**Software-Architecture Documentation – Tom Brauns**

Documentation for the Architecture of an MMO-Game

Tom Brauns

Matrikelnummer: 676672

E-Mail: [Inf3621@hs-worms.de](mailto:Inf3621@hs-worms.de)

Professor: Prof. Dr. Volker Schwarzer

Semester: Summer Semester 2022

Due Date: 26. June 2022

Table of Contents

[Glossary: 3](#_Toc106374437)

[1. Introduction and Goals: 4](#_Toc106374438)

[1.1: Design Purpose 4](#_Toc106374439)

[1.2: Requirements Overview 4](#_Toc106374440)

[1.3: Quality Goals 6](#_Toc106374441)

[1.4: Stakeholders 6](#_Toc106374442)

[2: Constraints 7](#_Toc106374443)

[3: 4+1 View – Context and Use-Cases to help understand 8](#_Toc106374444)

[3.1: Logical / Structural view 8](#_Toc106374445)

[3.2: Implementation / Developer view 10](#_Toc106374446)

[3.3: Process / Behavior view 12](#_Toc106374447)

[3.4: Deployment / Physical view 14](#_Toc106374448)

[3.5: Use Cases 15](#_Toc106374449)

[3.5.1: Use Cases of the Client 15](#_Toc106374450)

[8. Crosscutting Concepts 17](#_Toc106374451)

[8.1: Structural Model 17](#_Toc106374452)

[8.2: Architectural and Design Patterns 18](#_Toc106374453)

[8.2.1: Performance 18](#_Toc106374454)

[8.2.2: Availability 21](#_Toc106374455)

[8.2.3: Security 23](#_Toc106374456)

[10. Quality 24](#_Toc106374457)

[10.1: Quality Tree 24](#_Toc106374458)

[10.2: Quality Scenario 26](#_Toc106374459)

[Alternative Sources: 27](#_Toc106374460)

[Advantages and Disadvantages for CSA: 29](#_Toc106374461)

[Sources: 30](#_Toc106374462)

# Glossary:

User: Person that plays the game.

Client: The general platform that the entire game runs on (Software)

Server: Running component that the User needs to connect to

Pain Point: Potential issue.

Immersion: Potential to dive into the world / relate to the world

Downtime: Time, in which the product is unavailable

Stakeholder: Stakeholder usually refers to people being involved with the Product, be it Primary Stakeholders (Users), Secondary Stakeholders (e.g., Workers for the Project / Salespeople), or even Tertiary Stakeholders (People talking about it)

Non-playable Character (NPC): A video game character that is not controlled by a player of the game. (Oxford Language Definition)

Risk- / Reward-System: Implementing features usually comes at a cost. You can never be 100% certain that whatever you implement will work out. And spending resources / money on features that are not necessary might not be the best way to approach.

Client Server Architecture (CSA): Architecture that involves servers and clients accessing the first

Peer to Peer(P2P): Architecture, in which users are acting as both client and server

Reboot: Restarting of a server.

User Interface (UI): The means by which the user and a computer system interact, in particular the use of input devices and software. (Oxford Language Definition)

Denial of Service (DoS): An interruption in an authorized user’s access to a computer network, typically one caused with malicious intent. (Oxford Language Definition)

# 1. Introduction and Goals:

The focus of this game will be to enable vast amounts of players to connect and interact with each other’s.

Latency issues should be minimized as the interactions of players with one another and with objects in the world are the central focus of every massively multiplayer online role-playing game.

After therefore launching the game and creating an account, the user is then able to create a character in the menu or choose an already existing character, should this not be your first login.

Afterwards you load into the world and are free to interact with other players and the world.

(Diagram)

## 1.1: Design Purpose

This is meant to be a playable prototype that may be distributed under a few many so that they can then test the functions (an early alpha version) and give feedback.

The purpose of this prototype is to enable users to experience an open-world adventure, in which they can experience and explore the world by themselves or with others. The created world should be open, but also persistent, meaning that the actions the individual takes should have consequences to the greater scheme of the world.

## 1.2: Requirements Overview

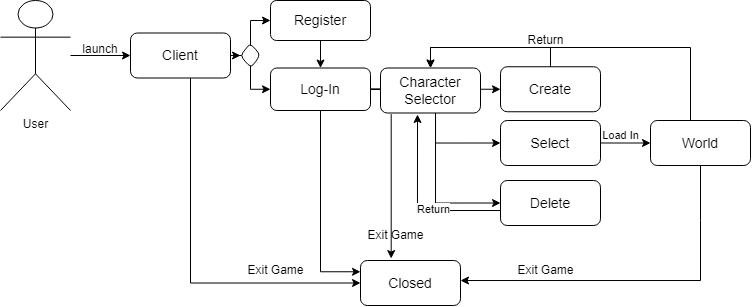
#### Functions include:

* the creation of an account,
* logging in to the client,
* creating a character or selecting an already existing one,
* interacting with other users in the game (mostly chat based),
* interacting with objects / Non-Playable Characters (NPCs) in the game (multiple choices to be selected)

#### A summarization of the functional requirements would look as follows:

|  |  |  |  |
| --- | --- | --- | --- |
| ID | Requirement | Description | Reasoning |
| F-1 | Register | User registers on a web application | Using a website to handle registration. |
| F-2 | Log-in | User logs in with his account name and password | Users have an account that they then sign in on. |
| F-3 | Create character | The user creates a character. | In the character creation the User can customize his own character to his own liking. |
| F-4 | Select character | The user selects a character | Users may have multiple characters so a selection process should be possible |
| F-5 | Delete character | User deletes an existing character | If the user is unhappy with certain aspects of an existing character (i.e., the characters looks, the name etc.), he’s able to delete it |
| F-6 | Load in | The user chooses a character and loads into the game | The loading process is a way for the Game to prepare certain areas that the user is either accessing at the moment or about to access. |
| F-7 | Playing | The User plays the game | A majority of the gametime consists of the User playing the game. |
| F-8 | Return to character selection | The User loads back into the character selection | Instead of having to open the game and close it every time he wants to log out / change characters, another path should be possible |
| F-9 | Exit game | The User closes the Game | The User should always be able to close the game. |

#### The following Graphic shows the different Functions available to the User:



## 1.3: Quality Goals

#### The Main Quality Attributes for this Project are as follows:

|  |  |  |
| --- | --- | --- |
| Quality Attribute | Priority | Motivation |
| *Performance* | 1 | The user and its interactions with the world are the main focus of this prototype. Performance is key when we want the user to experience the highest amount of immersion. |
| *Availability* | 2 | Users have little to no respect for downtime. The higher the availability, the higher the acceptability of the product. It’s not something that the user specifically asks for but will complain about if it adds up. |
| *Security* | 3 | Seeing as this has the potential to involve multiple users, the attribute “Security” shouldn’t be left out completely. Seeing as the world involves resources (those being Items and materials), those items should be secured and accounted correctly. Also, the aspect of harmful intentions of Interactions should be prepared for, such as a malicious attempt to access resources out of the users reach. |

## 1.4: Stakeholders

#### [Stakeholders](#_Abbreviations:) for such a Project could be as follows:

|  |  |  |
| --- | --- | --- |
| Stakeholder | Type of Stakeholder | Reasoning |
| User | Primary | The user should be kept in mind when decisions are made. |
| Sponsor | Primary | Gains to benefit from the success of the project whilst also providing some of the budget available for this project |
| Streamer | Primary | While they are considered users, they might also be contracted to affiliate with the product and generate revenue for the company by providing Publicity |
| Media | Secondary | Spreading the word about the project will go a long way when it comes to getting people invested / interested in the product |
| Developer | Secondary | The developer should be involved. The better his understanding of the shared vision the greater the results. |

# 2: Constraints

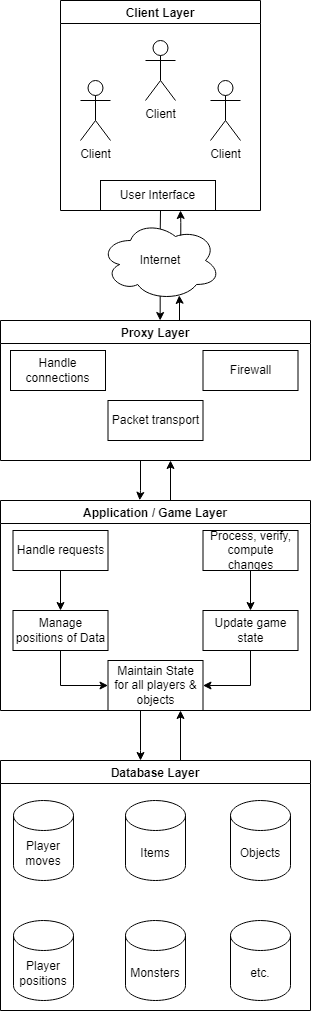
#### As with all large-scale projects, this prototype is also constrained by certain aspects, as shown in the Table below:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **ID** | **Constraint** | **Type** | **Description** | **Motivation** |
| CT-1 | Time | Business | The first testable prototype is expected to be finished in 1 year, marking the first of June 2023 | First results will lead to faster feedback and therefore a better final product. |
| CT-2 | Funds | Business | The funds available for the realization of this project are 300.000€ | Funds are always limited, and the risk / reward system is always to be kept in mind. |
| CT-3 | Capacities | Technical | Only two servers are available for the Systems realization | Hosting a massive project will lead to certain tensions and pain points. |
| CT-4 | Testing | Technical | Test of functions and system components are required to be thoroughly tested. A Test coverage of at least 90% is expected | The first impression is key when it comes to huge projects like this. Trying to enable the best potential product to users goes a long way therefore. |
| CT-5 | Programming Language | Technical | This product must be coded using and being optimized for java. | Most current systems support Java which makes it a great option to reach a broad audience. |
| CT-6 | Coding guidelines for Java | Conventional | Since the used programming language is Java and due to it being published, the conventions that come along with its usage are required. | Using certain guidelines helps new programmers / other programmers to easily get a grip on what the potential problems might be. |
| CT-7 | Interval of updates | Business | A new patch must be released once per month containing new content. | Frequent updates give the game a sense of freshness. A game that hasn’t received updates in a while might be perceived as finished. This is something we do not want, as we want to continuously supply changes and new content to the paying user. |

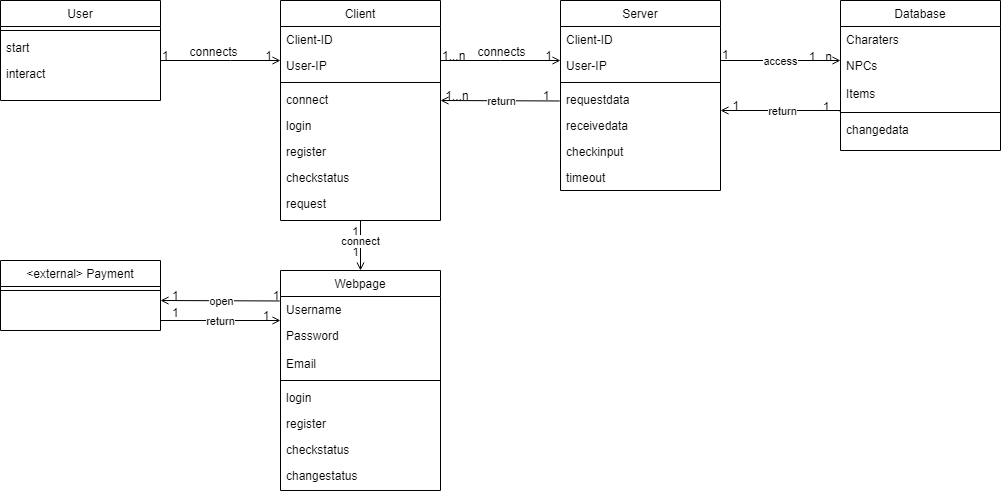
# 3: 4+1 View – Context and Use-Cases to help understand

## 3.1: Logical / Structural view

In this section we will want to focus on the requirements analysis and the functionality. For us to optimally do this, let‘s look at the general [Software Architecture](#_8.1:_Structural_Model) we are working with:

**

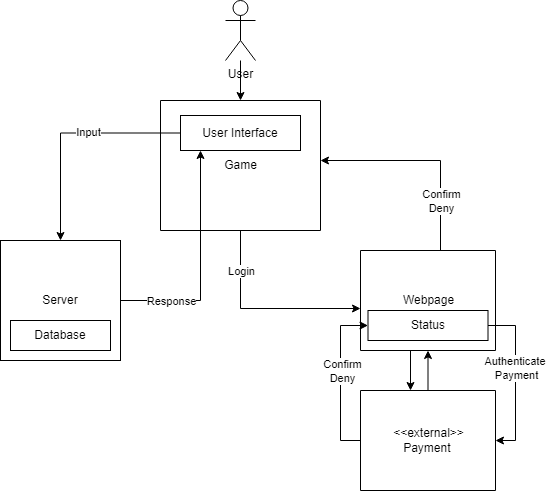
Users connect to the server through a locally installed Client.  
How exactly the 4 layers are interacting with one another is portrayed in the class diagram below:



## 3.2: Implementation / Developer view

This section will contain general implementations of the system components whilst still taking in consideration the functional aspects.

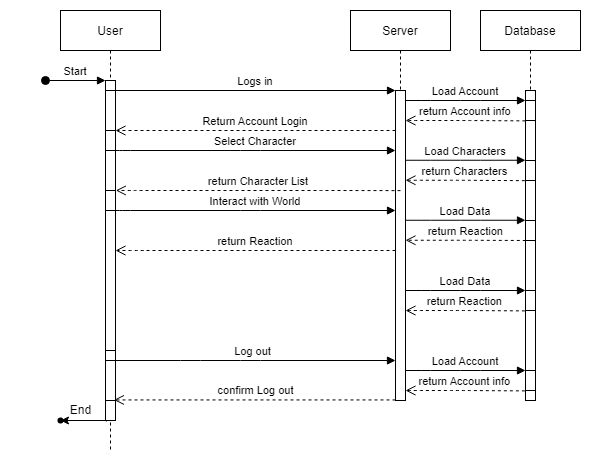
#### The technical context would look as follows:



#### Artifacts relevant for the technical context are as follows:

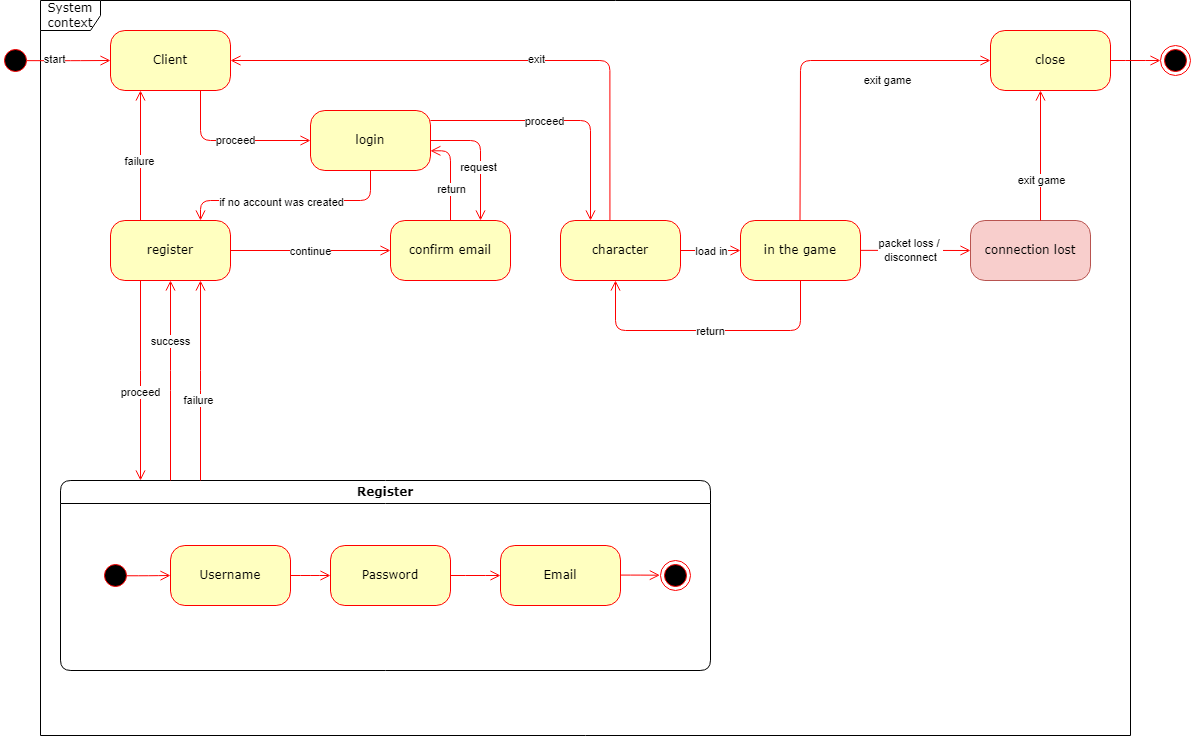
|  |  |
| --- | --- |
| Artifact | Description |
| User Interface | User interface considers the general client and the game experienced by user. Its inputs are being transmitted to the server which then sends his response. |
| Status | Status refers to the subscription status of the user.  This status can either be “active” or “inactive”.  If the user confirmed his payment, the status is changed, and the User can log in. |
| Server | The server is the general processing unit of all the users’ inputs. It must have the capabilities to handle inputs of a great amount of users. |
| Database | The database contains all the information regarding items, [Non-Playable Characters (NPCs)](#_Abbreviations_and_Annotations:), objects, and all of the metadata concerning the players |

#### The following Sequence diagram should depict a possible usage of the game:

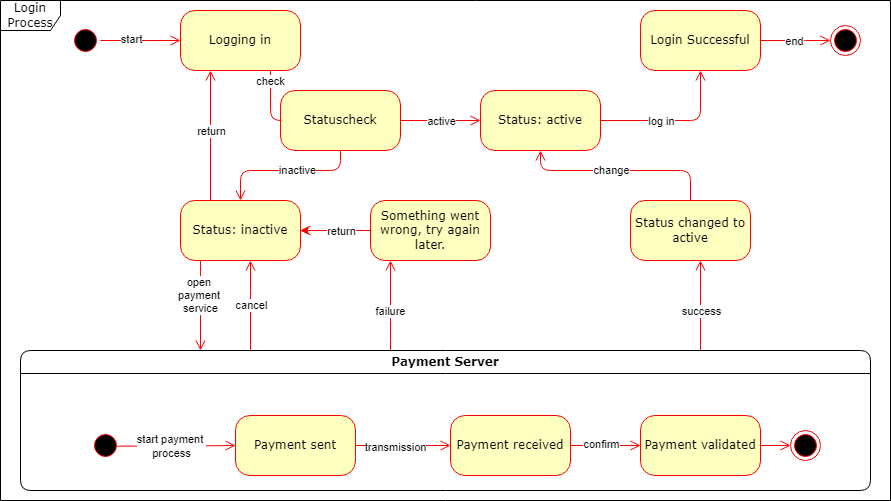


## 3.3: Process / Behavior view

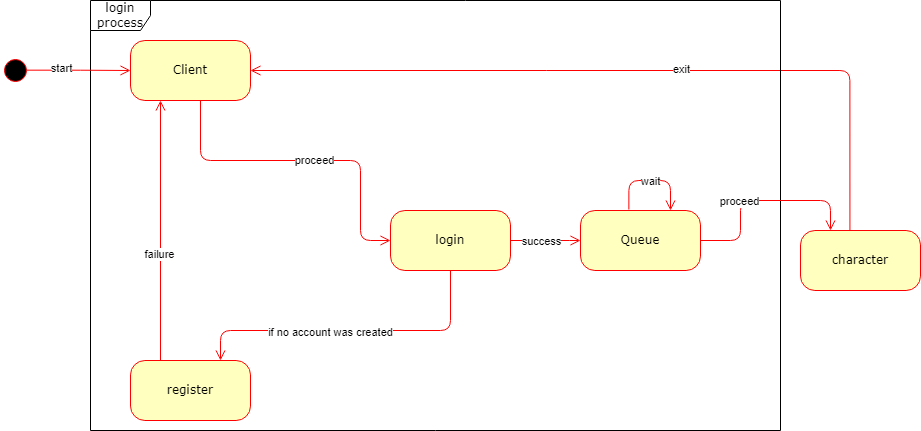
#### This section will focus on process decomposition:



#### And to get even further into the functioning of the status, this state-diagram should help:

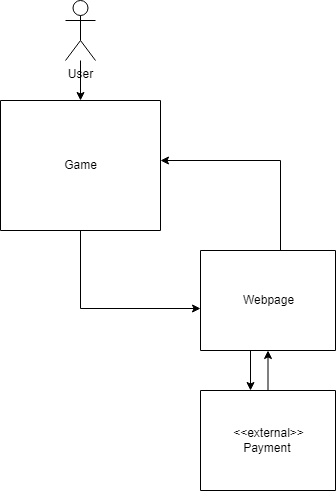


#### Whilst we are looking at how the server should deal with states, let’s look at how the Server handles login attempts when the servers are full (QA-3):



## 3.4: Deployment / Physical view

#### This graphic should help further depict the general business context of this product:



#### Artefacts relevant in this product are as follows:

|  |  |
| --- | --- |
| Artefact | Description |
| User | User downloads the application and attempts to log in to the game. |
| Webpage | User gets redirected to the webpage to register. After creating an account, he also must verify his email-address.  On his account his account status is displayed.  To access the game, he needs to log in, which is only possible after registering and having an active subscription to have the account status “active”.  As long as the payment has not been verified, the status of the account remains “inactive” and the user cannot access the game. |
| Payment | Users chose a payment option. To handle such a process and make sure it is protected properly, an external service is being used, as they are better equipped to deal with potential security issues.  Once the payment is approved, the payment system reports back to the webpage and the account status of this user is changed to approved. |

## 3.5: Use Cases

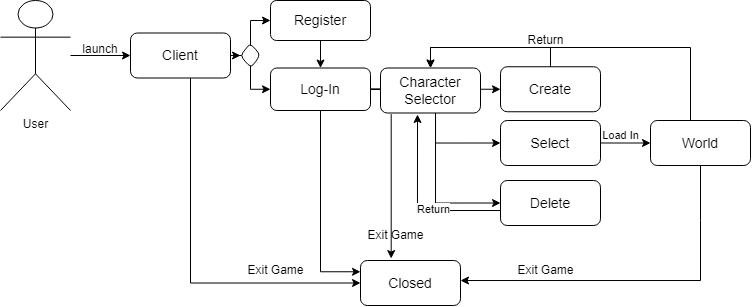
In this section we will try to combine all these views and try to give a summarization understandable for everyone.

## 3.5.1: Use Cases of the Client

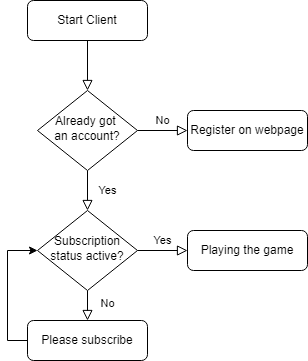
#### Considering the in [1.2 defined functional requirements](#_A_summarization_of) the user is left with (but not exclusively) the following use cases:

|  |  |  |
| --- | --- | --- |
| ID | Use Case | Description |
| UC-1 | Register | The user creates an account |
| UC-2 | Login | The user logs in to the server |
| UC-3 | Create character | The user creates a character. If this process is finished (customization finished and name selected) or canceled, the user will be returned to the character selection [User Interface (UI)](#_Abbreviations_and_Annotations:). |
| UC-4 | Select character | The user picks an already existing character |
| UC-5 | Delete character | The user deletes an already existing character |
| UC-6 | Load in | After selecting a character, the user can select to load in |
| UC-7 | Play game | The user plays the game. |
| UC-8 | Return to character selection | Once in the game, the user is then able to return to the character selection |
| UC-9 | Exit game | The user can close the application. |
| UC-10 | Disconnect | User gets disconnected after attempting to access resources he has no rights to access. A log gets created |

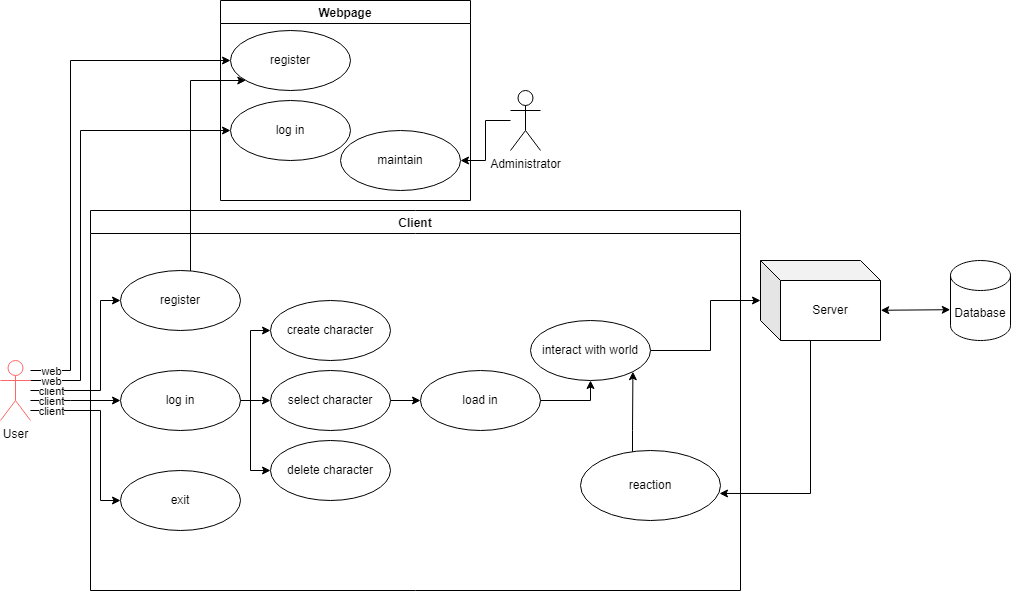
#### These Use-Cases are also depicted in this graphic:



#### A very generalized view of the structure or potential issues looks as follows:



Use-case diagram depicting the users’ potential interactions in the client itself:



# 8. Crosscutting Concepts

## 8.1: Structural Model

Most other Massively Multiplayer Online Games (MMOGs) have a software architecture that consists of either the [Client-Server Architecture](#_Abbreviations_and_Annotations:) (CSA), or the [Peer-to-Peer](#_Abbreviations_and_Annotations:) (P2P)-Architecture.

#### In the Following table we will explain why we chose the Client-Server-Architecture over the Peer-to-Peer-Architecture:

|  |  |
| --- | --- |
| Relevant Aspect | Reasoning |
| Performance | Due to the increased performance that the Client-Server-Architecture provides compared to the Peer-To-Peer (P2P)-Architecture, it is the considered option.  Whilst P2P-Architectures might be favored on very few users, the more users there are, the slower the connection and the Performance will be, as the data will have to travel through all the Clients. |
| Security | Having Central File Servers (CFS) provide a level of access which henceforth increases the security of the data. |
| Scalability | If there is a need to increase the server capacity, this is easier to do in the Client-Server-Architecture. |
| Maintainability | Seeing as the users have their own versions on their client, and only access the server through the internet, CSA has a great advantage when it comes to maintenance. |
| Location | Servers are centralized. This may result into high latency for users that either have a bad connection or are very far from the location of the server. |
| Cost | Central file servers are expensive. |
| Financing | Because the connection to the server can be somewhat regulated, monetarization is possible. |

## 8.2: Architectural and Design Patterns

Now that the structure for this project is settled, let’s look at the patterns used to realize our quality scenarios

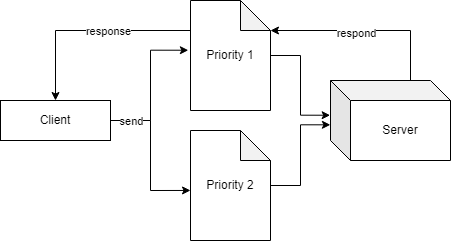
### 8.2.1: Performance

Seeing as we prioritize performance greatly, let’s talk about these first:

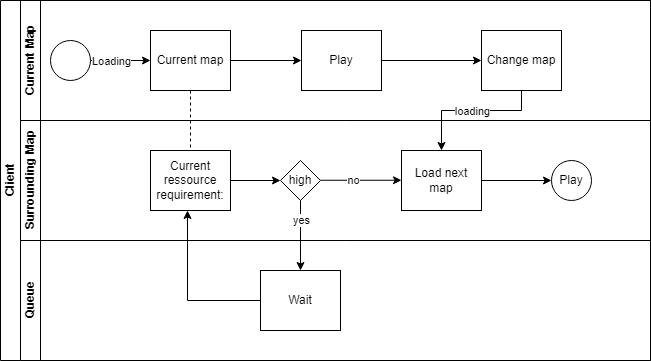
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Correlating Quality Attribute | Keyword | Tactic | Description | Reasoning |
| QA-1 | Latency | Limit event response, prioritize events,  Reduce computational overhead | Limit the event responses and prioritize the incoming requests as well as reducing the overhead delivered in the requests. | By limiting the number of messages received and by limiting the overhead contained in the requests, performance can be increased greatly |
| QA-2 | Peak load | Working set algorithm | To reduce potential loading times, the user loads the surrounding areas whilst playing to reduce general loading times | Reducing loading times is a great way to keep the users immersed in playing the game. In trying to achieve that having him load the surrounding areas in a working set whilst playing on the current map could help immensely. |
| QA-3 | Login time  (Queue) | Bound queue size | Implementing a “First in First out”- queue to hinder server overflow. | If the capacities of the server are exhausted, a queue system is implemented, so that players trying to connect must wait for others to leave. |
| QA-4 | Time-to-reconnect | Load balancer | A load balancer is an intermediary that handles and balances messages originating from a set of clients | To minimize the occurrence of disconnects, the inputs sent by the client towards the server are minimized to important packages only. That way in case of missing packets the connection is rarely broken and can easily be continued. |

### 

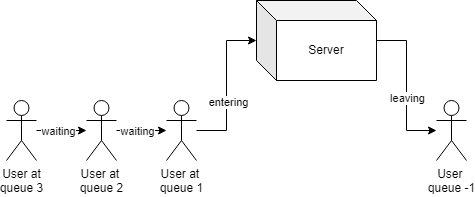
#### QA-1: Prioritization of events would look as follows:



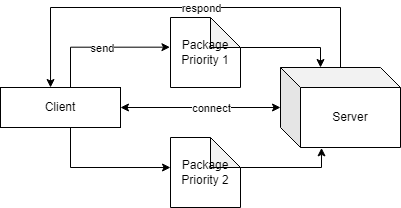
#### QA-2: An example of how we intent to reduce loading times is by loading them whilst the players System is not requiring the resources:



#### QA-3: A typical limit event response in form of a queue looks as follows:



#### QA-4: By focusing on the maintained connection of client and server and by prioritizing responses disconnects can be reduced:

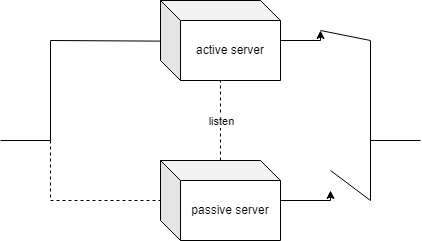


### 8.2.2: Availability

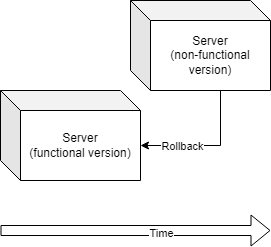
Availability, while not being considered by companies as a big deal, is sort of a given when it comes to users. To make sure users don’t specifically complain about this, let’s make sure they don’t have to with the following table:

|  |  |  |  |
| --- | --- | --- | --- |
| Correlating Quality Attribute | Keyword | Tactic / Pattern | Reasoning |
| QA-5 | General downtime | Passive Redundancy Pattern | Seeing as we want to maximize the availability of our system, we want to ensure that even in case of a system emergency (Server crash etc.), a spare version is functional to take over. |
| QA-6 | Rollback | Preparation and repair tactic 🡪 Rollback | Having to perform a rollback means loading up a previous version of the system in a “good” state. This implies that the servers save their data and therefore their state, which must be loaded back up then. |
| QA-7 | Reboot time | Passive Redundancy Pattern | To minimize the time, it takes for the new version to be launched on the server, it’s best to load up the new version on one of the passive “listeners” and then launch these up to take over. This way the clients won’t have to wait for the server to finish loading the new version, due to that happening in advance. |

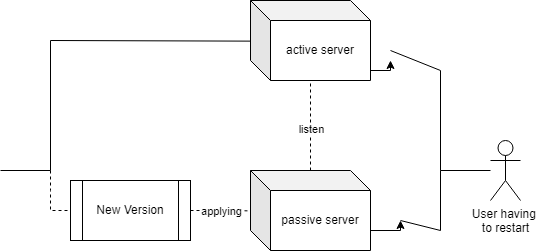
#### QA-5: To minimize downtime a second server is always at the ready, listening to the server:



#### QA-6: In case a problem with a new version occurs, we want to be able to perform a rollback without losing too much data in the process:



#### QA-7: The new version is being applied to the passive server and once the update is on the server, the passive and the active servers swap and the user must restart downloading the new version:

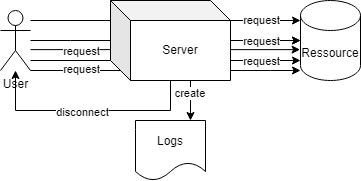


### 8.2.3: Security

As this project is rather large in its entirety, some aspects of Security can’t be left unspoken of. One major security concern is Denial-of-service:

|  |  |  |  |
| --- | --- | --- | --- |
| Correlating Quality Attribute | Keyword | Tactic | Reasoning |
| QA-8 | Protection of [DoS](#_Abbreviations_and_Annotations:) | Detect Service Denial,  Identify Actors,  Revoke Access,  Audit | If the system detects inputs similar to historical profiles of Denial-of-Service (DoS) attacks, the actor is identified via his digital footprint and has his access revoked. In addition to this a record of the user and the system actions is being kept to better identify if this was a one-time occurrence or a recurring attack. |
|  |  |  |  |

#### QA-8: The following graphic depicts how the implementation of these tactics looks like:



# 10. Quality

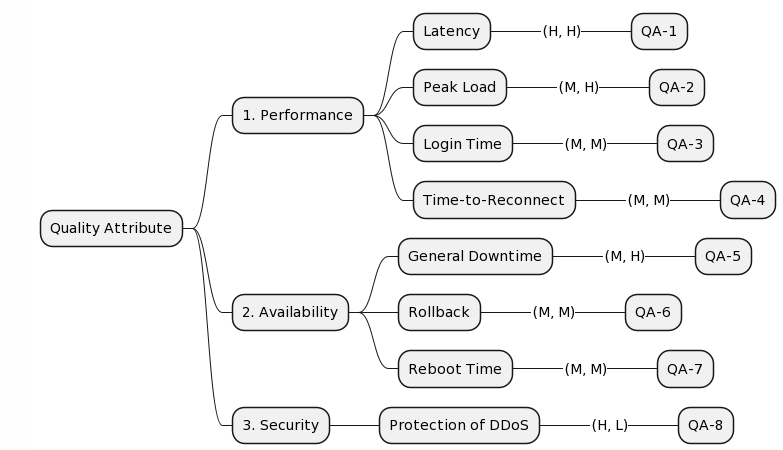
## 10.1: Quality Tree

Whilst quality attributes such as usability and modifiability are important, they are not the main scope of this project. [Performance](#_8.2.1:_Performance) is the most important aspect. [Availability](#_8.2.2:_Availability) might be just as important for a user-focused product. [Security](#_8.2.4:_Security) is also an important factor as soon as multiple users are involved.

To prioritize the following [Quality Attributes](#_1.3:_Quality_Goals), they were prioritized with scores (H being high, M being medium, L being low priority) and sorted by:

1. Importance to the [Architect / Developer](#_Glossary:)
2. Relevance to the [User](#_Abbreviations_and_Annotations:)

#### The following graphic should help better understand the priority of the quality attributes for the project:



#### The following table helps better understand the Quality Attributes and Use-Cases:

|  |  |  |  |
| --- | --- | --- | --- |
| ID | Quality Attribute | Scenario | Associated Use Case |
| QA-1 | Performance | As the user performs inputs on the server, the general latency is measured. | UC-6, UC-7, UC-8 |
| QA-2 | Performance | As the user transitions from one area to the next, the time it takes for this loading process is measured. | UC-6, UC-7, UC-8 |
| QA-3 | Performance | Due to a heavy straining of the servers’ capacities, the users wait in a queue until the server has the capacities to log them in. | UC-2 |
| QA-4 | Performance | Requests made by the user cannot be answered properly due to a high load, so the connection is lost. The general time to reconnect is measured. | UC-7, UC-6 |
| QA-5 | Availability | The game is not available to be played due to updates / general outages. | UC-2, UC-9 |
| QA-6 | Availability | Due to incompatibilities in the data, the server must perform a rollback. The server will shut down for this and players won’t be able to log in. | UC-2, UC-10 |
| QA-7 | Availability | Because of a quick reboot of the server, the game is unavailable for the user. | UC-2, UC-10 |
| QA-8 | Security | A user sends multiple harmful requests. The server then disconnects that user and creates a log entry documenting this attempt | UC-10 |

## 10.2: Quality Scenario

#### The Following Table shows some of the Quality Scenarios that should not be left unspoken:

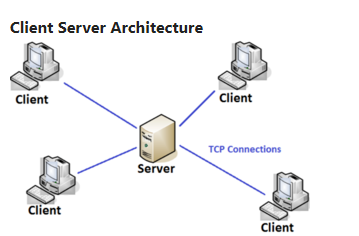
|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| *ID* | *Source* | *Stimulus* | *Artifact* | *Environment* | *Response* | *Response-Measure* |
| QA-1 | User | Plays | Client | Normal circumstances | Inputs are being handled | Latency |
| QA-2 | User | Loading in a different area | Client | High Load Balance. | Loading of the new area | Time of loading |
| QA-3 | User | Logs In | Server | System under high Load Balance | User logs in | Time |
| QA-4 | User | Loses Connection | Server | High Load Balance | User gets reconnected | Time to reconnect |
| QA-5 | System | Is unavailable | Server | Normal  circumstances | The server launches back up | General downtime (yearly) |
| QA-6 | Server | Requires a rollback | System | Normal Circumstances | Server loads a previous state and launches back up. | Time required from the start of the process until the availability of the game |
| QA-7 | Server | Requires a reboot | System | High Load Balance | The server shuts down and reboots | Time taken for this process |
| QA-8 | System | Detects multiple Inputs | Server | High Load Balance | Disconnects user causing inputs and creates a log entry | Interval of entries |

# Alternative Sources:

**Client Server Architecture:**

Client Server Architecture is a computing model in which the server hosts, delivers and manages most of the resources and services to be consumed by the client. This type of architecture has one or more client computers connected to a central server over a network or internet connection.

A typical structure of the most basic Client Server Architecture is depicted in the Graphic below:

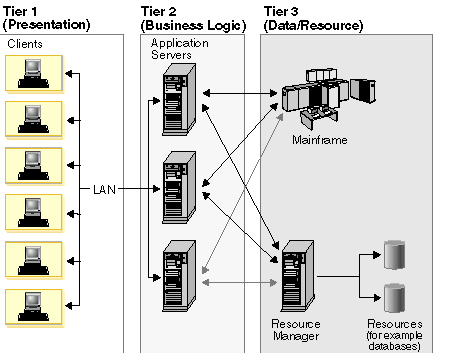


(Source: <https://practice.geeksforgeeks.org/problems/explain-client-server-architecture> )

Another common usage of the CSA would be the Three-tier Client Server Architecture, which involves:

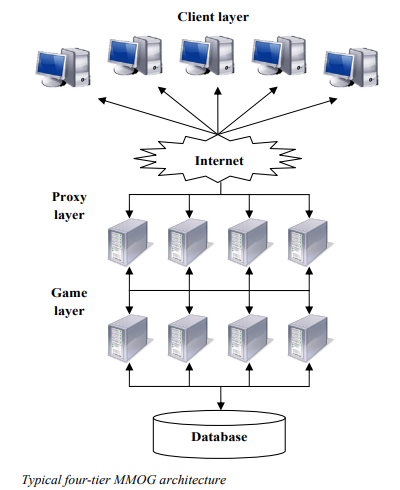
* A client that interacts with the user
* An application server that contains the business logic of the application
* And a resource manager that stores data

The following Graphic shows the general Structure of such a architecture:



(Source:<https://www.ibm.com/docs/en/txseries/9.1?topic=SSAL2T_9.1.0/com.ibm.cics.tx.doc/concepts/c_three_tierd_clnt_sevr_arch.html>)

The common usage in MMOGs is the so called four-tier server architecture, which is depicted in the graphic below:



(Source: <https://www.cs.ru.nl/bachelors-theses/2006/Martijn_Moraal___0131903___Massive_Multiplayer_Online_Game_Architectures.pdf>)

The four layers used in such an Architecture are: The Client Layer, the Proxy Layer, Application/Game Layer, and the Database Layer

**Client Layer:**

* Players are considered as single Clients, connected to the server through a client application on their own device
* User has [User Interface (UI)](#_Abbreviations_and_Annotations:)
* Inputs are being transmitted to the server -> alterations are being received by the user (from the server)
* Number of Interactions between Client and Server minimized to not waste bandwidth
* Client uses motion prediction to present “smooth playing” when in reality updates are only received in intervals

**Proxy Layer:**

* Considered a bridge between [Client Layer](#_Client_Layer:) and [Game Layer](#_Application/Game_Layer:)
* Handles connections between clients and servers
* Handles packet-transport (forwards packets to the appropriate servers in the game layer)
* Can function as a form of firewall (and therefore also be the option for payment-based games -> only paying customers can access)

**Application/Game Layer:**

* Heart of the operation: All requests and commands are handled at this layer
* Processing, verification, computing of changes and updating the game state for those that are affected
* Objective: Maintaining states of all players and objects in the world
* Manages positions of Players and controls players, [non-playable characters](#_Abbreviations_and_Annotations:), treasures, weather, and other data

**Database Layer:**

* Player moves, Items, Objects, Monsters etc. are all considered data elements included in the database
* Ideally: entire game database in memory -> increased performance
* Reality: Requires too much CPU -> transaction time still increasable

## Advantages and Disadvantages for CSA:

**Advantages for CSA:**

* Increase in Performance over P2P
* Central File Servers -> level of access -> better security
* P2P-Networks suffer in Performance the more users there are
* CSA are more scalable
* Distributed Model -> Maintenance advantage (Replacement, Repair and Relocation of servers possible while clients remain unaffected) = Encapsulation

**Disadvantages for CSA:**

* Implementation of Central File Servers expensive
* Overloaded Servers: attempting to accumulate resources simultaneously will overload the servers
* Centralized -> on a critical server failure whole system will crumble

**Result:**

Due to the increase in Performance resulting from CSA and the greater opportunities for commercialization and the major implemented options for Security, CSA is the preferred option chosen for this project.

# Sources:

The Following List includes Sources used for Research purposes, or Sources used for creating content included in this Software Architecture:

<https://moodle.hs-worms.de/moodle/course/view.php?id=2983>

<https://app.diagrams.net>

<https://docs.arc42.org/home/>

<https://ubm-twvideo01.s3.amazonaws.com/o1/vault/gdc2017/Presentations/Clarke-Willson_Guild%20Wars%202%20microservices.pdf> (Microservices)

<https://www.cs.ru.nl/bachelors-theses/2006/Martijn_Moraal___0131903___Massive_Multiplayer_Online_Game_Architectures.pdf>(Bachelor Thesis considering P2P / C-S-Application)

<https://cio-wiki.org/wiki/Client_Server_Architecture#:~:text=Client%20Server%20Architecture%20is%20a,a%20network%20or%20internet%20connection>. (Client Server Architecture)

<https://practice.geeksforgeeks.org/problems/explain-client-server-architecture> (Client Server Application Graphic)

<https://www.ibm.com/docs/en/txseries/9.1?topic=SSAL2T_9.1.0/com.ibm.cics.tx.doc/concepts/c_three_tierd_clnt_sevr_arch.html> (Three-tiered C/S-Architecture)

<https://www.oxfordlearnersdictionaries.com/definition/english/user-interface?q=User+Interface>

<https://www.oxfordlearnersdictionaries.com/definition/english/non-player-character?q=non-player+character>

<https://www.google.com/search?q=denial+of+service+definition&client=opera-gx&hs=maO&ei=-bylYt7CKuuTxc8PsNiE0As&oq=Denial+of+service+defini&gs_lcp=Cgdnd3Mtd2l6EAMYADIFCAAQgAQyBggAEB4QFjIGCAAQHhAWMgYIABAeEBYyBggAEB4QFjIGCAAQHhAWMggIABAeEA8QFjIICAAQHhAPEBYyCAgAEB4QDxAWMgYIABAeEBY6BwgAEEcQsAM6CAgAEB4QFhAKSgQIQRgASgQIRhgAUIQWWPkpYJoyaAJwAXgAgAFXiAGxBJIBATeYAQCgAQHIAQTAAQE&sclient=gws-wiz>