Table of Contents

Document History

Rev.	Author	Date	Changes
1	Luis Traffa	9th July 2022	Everything

1 Introduction

1.1 Overview

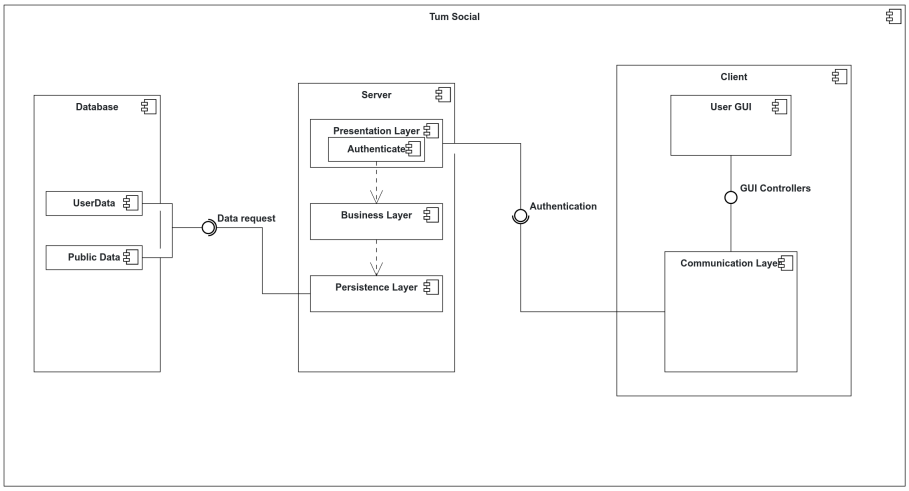
The system is essentially a standard client server application. The client is a webbrowser like Firefox or Chrome, which connects to the TUM SOCIAL server and requests services from it. The server handles these rquests and sends the responses back to the client.

2 Design Goals

When designing the system we had certain non-functional requirements in mind. We determined the most important of these to be scalability, usability and maintainablility. These fundamental priciples shaped the development of our system and are easily noticed by users. Our registration process is very simple and new users can sign up with just a couple clicks and have full access to all the features right away. Furthermore, the database is highly capable and enables the network to expand and support a growing number of users. The code is also easy to read as the system uses the MVC pattern with various different controllers for different features.

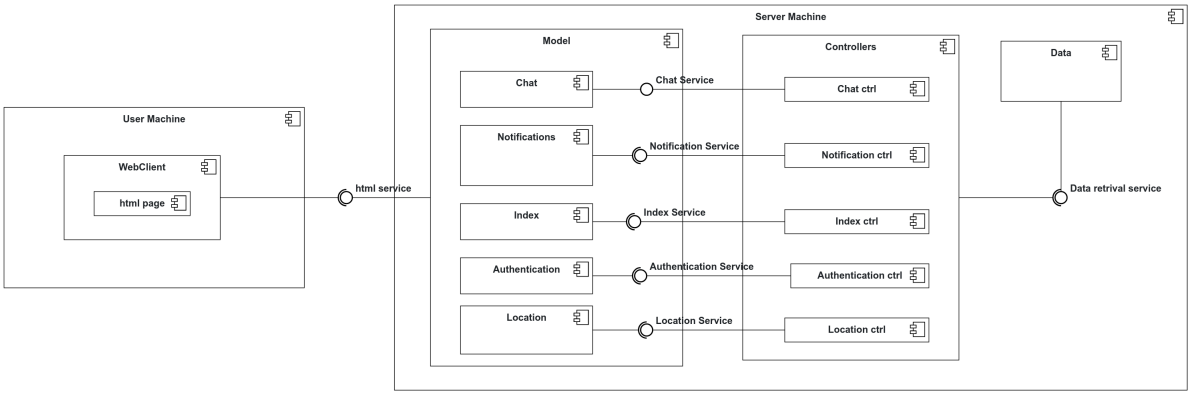
3 Subsystem Decomposition

This section describes the decomposition of the system into subsystems and the services provided by each subsystem. The services are the seed for the APIs detailed in the Object Design Document. The system can generally be seperated into 3 distinct components. The database, server and client. The database stores all data and offers a facade interface to let the server request data. The server is a standard spring boot application, which handles all of the clients requests. The client provides the end user with a GUI, that makes use of various controllers for different functions, to make requests from the server. The following diagram is a rough representation of the system seperated into its basic components/subsystems.



4 Hardware/Software Mapping

In physical space the subsystems are distrubuted on 2 seperate hardware nodes. The client machine and the server machine. The client machine only contains the view, which is displays the current state of the model and is updated by controllers when the user sends requests to the server. There exist several different kinds of controllers. One for each core aspect of the system like the profile or chat. The controllers provide services that the models need to get the data and display it. The database is also located on the server node. The protocol employed is the standard https, which comes with Spring boot.



5 Persistent Data Management

Every class of the model has its own table in the database, that stores the values and references to other related objects. We use standard relational database model. For simplicity we use SQLite, but the system is designed in a way, that a change to a more performant database system, can be achieved easily.

6 Access Control and Security

The biggest potential security problem is a user being able to log in as another user and be able to read all the sensitive data. Therefore we implemented a hashing algorithm, that hashes the passwords, so that even if the server were to be hacked, the attackers would not be able to assume control of the profiles. Another susceptible feature is the friend system. Only friends should have more communication features activated between them.

	Friend	Stranger
Friend	extended communication	not possible
Stranger	not possible	limited communication

7 Global Software Control

The design of TUM Social is highly centralized. All data is stored on the server and if someone were to seize control of the server, that person would have full control over the network. Furthermore, the system is a polling based design, which means that the server is continuously polling for new requests and doesnt shut-off when it does not need to do anything, like an event- based system would. Requests are initaited when an end-user makes a request through a running client.

8 Boundary Conditions

First of all the server needs to be started, as it handles all of the communication within the network. When the server is running clients can start up and try to connect to it. If a connection cannot be successfully made, clients will simply have to retry until the server responds. Ideally the server is running continously, but it is possible that at somepoint the server stops answering requests. This may be because of a "distributed denial-of-service" attack or technical problems. When this happens the clients will not be able to communicate with each other. All connected clients will experience a lack of response from the server. The clients will have to wait until the server is operational again. If at some point the system is supposed to shut sown indefinitly, the clients should end the connection with the server before it will shutdown.