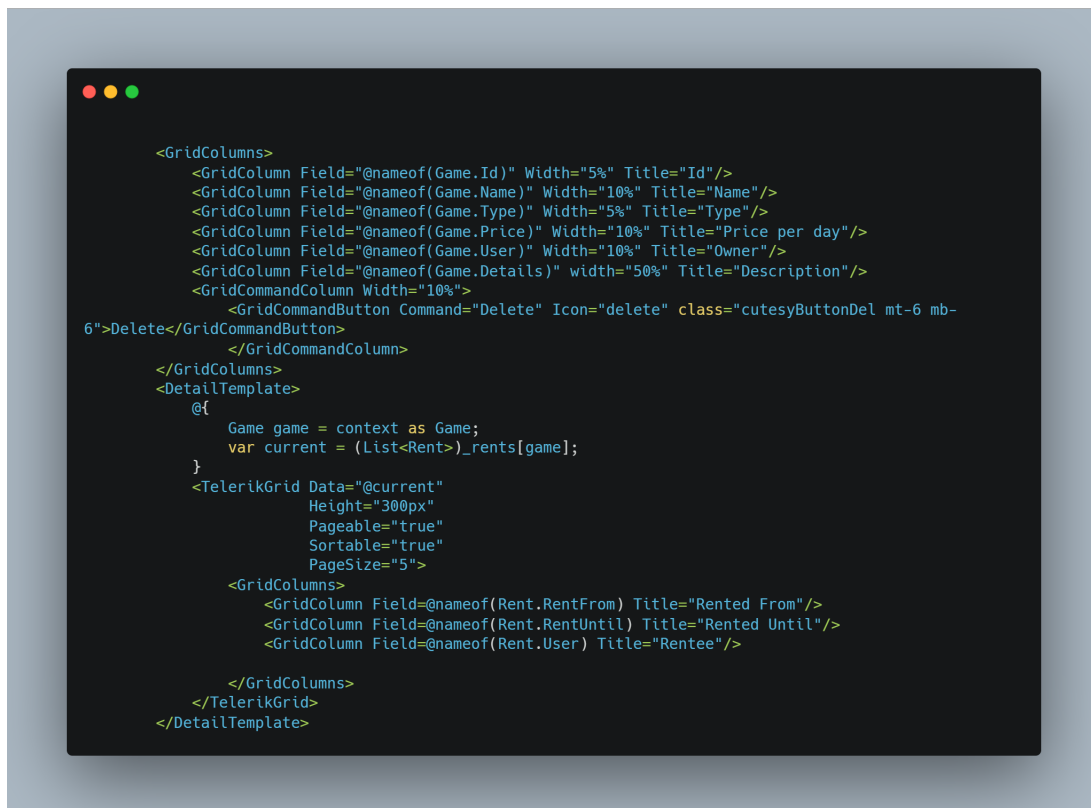
```

Figure 4.3: Frontend Snippet 2

Another example is the EditGame method, first the game that has been changed is serialized and put inside of a StringContent. In this case HTTP's method put is used to set the target resource's state to the state defined by the representation enclosed in the request. The target is specified by the request URI that's been exposed by tier 2.

Pages:

The system contains 11 pages in total. Each page serves its own specific purpose, though three types can be seen: Viewing, CRUD and authentication purpose pages.



```

<GridColumns>
    <GridColumn Field="@nameof(Game.Id)" Width="5%" Title="Id"/>
    <GridColumn Field="@nameof(Game.Name)" Width="10%" Title="Name"/>
    <GridColumn Field="@nameof(Game.Type)" Width="5%" Title="Type"/>
    <GridColumn Field="@nameof(Game.Price)" Width="10%" Title="Price per day"/>
    <GridColumn Field="@nameof(Game.User)" Width="10%" Title="Owner"/>
    <GridColumn Field="@nameof(Game.Details)" Width="50%" Title="Description"/>
    <GridColumn Width="10%">
        <GridCommandButton Command="Delete" Icon="delete" class="cutesyButtonDel mt-6 mb-6">Delete</GridCommandButton>
    </GridColumn>
</GridColumns>
<DetailTemplate>
    @{
        Game game = context as Game;
        var current = (List<Rent>)_rents[game];
    }
    <TelarikGrid Data="@current"
        Height="300px"
        Pageable="true"
        Sortable="true"
        PageSize="5">
        <GridColumns>
            <GridColumn Field="@nameof(Rent.RentFrom)" Title="Rented From"/>
            <GridColumn Field="@nameof(Rent.RentUntil)" Title="Rented Until"/>
            <GridColumn Field="@nameof(Rent.User)" Title="Rentee"/>
        </GridColumns>
    </TelarikGrid>
</DetailTemplate>

```

Figure 4.4: Frontend Snippet 3

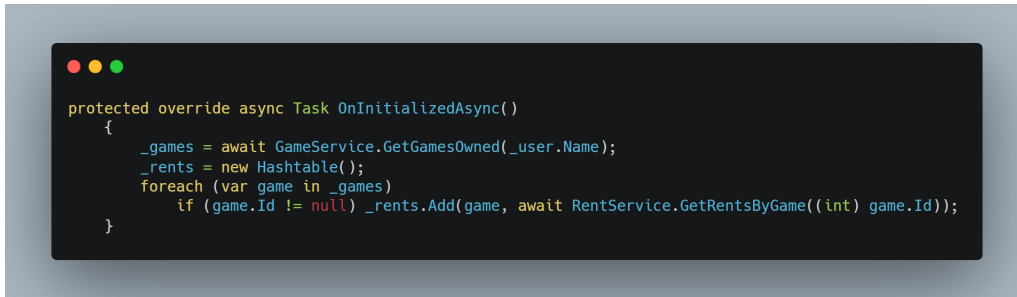


Figure 4.5: Frontend Snippet 4

One of the most interesting pages, MyGames shows the games that belong to the user while also showing each unique rent that belongs to the specific game. A grid inside of a grid was made, where the first grid gets all the games by using the GameService and the second grid getting each unique rent belonging to that game. All viewing type pages act in a similar fashion, they all have a grid which displays object that have been fetched by the specific data service.

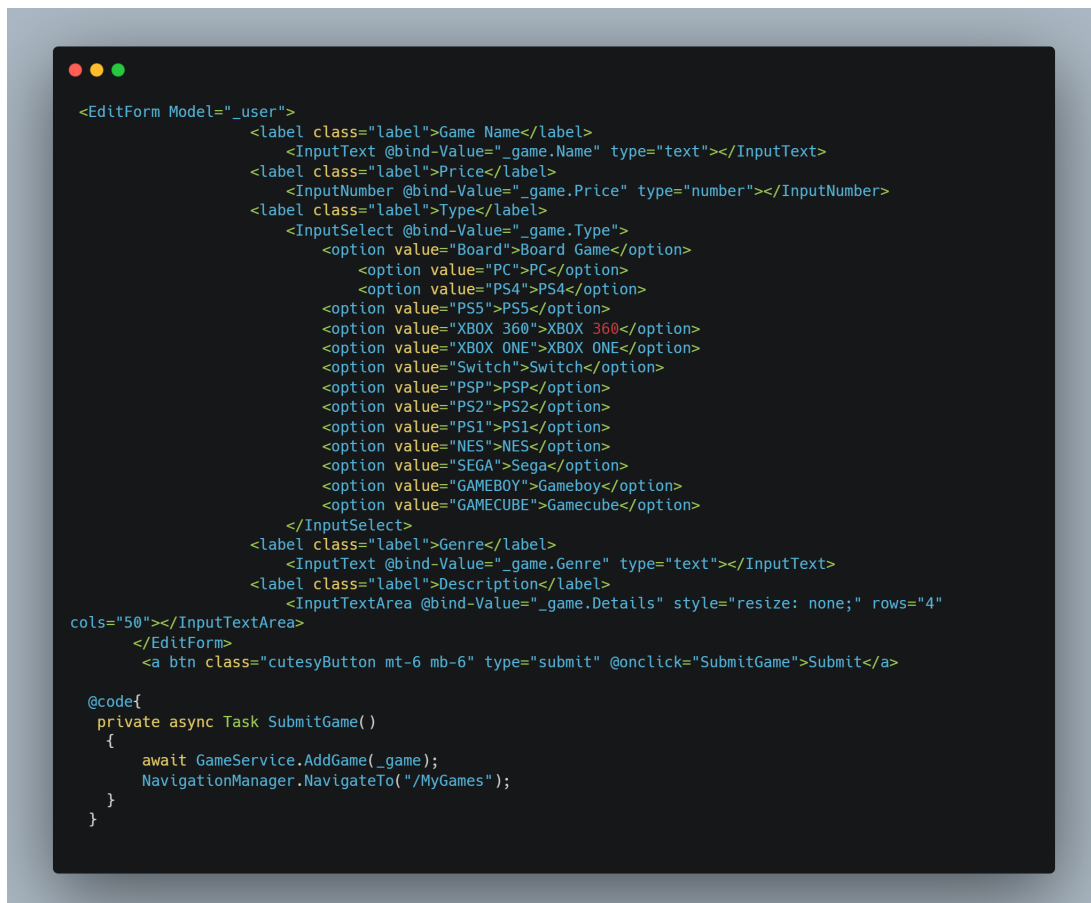


Figure 4.6: Frontend Snippet 5

These pages are relatively simple, their main purpose so to do one of the CRUD operations. For example,

the AddGame page serves as a Creation operation for the Game object. The coding part is fairly simple, an EditForm component that's been provided by Blazor has been used to act as the information holder, after the submit button is pressed a Game object is created by using the information provided in the fields and after the POST method is called to add the newly made game to the system.

Authentication:

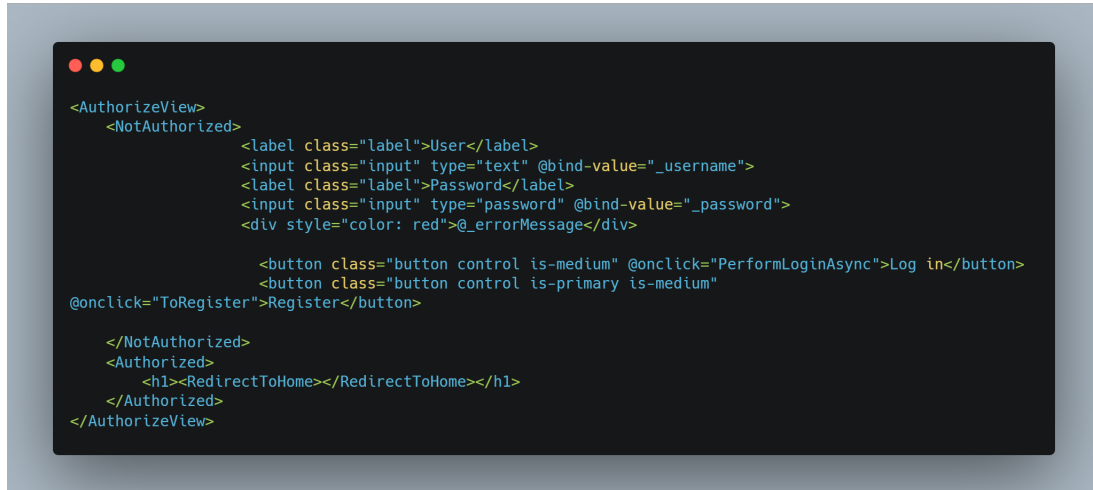


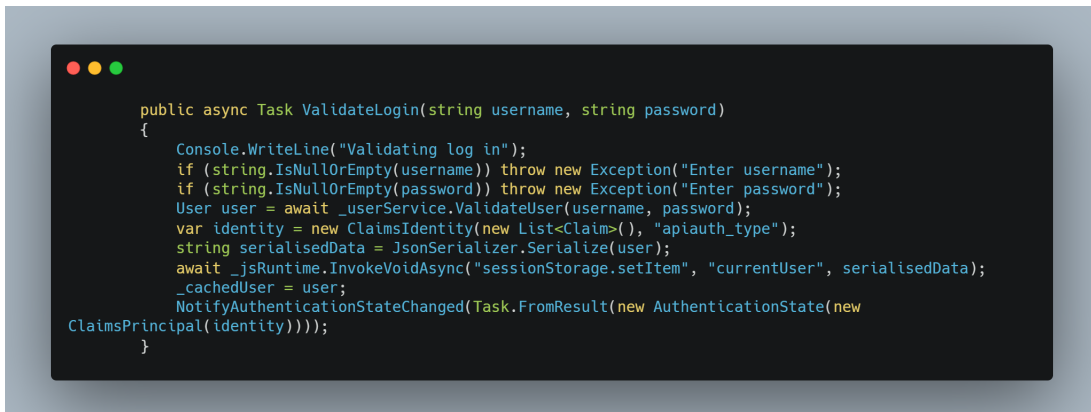
Figure 4.7: Frontend Snippet 6

The last type is the authentication type, which has the purpose of authenticating the user, it has the input fields for username and password and two buttons, which using the navigationamanger take you to the register makes if clicked on register and one that validates your credentials and takes you to the home page. There are two views, one for unauthorized users which displays the login page and an authorizedview which has a RedirectToHome component that redirects you to the home page in case you are already logged in. A similar component is also added into the mainlayout called redirecttologin that redirects all unauthorized users to the log in page.

AuthenticationStateProvider

One of the most important parts in tier 1 is the AuthenticationStateProvider, it has the purpose of handling authentication, validating log ins, assigning claims, caching users and logging them out.

The validation process begins by checking if the entered fields are empty, either one are an Exception is thrown, otherwise the UserService is called to validate the username and password combination, if such combination exists a user is returned which is then cached, the authentication state provider is notified and the user is successfully logged in.



```
public async Task ValidateLogin(string username, string password)
{
    Console.WriteLine("Validating log in");
    if (string.IsNullOrEmpty(username)) throw new Exception("Enter username");
    if (string.IsNullOrEmpty(password)) throw new Exception("Enter password");
    User user = await _userService.ValidateUser(username, password);
    var identity = new ClaimsIdentity(new List<Claim>(), "apiauth_type");
    string serialisedData = JsonSerializer.Serialize(user);
    await _jsRuntime.InvokeVoidAsync("sessionStorage.setItem", "currentUser", serialisedData);
    _cachedUser = user;
    NotifyAuthenticationStateChanged(Task.FromResult(new AuthenticationState(new
ClaimsPrincipal(identity))));
}
```

Figure 4.8: Frontend Snippet 7

CHAPTER 5

Testing

The testing chapter provides a closer description on the tests performed to verify the correctness of the system. Testing can be specified as functional and non-functional. The functional testing will test the application against the set requirements in order to verify the parts of the software behave as expected.

5.1 Functional Testing

In this paper the focus for testing will rely mainly on Unit testing, Integration testing and System(Blackbox) testing.

5.1.1 Unit testing

The first unit testing performed is done suing NUnit test in order to verify the Web services used in the Blazor client or Presentation Tier of the system. Results are positive and can be seen in [Figure 5.1](#).

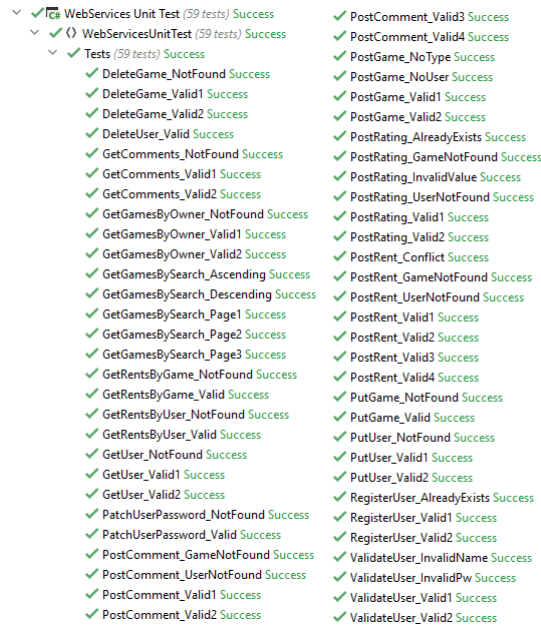


Figure 5.1: Nunit test for Web Services

Java Unit tests or dependency based testing.

5.1.2 System testing

The system testing is the blackbox testing necessary to evaluate the integrated system in general. It ensures that specific requirements, derived from the use cases and following specific scenarios, are met to ensure the integrity of the application. Verification is obtained when the expected outcome matches the achieved one. The tables provided bellow offer the test cases concerning some of the more important use cases. The use case test for the Register Use Case needs a simple navigation to the specified Register Area. What follows is a field where a user can input desired credential. The registration is finalized when the button to register now is selected and the user is navigated back to his Homepage. The use case test Login assumes the user has an existing account and interact with several tiers of the system, by validating the user credentials and comparing them to information in the database. It is vital to the system, that the Username is unique, although it may be edited later. The full PDF of Blackbox testing can be seen in Appendix D.

Use Case	Data	Steps	Expected Outcome	Actual Outcome	Test Passed Yes/No
Register	Name: "Blackbox" Password: "123456" Retype	1 Press Register button 2 Add information 3 Retype Password	User registered	User registered	Yes
Login	Name: "Blackbox" Password: "123456"	1 Input username and password 2 Press Login	User login	User login	Yes

Figure 5.2: Register and Login Use Cases

5.2 Non-functional testing

Minimal non-functional testing is performed on the system, in order to satisfy the bare minimum in performance, security, usability and compatibility.

CHAPTER 6

Results and Discussion

The result of the project is a fully-functioning three-tier distributed system, which managed to implement all core features while utilizing a continuous workflow methodology. The requirements were satisfied and the final result is a system, in which a user can manage a collection of games, as well as rent out games from different users with the intent of exchanging them personally, which is the extent of the platform. The User Manual can be found under Appendix E. The overall look of the system is appealing to the expectations, it bears good implementation of the requirements and the results are a good-looking frontend of an otherwise completed system. The testing made on the system proved to be successful and with little to no problems, which concludes the results of the project to be satisfactory.

CHAPTER 7

Conclusion

The rise in demand for board, video and out-of-production games has risen in the past years, thus creating the need for a solution for people to easily exchange games in a local area. The current existing solution are subscription-based sites, which furthermore showed the need for a system, which can easily connect people with similar interests. The chosen methodology for this project is Kanban.

Based on these problems, a set of requirements is made, which are then categorized based on priority - from Critical to Low. From all 17, only 7 weren't implemented and all of them are in the medium or low priority list.

After the requirements were made, the analysis phase beared the creation of a Use Case Diagram and Descriptions derived from each Use Case. Based on the Use Cases and their descriptions, Activity Diagrams and a Domain Model were created to further analyze the development of the system. After which the design was started.

The design of the system was created based on the requirements and use cases derived. The Architectural composition is to be three-tiered with security protocols implementation and database to further expand the reach of the system. The class diagrams of each tier represent the connection of the classes with each other. A state diagram is created, which shows the navigation in the frontend of the system. The Sequence Diagram demonstrates the connection of the whole system through a single method.

The implementation was done using Java, PostgreSQL, C# and libraries, such as Telerik for the Blazor Client Side, as well as Web Services for the exchange of data between tiers. The middleware was implemented using Spring Boot to manage and interpret requests from client, and using SSL Queries for the database connection.

There were two types of tests done on the system - Unit testing and Blackbox Testing. Unit testing was firstly done using NUnit to verify the web services' usage in the Blazor client. The Blackbox testing was

conducted to proofcheck the usability of the final system by using Use Cases and scenarios, in which the tester is seen as the user. It is then, that if the actual and expected outcomes are the same, that the test has passed.

The project could be worked on in the future. There are requirements, which can be added with ease, such as the Leave a comment requirement, which is already being handled by Tiers 2 and 3, but only needs frontend implementation. In conclusion, the goal of the system has been achieved, which was to implement a system, where users can add and trade their desired games.

CHAPTER 8

Future Development

The system is fully functioning, despite not having implemented some medium to low priority requirements. Due to time management, some low priority requirements had to be left out, even though Tiers 2 and 3 supported their handling, such as the leave a rating and comment requirements. In the case of future collaboration between the members of the team were to happen, Tier 1 would need the implementation of said requirements, as well as introducing a home page to the system with News, Statistics and other similar features. Another element, that could be included, is the Instant Messaging function, which would require implementation in all tiers, as this was not in the plans for this project.

Overall, no further changes need to be made, because the system achieved its goal, which was for users to be able to have a collection of games, as well as being able to rent out other users' games.

CHAPTER 9

References

- [1] STRIDE chart - Microsoft Security (2007). Available at: <https://www.microsoft.com/security/blog/2007/09/11/stride-chart/> (Accessed: 18 December 2020). [2] 2 (2020). Available at: <http://www.cs.sjsu.edu/faculty/pearce/modules/p2/> (Accessed: 18 December 2020). [3] What is Three-Tier Architecture (2020). Available at: <https://www.ibm.com/cloud/three-tier-architecture> (Accessed: 18 December 2020).

Process Report

Game Rental System

by



Levente Nagy(293115)

Eva Nikolaeva(293164)

Audrius Sauciunas(293156)

Philip Philev(293735)

Supervisors:

Troels Mortensen

Joseph Okika

Software Technology Engineering

3rd Semester

Table of Contents

- 1 Introduction
- 2 Group Description
- 3 Project Initiation
- 4 Project Description
- 5 Project Execution
- 6 Personal Reflections
- 7 Supervision
- 8 Conclusion

1. Introduction

The purpose of this process report is to show a detailed overview of how the group formed and worked together to develop the 3rd semester project. The focus area of this document is to show how the group formed, collaborated and finalized a system, while solving issues related to the personal reflections from the members.

The goal of this project is to develop a distributed system, that adheres to the three-tiered architecture pattern, while using Java, C# and a database system. The group had a choice for the topic of the project, as well as the distribution of its components, as long as it followed the VIA Guidelines and Protocols.

The courses, which were utilized in this semester project were as follows: Software Development of Distributed Systems 3(SDJ3), Internet Technologies, C# and .NET (DNP1) and Network and Security(NES).

The group agreed to meet every Wednesday and Sunday through online platforms such as Discord. The meetings were held online, because of the travelling expenses some of the members had, thus resulting in the united decision.

The group was formed at the beginning of 3rd semester through a past 2nd semester group inviting Eva to join. The decisions were made based on past experiences, motivation and dedication related to each person's studies.

2. Group Description

Group creation

The formation of the group was based on the past experience of all members, which is why Levente, Philip and Audrius wanted to work together once more for their 3rd semester project. Eva was invited to the group based on her similar motivations and good teamwork. This formation contributed to the equal distribution of work on the project and the guidelines, which were followed were shown in the group contract made the first week.

The group contract assured the flow of the work was constant and the conflicts were brought down to a minimum, as well as making sure each person was in a healthy environment, where they could express their opinion with no judgement. This contract was essential to the work done, because of the remote work that was to take place. The document can be found at the end of the chapters.

The group consists of 4 members from 3 different countries – Bulgaria, Hungary and Lithuania. Due to the diverse culture, the group was filled with different opinions essential to receiving perspectives from all angles, productivity and different approaches to problems.

E-stimate Profiles

The estimate profiles were made during our 1st semester in VIA under the supervision of Mona Wendel Andersen, who was our Study Skills for Engineers (SSE) teacher. These profiles consist of the personal and professional description of each person and utilizes a simple questionnaire, which gives you a detailed description in the form of a chart. The chart uses four colors, which represent a group of descriptions and depending on how big your color is, the more you fit the description of said color. The profiles can be seen below. (Figurec 1,2,3,4) All Figures can be seen as PNG in Appendix A.



Figure 1

Audrius

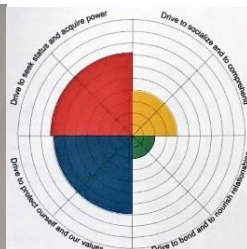


Figure 2

Eva

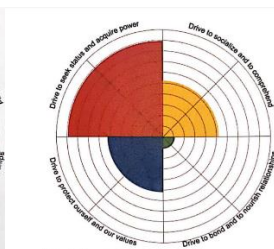


Figure 3

Levente

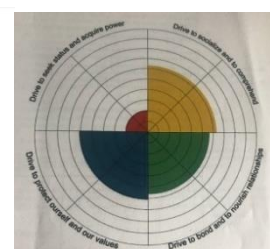


Figure 4

Philip

Audrius:

My e-stimate (Figure 1), reveals my profile to be predominantly blue, while still having a high amount of red and green, and slight amount of yellow. I can recognize myself in this profile as I find myself to be quite introverted, structured and detail-oriented in my line of work, though if needed I can initiate action and be the leader if work stalls, while I do possess red traits, I am not

a dominant person, I care about each teammates well-being and prioritize their well-being before kicking off serious work.

During group work, the color that materialized the most was blue, I created a structured work environment for myself, where I got to utilize newly gained knowledge on the group project, which appealed the pragmatic side of me wonderfully. While all of this might seem as a positive endeavor, it comes with a cost. Being blue also means being a perfectionist, striving to achieve the greatest possible outcome in every way possible, which means I sacrificed every single day out of 21 days of work we had, including weekends. In result, I am proud of what I have achieved, though I must admit it did come at a cost of stress and loss of free time.

My project group has said, that my e-stimate profile characterizes me properly. I mostly strived to have a structured, purposeful work, though considering there was poor communication at a start I tried taking the role of a leader trying to put everything back in place and if it worked only the result will tell.

Eva:

My diagram (Figure 2) shows, that my dominant colors are red and blue, with red being slightly bigger. I believe this visualization represents me very well, as red describes a person being confident, goal-oriented and the initiator of driven actions, whereas blue describes introverted, detail-oriented and structured people.

In a group, I tend to show more of my “blue” traits, as I like to organize and schedule every step of the way, in order to ensure quality work and time management to its full extent. On a bad or stressful day I tend to show the “red” traits, which include arrogance, impatience and dominance, but I’ve always had a team backing me up, which is why I make sure this does not happen often.

My project groups so far have all said, that the e-stimate profile describes me well and have noticed on regular occasion, that I tend to get along with almost all color representatives. E-stimate has helped me understand, that no matter your colors and “bad days”, you must always show respect and understanding to the people you work with, because they are the ones, who will support you and your work no matter what.

As much as I believe I am a representative of the colors red and blue, I still think that every person bears all colors and all traits associated with it, because people cannot be judged based on a single personal profile, no matter how accurate it may be.

Levente:

According to my diagram, I am predominantly red, with equal yellow and blue. Based on this and the past project works, this proven to be right in general, as I usually tend take the lead in some parts of the project. Also because of my red profile I have no problem and fear of rational, moderated conflict between me and other team members.

In the current team environment, my behavior proves to be beneficial as I make the team moving forward. But a big part of this result is probably due to the fact that I am the most dominant one, and even though I have no experience in working with team members whom has the same or

higher red oriented mindset and behavior as me, depending on the exact situation I believe I would still be an important asset on the team.

Philip:

Personality tests are something that can give an overview of a person's tendencies in a work environment. Of course, it is not concrete but can make an estimate of a person's strong sides.

My estimate profile is mostly yellow, blue, slightly less green and almost minimal red. The yellow traits describe a more outgoing and innovative attitude with a creative role in the team. The presence of blue tends to signify a more detail-oriented and introverted approach, which tends to balance with the yellow traits. Minimal red and somewhat high green tends to present a non-dominant and willing to compromise side.

Although I tend to agree with the estimates, I believe that the lack of red is not entirely correct since I exhibit certain traits from that spectrum. Furthermore, the profile gives an overview of a person's characteristics, but does not go into specific details of a person's character that almost always is a combination of different factors that shape a person's personality.

Hofstede Cultures

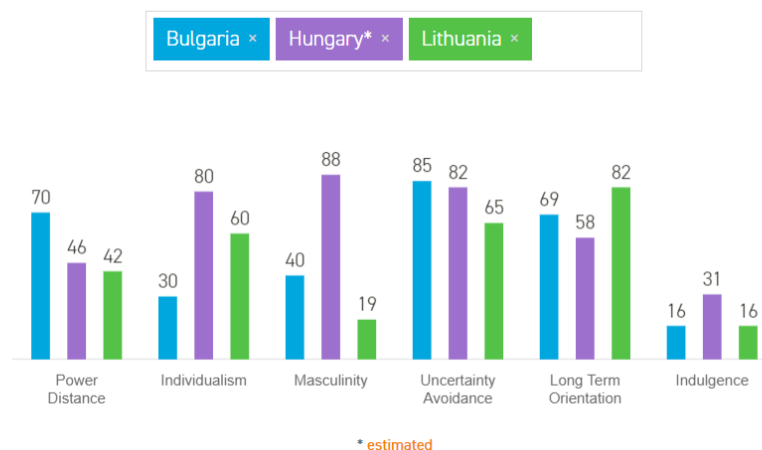


Figure 5

The Hofstede Cultural Dimensions is a diagram-based cultural description, that uses six categories to describe the people from the chosen country. For example, Power Distance is the first category, which relates to how much the culture of said country is considered “unequal” and respect being highly or lowly valued in the professional industry and/or daily lives.

Based on these descriptions a person can reflect on how much they consider to have inherited these values just by living in a country that owns a set of the Hofstede descriptions.

We decided to include a graph, which consists of the three countries our group represents and reflect on how high or low it also describes us on a personal level.

Figure 5 can be seen as PNG in Appendix A.

Audrius(Lithuania):

Lithuania has the lowest power distance out of the three countries, which reflects me as a person quite strongly in the group, I believe that every person in the group should be equal and given equal opportunity to learn and achieve just as much as anyone else. Personally, in Lithuania we do not have a very strong sense of titles, though we do frequently use formal language, which in return means that we do not believe in people being higher than you, but we do believe in having formalities to keep people being higher than you, but we do believe in having formalities to keep the harmony between each other.

Personally, I do not reflect my country in the dimension of Individualism, while I do not disagree with the high number, I do disagree that it reflects me as a person. I believe I do believe that taking care of yourself first is important, but that cannot be in the expense of other people's well-being. In my eyes, if you are able to help yourself, you should be able to help people around you as well.

Eva(Bulgaria):

My country is shown as having high power distance and uncertainty avoidance, as well as having low individualism and masculinity. I believe this is absolutely correct regarding my views and experiences based in Bulgaria. For example, teachers in school are always called "Mr" or "Mrs", which indicates a high power distance. People rarely take business-associated risks, such as changing professions, which is a clear representative of a high uncertainty avoidance.

I believe Bulgaria has inherited these descriptions from accumulated history, as well as a lot of influence from neighboring and/or related countries, which brings me to believe, that it is only a matter of time, until the culture and social views change.

Bulgaria's social descriptions and traits have taught me a lot, but my moving to Denmark has also shown me a different perspective, culture and people, which is why I never inherited my country's views.

From a professional standpoint, I believe I bear no similarities to my country – I believe in people, who bring equal contribution to work should not be separated by authority. Risk-takers should also be encouraged, rather than regulated by strict rules.

In conclusion, I have seen the polar opposites of almost all descriptions and while Bulgaria has taught me a lot, I bear no similarities to the view of my countrymen.

Levente(Hungary):

The most obvious behavior which both myself and team members as can observe on me is the high individualism trait. This is mostly reflected on situations where I find it more efficient and beneficial to do certain tasks alone, rather than splitting it up or getting help from fellow team members. I am still working on containing this behavior of mine as it is proven to be non-beneficial, neither for myself nor team members.

The other obvious trait is the high masculinity which can be always observed on the project's design as I tend to push it towards the most efficient technologies and techniques I know, so that

we can achieve the most in a given timeframe, rather than “having fun” with other less known or inefficient ways, probably resulting in less advanced end product.

As for all the other traits, I do not think they can be particularly observed on my behavior. This probably because there is not a huge difference in between the level of those attributes and the corresponding attributes of other team members’ countries’, making these behaviors not outstanding in the current team environment.

Philip(Bulgaria):

Although I tend to agree with some of the estimates presented in the country chart, I believe no single population can be summarized as an average of a thought process. It is true for family values, but in work and collaborative environments, a person’s personality tends to play the more important role.

I do not think being from Bulgaria, alters my ability to collaborate better or worse with people from other countries. The way education was structured in Bulgaria was obviously different from how it is in Denmark. The former tends to focus more on individualistic achievements, while the latter tends to value a more team oriented.

As I mentioned, the descriptions identifying Bulgaria, tend to form a more collectivistic and unequal in a work environment picture, however in my opinion the collectivism mostly applies to family values and the power dynamic is mostly concentrated in government institutions, remnants of the previous socialistic tendencies of politics. This is changing rapidly as the younger generations tend to follow a more outgoing and informal approach in work environments matching Denmark.

In conclusion, collaborating and team-oriented work has never been a personal issue, although, looking at the estimates may paint a different picture. This summarizes my argument that country estimations especially on the modern European continent are largely based on data present from previous generations. Young people in modern European Union tend to share similar way of thinking and behavior. The country profiles may be more relevant in the current discussion if group members were from different continents or political entities.

Group contract as PDF can be seen in Appendix A.

Group Contract

Date: 16.09.2020

These are the terms and conditions we agree on as a team in the group contract:

Participation:

1. Members should attend all classes they can and inform group members of late arrival and/or absence.
2. Members should actively participate in all scheduled SEP meetings.
3. Members should catch up and do their assigned work in the case of missed meetings.

Communication:

1. Members should discuss all project-related ideas openly without judgement.
2. Maintain a friendly and motivated environment, where all members can feel valued and heard.
3. Members should be able to check and respond to messages regularly from other group members regarding the Project.

Meetings:

1. Members should meet a minimum of one time a week to work on the project and/or give feedback and document progress.
2. The workload should be divided equally among members and finished in the time period the members agreed on.
3. All members must attend all scheduled supervisor meetings.
4. Extra work sessions should be scheduled and agreed upon 1 day before to ensure proper communication and understanding of each other's schedules.

Deadlines:

1. All project tasks and assignments should be done 1 day prior to the deadline, as to have time for review and documentation.
2. The Project must be done 3-7 days before the hand-in, in order to have time for review and correction of small mistakes made.

Conflicts:

1. Members should collectively discuss group conflicts and reach a consensus.
2. If members cannot resolve the conflict, a supervisor meeting should be held.

Other issues:

1. Changes can be made in the group contract with the approval of all group members.
2. All members are responsible for all their learning, as well making sure it doesn't interfere with the Project.

Members:

Levente Nagy (HU)

Audrius Šaučiūnas(LT)

Philip Philev(BG)

Eva Nikolaeva(BG)

3. Project Initiation

The group was tasked with coming up with an idea for the 3rd Semester Project while following the guidelines of being a three-tier distributed system. At the start, we had no ideas and the brainstorming proved to be not as useful, so Eva suggested we should recreate her 2nd semester project, which was a Board Game Rental System, only in this project the scope would be all video, board, retro and out-of-production games. The whole group agreed to the proposal, as long as it differentiated itself from the past works of one of the members.

Due to COVID-19 regulations and group members not living in the same city, it was agreed upon, that remote working will be done with the help of online tools. For group work on Wednesdays and Sundays, as well as the Project Weeks we decided to use Facebook Messenger and Discord for fast communication. Git and GitKraken were used for Version Control, while Microsoft Teams was used for Diagram and Documentation storage and simultaneous editing. VIA itslearning and studienet sites were used for guidelines and templates, which ensured the group was on the correct path to realizing the project. The tool for Project Planning, which we decided upon was Jira Software by Atlassian, as it provided us with the

The methodology used for the project was decided to be Kanban, as it followed a path, which needed no roles, no constrict time schedule and tasks could be added, deleted and edited without the need to wait for the next Sprint. This approach was experimental, due to the previously used SCRUM methodology, which proved to be not as effective to all the members, as it was not the most optimal approach regarding the current world situation and the coordination of the members' time schedules. Detailed description of Kanban and how we used it is provided in the Project Execution chapter.

4. Project Description

The Project Description was the first task of the team and a “test run” of the Kanban methodology. The group distributed chapters to each member and in the span of a week all work was done, revised and sent for approval. The Project Description was the easiest part of our project, but in no means was it not useful. The group was referring to the Risk Assessment and Time Schedule chapters each meeting to ensure proper workflow.

Our main difficulty was assembling our tasks in an organized and time efficient way, so that our schedules do not infringe on the work being done regularly and progressively. The new environment we had for this project was also a slight hinder in our workflow at first, but we quickly managed to overcome it. In this process Kanban showed to be flexible and worked impressively well in a remote environment.

The comprisal of the Project Description was surprisingly easy, as we were all educated enough on the Board and Video Games topic to be able to envision the system with its components and its key functions without even starting the analysis phase.

5. Project Execution

In first semester we were introduced to Methodologies, which are methods for structuring work in the Software Engineering field. The first one we learned was the Waterfall method, which we were required to use and it proved to us, that it was not as effective as we had hoped. In second semester we were introduced to SCRUM, which is an Agile methodology consisting of Sprints, Roles, Charts and Backlogs. We experienced working with SCRUM and saw both its positives and its negatives. In third semester there were no requirements for Methodology, which meant the group chooses it. In our group we decided on Kanban, which would be a first to work with for all members.

Kanban is an agile methodology used by many software development teams all around the world, because of its basic easy-to-understand principles and the vast positives associated with it, such as flexible planning and scope, clear status on the WIP (Work in Progress) and the omnipresence of the Kanban Board. The Kanban Board is a board, which is a visualization of the work In Progress, Done and To Be Done. Instead of starting a new Sprint when the time is up, the Kanban board just adds new “Kanban Cards” to the board. The roles, which are seen in SCRUM are not present in the Kanban methodology, as the team is self-organized and much more flexible to the open cards, which need to be done.

The figure below is a screenshot of the Kanban Board during one of the project stages. It can be seen as PNG in Appendix A.

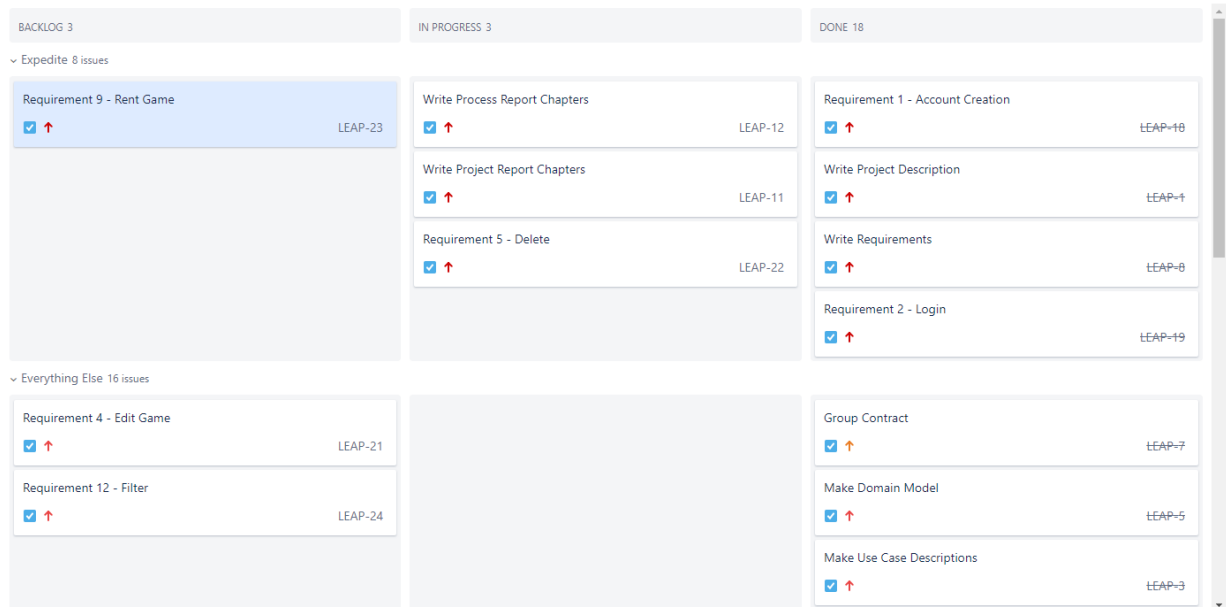


Figure 6

As it can be seen, the board describes the tasks To Be Done (Backlog), In Progress and Done, which are ranked from Low to Highest with Highest having its own Expedite

window, where only the most important tasks, such as documentation and the most basic of requirements, which are essential to the proper functions of the system, are added.

5.1 Log Book

23.09.20 – Wednesday

First day of project work. We have a task to write the Project Description. We decided on using Kanban as a methodology and we set up a Jira Software board to track our progress. Each group member received one to two chapters to write in the following 2 days, after which we will all review the document and edit anything needed.

07.10.20 – Wednesday

We started working on the Architecture, Requirements and Domain Model. The Requirements were done during the meeting with the effort of all members. Domain Model and Architecture are still in question, as we need to work out the details and structure of each Tier.

21.10.20 – Wednesday

We finished the domain model and started working on Architecture and Read-Write Examples. The Tier 2 Class Diagram came in question while a group member was explaining the details of the Architecture and we started preparing the backlog for the next few meetings.

25.10.20 – Sunday

We started working on the Use Case and Sequence Diagram. Read-Write Examples and Architecture are finished and ready for review in the next supervisor meeting.

28.10.20 – Wednesday

We changed the architecture based on the meeting earlier in the morning. We finished the Use Case and Sequence Diagram and started working on Use Case Description and Activity Diagram, as well as starting all the Tier Diagrams. First mock design of the GUI is done.

04.11.20 – Wednesday

We finished the Use Case Descriptions and Activity Diagram. Still some polishing left on the ER Diagram, as well as the Tier 1 and 2 Diagrams.

11.11.20 – Wednesday

We started assembling the Tier Skeletons (making the empty classes, pages and interfaces), as well as the Project and Process Report.

25.11.20 – Wednesday

We started implementing the Tier 2 and Tier 1 Models, as well as populating parts of the database. Project and Process Report have Introductions. Work done on Reports is mostly on the Process – Group Description and Project Initiation chapters are done. Security Assignment is finished.

30.11-14.12 – Project Weeks

The first project week we implemented the first few requirements, as well as doing some more writing on the Project Report. Analysis chapters are done, as well as Project Description chapter. Project Execution chapter started.

Second week and most of the requirements are fully operational, as well as more Project Report writing. Some team members have problems with running the database scripts in pgAdmin. JUnit and Blackbox testing started.

Third week and all requirements are finished. The reports are done and need only conclusions, abstract and appendix filled in. The diagrams are updated wherever needed and only thing left is the User Guide.

6. Personal Reflections

Audrius:

This semester was the second covid-19 marked semester. I must admit that VIA did a tremendous job in making sure that this semester is just like any other, there's a huge contrast between second and third semester as the knowledge I managed to gain highly outweighs second semester. After gaining lots of knowledge from teachers throughout the teaching period we finally started working on our semester project. Though the journey began way before, our capability of working with each other was shown after project period began.

Although we had a rough start the very first week as there was a lot of poor communication between each other, we managed to pull ourselves together and create a fully functional system. Though I had to sacrifice a lot of my free time as I've been working every day of the week. For the first time, I managed to be fully responsible for a big part of the project, I managed to work on it independently, something never fully managed to achieve in previous semesters.

We got our project idea from our group member Eva, who proposed to create a game rental system. In the end I am happy that we chose this idea as it was not only an interesting concept, but I also enjoyed coding every part of it. For the first time it also felt as if we were working on a real world project as we applied many concepts used in real companies. We made a three-tier architecture, we planned together the communication between the tiers and how it was supposed to be coded. It wasn't an easy task, but our workflow wasn't made overly complicated since we chose to work with the Kanban methodology which did not overly pressure us.

All in all, I am pleased to have had a chance in participating in the creation of our rental game system project. This semester I gained a tremendous amount of knowledge in many areas. I am happy that I also got to use that knowledge on an actual project. I believe that this semester helped me grow as a software engineer and as a person in general as I managed to achieve great independence and take responsibility for different areas.

Eva:

I was very optimistic when we first started the project. I was excited to work with people who were not only my friends, but also who had the same vision about this project as me. I was very glad when they agreed on my idea to re-do my last semester project, because I believed, that the three-tier distributed system would work very well with the idea of a Game Rental System. I was very pleasantly surprised when they shared my opinion on not wanting to use SCRUM as methodology for this project, because this way we could try something new, which presented itself in Kanban. I adored working with Kanban and if I could, I will definitely do it again.

I am extremely happy with how the system turned out. We all did our best and we worked every day, talking on discord, listening to music and just genuinely having fun with the experience. Most of all, I really like how the system looks.

Overall, I had the most pleasant, stress-free experience and I am very happy with both the result and the company.

Levente:

Considering the fact that the COVID-19 pandemic was still going on, I honestly think we managed to adopt much better than last semester, and our final result is pretty close to what we would have achieved if we don't have the limiting factors from the pandemic. Also this was the first semester when we did not have hitchhikers on the team and it felt good that everybody took their part and contributed more or less equal amounts.

At the E-stimate profiles part of this document, I stated that I have no experience working with people having the same amount or bigger red traits as me, which still stands, although this was the first time I worked with another predominant red person, Eva. And to my pleasant surprise, we did not have any problems working together. In conclusion, this was a pleasant semester all together.

Phillip:

The third semester, just like the second was during the coronavirus pandemic. This lead to a similar remote work experience. Nevertheless, due to the fact that we have worked in a similar manner the previous semester, we were more prepared on how to collaborate remotely.

There were no interpersonal problems in the group compared to previous semester where tensions were rising due to our inexperience with remote work.

The project itself was varied in work flow. We had productive weeks and some which were not especially good due to health problems or other issues.

The final system was close to what we envisioned, and we managed to test our obtained knowledge from subjects in practice.

In conclusion, I believe we can perform better in real life than what we achieved during this semester, however, it was a better experience than the previous semester.

7. Supervision

28.10.20 – Architecture Feedback Meeting – Supervisors present: Troels Mortensen, Joseph Okika

We had a supervisor meeting regarding the architecture we submitted. We were informed, that our architecture was missing a Tier 3 Module and needed to be changed accordingly. Our Read and Write examples also needed to be changed to show the communication between the client and tier 1 only, whereas the diagrams showed the communication between all layers and the client.

Thanks to this meeting, we understood better how our database is going to communicate with Tier 2 and it was an insight on how to better develop the later stages of the project.

10.12.20 – Supervisor Meeting – Supervisors present: Troels Mortensen

We were informed we needed to have a supervisor meeting regarding the state of our project. We had a chat about how our project is going and what we needed to work more on, as well as understanding that one or two people working on one part of the project, while the others worked on something different was not a good way of managing a project, so we quickly changed our methods.

Thanks to this meeting we realized, that our approach had to be changed and we quickly stepped back on our feet. In conclusion, better late than never.

8. Conclusion

We all agree, that we are satisfied with how our project went this semester. The team was cohesive, motivated and helpful every step of the way and while some of our project management methods were not correct, we still persevered and found the right way. Kanban was an advantageous tool in our project and we managed to get used to working with a new methodology, that showed us a different approach.

Since we all had experienced remote working beforehand, this time we were prepared for the hardships involved and managed to use the time given to us efficiently and the whole team maintained a continuous workflow despite mixing up work and home.

The biggest hardship in our project was understanding the mistakes we made, which were a result of mishandled and inconsistent supervisor meetings. We all agree, that the lessons we learned along the way would improve our teamwork, management and approach for the future semesters to come.

In conclusion, this experience was highly beneficial not only from a teamwork standpoint, but also from a professional one.