Semester Project 3
Re-exam

Dimitrios Antoniou, 293165

Marwan Summakieh, 285805

Supervisor: Jakob Knop Rasmussen

[Number of characters]

Software Engineering

4th and 6th Semesters

11/8/2021

Table of content

Table of Contents

List of figures and tables

Abstract

The purpose of this project is to develop a system by which the users can communicate with one another using messages.
The system is a web application created to be a messaging system created with the intention to avoid the business logic that the current messaging services have.
The system is implemented using three tier architecture basis and designed as heterogenous system, it has a user-friendly interface and developed using .net framework for the first tier and second tier consists of business logic and it is implemented with java, Data access object with model entities are presented in the data tier. the system used MS SQL server to store entities.

Introduction

Done by: Dimitrios

The primary focus of this project was to create an easy-to-use, clear chatting application, which would target an audience of people who are searching for a simple, broad texting service.

In order to meet this demand, the application needed to be stripped of any unnecessary embellishment, and complex features, and only keep the core values of a messaging application. An additional feature was the option for a user to search the name of another user in the system via a search bar tool.

Among the most prominent issues was the consumption of the web services by the client. Since we had already established a connection between the Server and the Data Tier, all that was missing was linking the Client with the Server, a task that was implemented successfully for the most part.
The chat application is not able to process and send alternative types of messages (pictures, gifs, videos).

The requirements were met to a fair degree, given the size of the group and the working time frame. The use case diagram and the use case descriptions were derived from them and provided us with the skeleton of our system.

Requirements

Done by: Dimitrios

An important thing to note in this project, is that the more important requirements were met with success, even though most of the lower priority ones were not. The system should be able to send messages and store them in the database, through the server. The ability for users to be able to log in and out and create an account was the most vital functionality, as it would allow them to actually use the system.
The most challenging part of the project was to send messages back and forth, while also keeping track of the sender and the receiver.
The overall goal of this system was to create a simple, almost basic, messaging service, which would offer a certain degree of privacy and security, but also contrast
the hugely complicated systems of today.

After the requirements were formulated, the use case diagram was put together, after which the use case descriptions themselves were structured, by deriving from the latter.

Functional requirements

Critical

i. As a user, I would like to create an account, so I can use the system
ii. As a user, I would like to log in the system with my account, in order to use the app

High

I. As a user, I would like to to search for another user that is using the system
II. As a user, I would like to send messages to other users who are using the system
III. As a user, I would like to store information about myself, so that other users can see it

Low

I. As a user, I would like to be able to see a list of users, so I can see the total amount of users in the system
II. As a user I would like to be able to send pictures.
III. As a user I would like to be able to send a voice message
IV. As a user, I would like to be able to view the profiles of other users, so I can see some information about themselves
V. As a user I would like to have secure connection so no one can snoop through my messages

Non-Functional Requirements

I. As an owner i would like the system to be split into 3 parts so if one part fails the system doesn’t shut down
II. As an owner I would like the system to be built using java and c#
III. As an owner I would like the the system to use sockets
IV. As an owner I would like the system to use web services
V. As a user I would like to have an easy interface to use

Analysis

Done by: Marwan

The analysis consists of scenarios, use case diagrams and use case descriptions. The use case diagram is the depiction of what a system can do for the users. Use case diagrams are based on the scenarios so these two are connected to each other. A scenario is a representation of what is going to happen when someone uses the system. Furthermore, use case descriptions have been made for each use case of the

Scenarios

Here only one scenario is presented and that is “Login” scenario of the system.

Login scenario:

User enters credentials in the specific fields username and password, and click login button then the system checks if there is an existing account with the entered parameters. If so, the system accepts and enables the chat, if not the system sends an error.

User case diagram

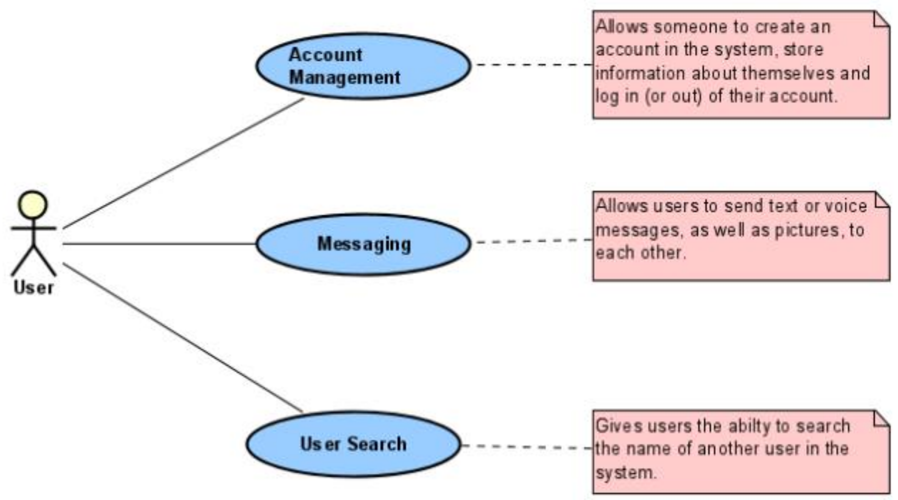
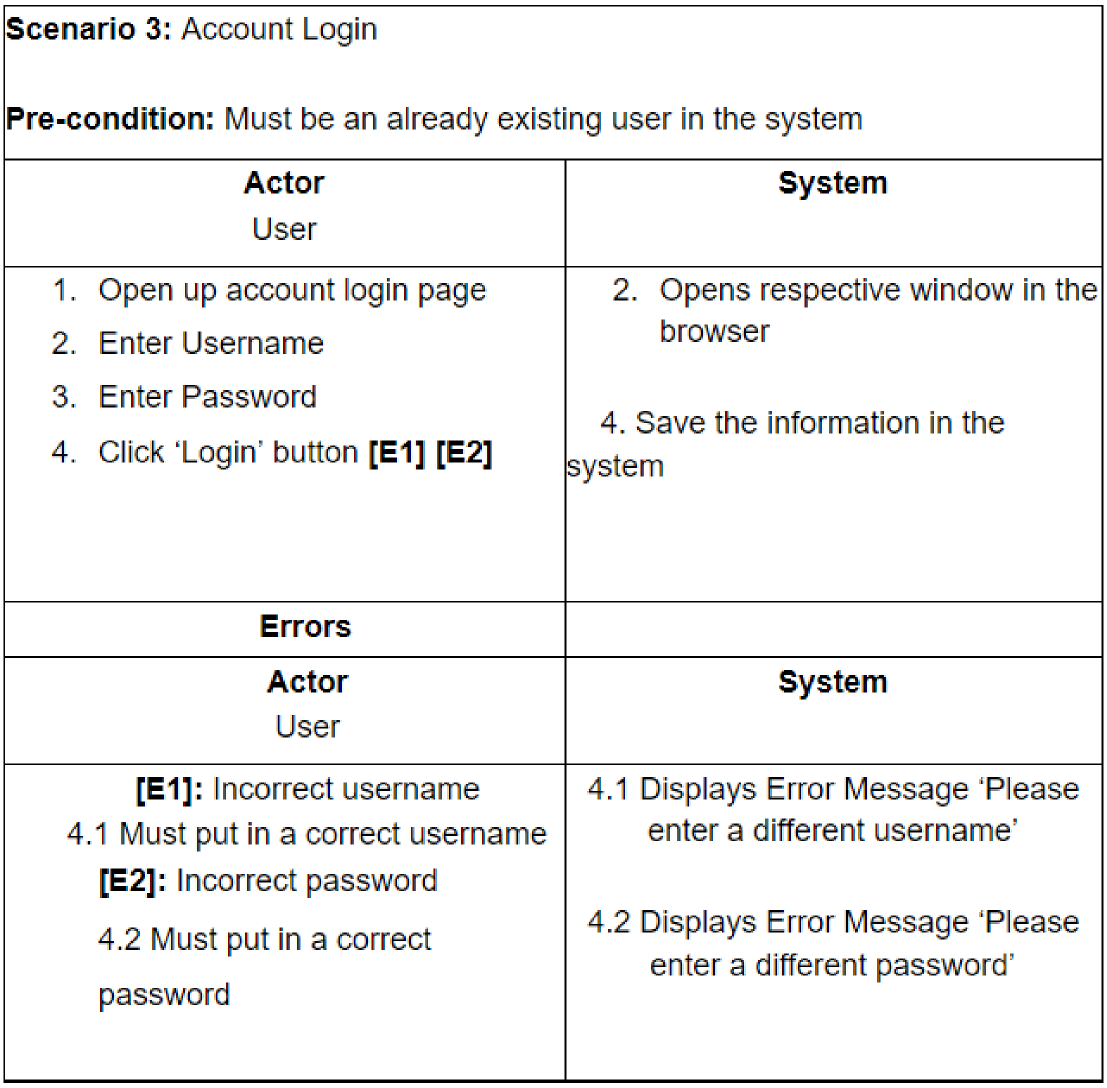

Account Management
Allows someone to create an account in the system, store information about themselves and log in (or out) of their account.
User
Messaging
Allows users to send text or voice messages, as well as pictures, to each other.
User Search
Gives users the abilty to search the name of another user in the system.

Figure 1: Use case Diagram

in Figure 1 - shown above - the use cases were rated by following the requirements. there is one actor the user who will send messages, search for other users and manage the account details

Use case description

Based on the use case diagram shown above, several use case descriptions were made for this subchapter. Only one-use case description will be shown as an example



Scenario 3: Account Login **Pre-condition:** Must be an already existing user in the system	
Actor User	**System**
1. Open up account login page 2. Enter Username 3. Enter Password 4. Click ‘Login’ button **[E1] [E2]**	2. Opens respective window in the browser 4. Save the information in the system
Errors	
Actor User	**System**
[E1]: Incorrect username 4.1 Must put in a correct username **[E2]:** Incorrect password 4.2 Must put in a correct password	4.1 Displays Error Message ‘Please enter a different username’ 4.2 Displays Error Message ‘Please enter a different password’

Figure 2: Login use case description

Design

Conceptual Diagram

Done by: Marwan

The conceptual diagram depicts the relationships between classes. As shown below the three-tier architecture system visualizes how the tiers are interacting with each other. Starting from the third tier, which contains data access object together with model Entities which will be represented as tables in the database.

Business Logic Tier (Tier 2) consists of many services that provide the logic of the system. All business logic operations

are implemented in this tier. Web Services are also presented in this tier. Presentation tier (Tier 1) occupies the top level, which consists of the client that consumes the web services

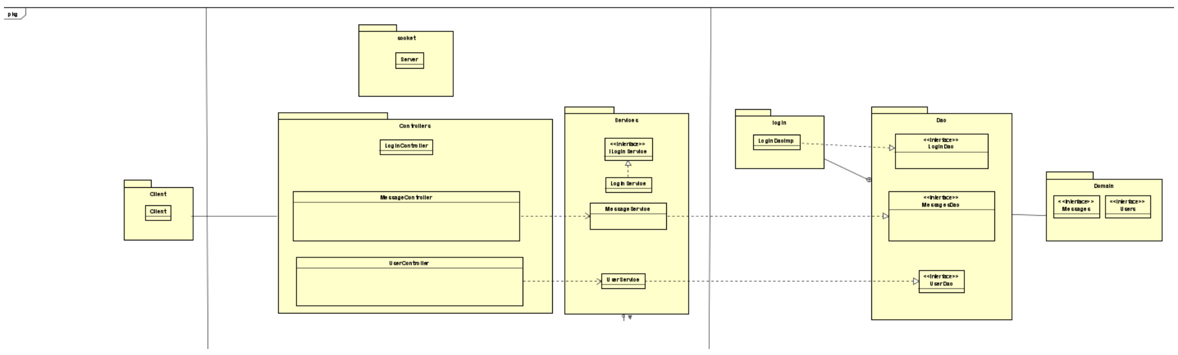


Figure 3: conceptual diagram

Database

Done by: Dimitrios

For this semester project, we were given the choice to decide on what to choose our database on our own. We chose to use a cloud service database named “MSSQL server”. We chose this cloud service for the multitude of different services it provides.

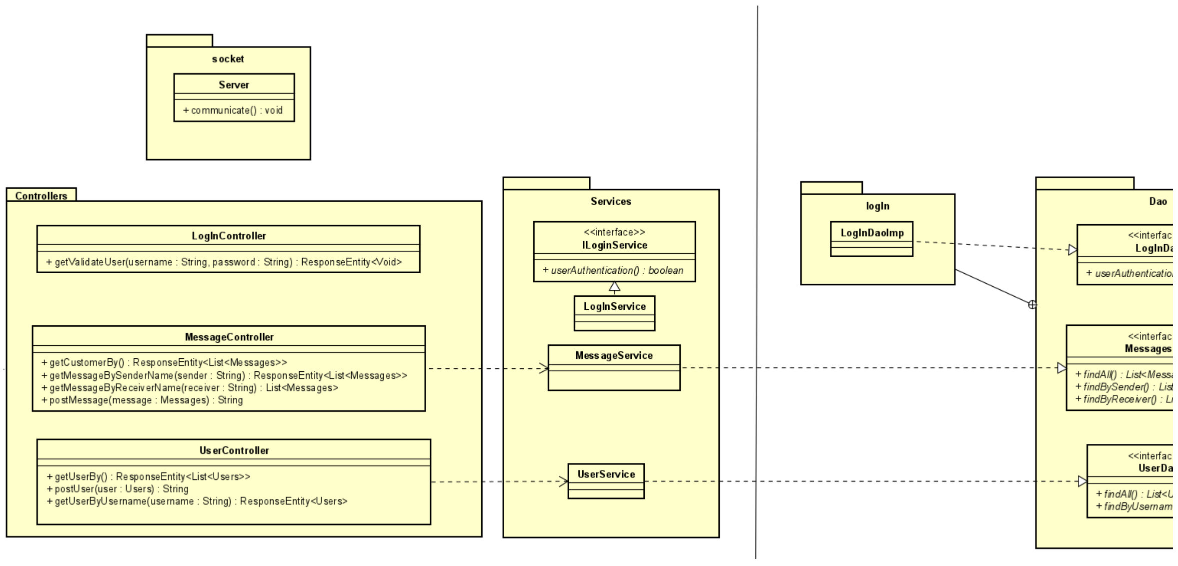

socket
Server
+ communicate() : void
Controllers
LogInController
+ getValidateUser(username : String, password : String) : ResponseEntity<Void>
MessageController
+ getCustomerBy() : ResponseEntity<List<Messages>>
+ getMessageBySenderName(sender : String) : ResponseEntity<List<Messages>>
+ getMessageByReceiverName(receiver : String) : List<Messages>
+ postMessage(message : Messages) : String
UserController
+ getUserBy() : ResponseEntity<List<Users>>
+ postUser(user : Users) : String
+ getUserByUsername(username : String) : ResponseEntity<Users>
Services
<<interface>>
ILoginService
+ userAuthentication() : boolean
LogInService
MessageService
UserService
logIn
LogInDaoImp
Dao
<<interfac
LogInDa
+ userAuthenticatio
<<interfac
Messages
+ findAll() : List<Messa
+ findBySender() : List
+ findByReceiver() : Li
<<interfac
UserDa
+ findAll() : List<U
+ findByUsernam

Figure 4:Business Logic and connection with the Data Tier

Business Logic Tier

The Business Logic Tier of the project is, in essence, the Server side where the endpoints, in the form of Web Services, are implemented via the *Controllers*. The *Services* include important methods which are being utilized by two of the *Controllers*. The names of the *Controllers* themselves are self-explanatory; The *LoginController* posts the mapping which is required by the *Login* page over in the Presentation Tier for an existing user to gain access to the system. The *MessageController* has the *GET* requests which get the messages, by sender, receiver. The *UserController* provides the mapping to get users, get users by their username and posts the mapping for a user, which is required to create an account for the system. Finally, the *Socket* class receives a string text sent by the sockets on the client side, and sends a string text back to the Client.

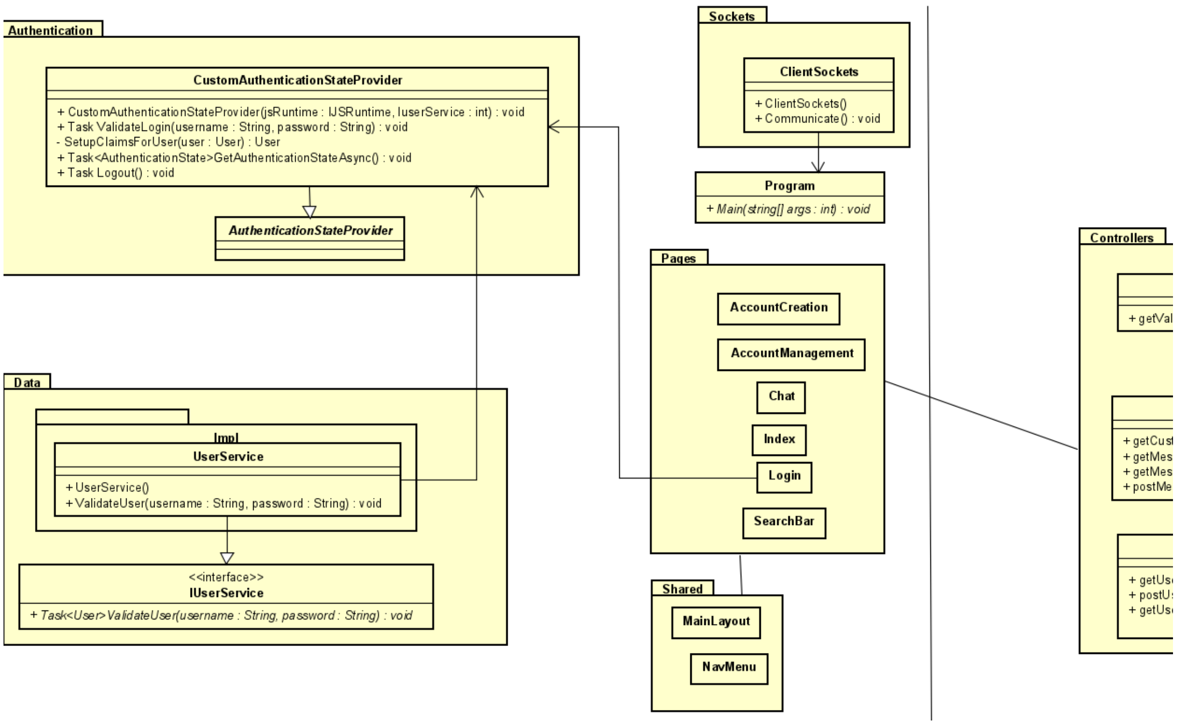

Authentication
CustomAuthenticationStateProvider
+ CustomAuthenticationStateProvider(jsRuntime : IJSRuntime, IuserService : int) : void
+ Task ValidateLogin(username : String, password : String) : void
- SetupClaimsForUser(user : User) : User
+ Task<AuthenticationState>GetAuthenticationStateAsync() : void
+ Task Logout() : void
AuthenticationStateProvider
Data
Impl
UserService
+ UserService()
+ ValidateUser(username : String, password : String) : void
<<interface>>
IUserService
+ Task<User> ValidateUser(username : String, password : String) : void
Sockets
ClientSockets
+ ClientSockets()
+ Communicate() : void
Program
+ Main(string[] args : int) : void
Pages
AccountCreation
AccountManagement
Chat
Index
Login
SearchBar
Shared
MainLayout
NavMenu
Controllers
+ getVal
+ getCust
+ getMes
+ getMes
+ postMe
+ getUs
+ postU
+ getUs

Figure 5:Presentation Tier and connection with the Business Logic

Presentation Tier

Done by: Dimitrios

Regarding the presentation tier, we opted for Blazor. It was a framework that we both were familiar with, and one that we had the most documentation and information available about. Since Blazor utilizes Razor pages to display the content, most of the background processes are being done within other classes. The pages are directly connected to the *Controllers*, as seen above, to consume the Web Services. They take the *URL* given in the controllers and display, receive and send the corresponding data accordingly. The *CustomAuthenticationStateProvider* is the class that essentially provides the authentication and security level of the users who exist in the system. The *Login* page is directly connected to it. The *Login* is also connected to the *MainLayout* page, where it is being displayed in the form of a Top Bar. In the *Data* package, we have the *UserService* which implements the *IUserService* interface, and is also being implemented in the

CustomAuthenticationStateProvider class*.* Their purpose is to ensure the *Login* page does the authentication for the users properly and correctly. Finally, the *ClientSocket* class is responsible for sending a string text to the sockets to the 2nd Tier.

Class diagram

Done by: Marwan

Presentation Tier where the actual client is interacting with the system to perform a certain operation. In this tier some graphical interfaces have been implemented to provide one interface which is the user as there is no other actors in the system. Presentation Tier communicates with the second Tier, where the system controls application functionality by performing detailed operations. In the business Tier, 3 controllers will provide HTTP verbs in order to expose web services. The controllers coordinate with the services that add the logic needed for each function. Here is where the "heavy lifting" of the application is done. It uses data that it has retrieved from the database or been sent by the user via the user interface. Services apply a layer of abstraction to make the system open to use external systems in the future. The services pull the data from the third tier, where data access object packages provide CRUD operations for persisting and retrieving data from the actual database which is located on the cloud. Model package is the collection of objects that will be mapped to the database using Hibernate ORM framework.

Positive and negative sides of using three Tier architecture

Advantages:

Three Tiers architecture is designed for CRUD applications where the database is the center of the application design which fit perfectly in the manner of this project.

Disadvantages:

There is ambiguity about where application, abstraction and domain level abstraction should go, and in practice, the application logic gets mixed with domain logic. Each layer is coupled to the layer below it

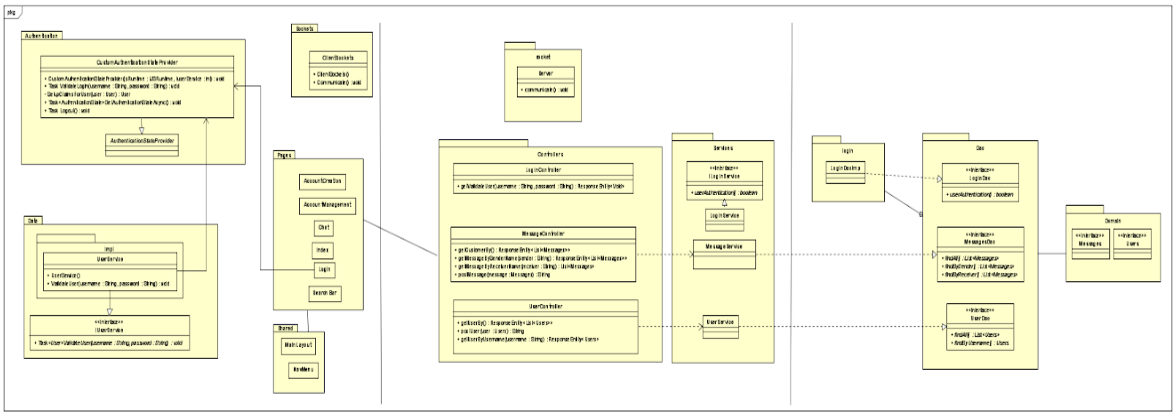


Figure 6: Class diagram

Socket

Done by: Marwan

One of the requirements for this project is to design and implement protocols for sockets. For this purpose, socket is implemented on sending welcome messages to the user when login. TCP/IP is used for communication between first tier and second tier but only for sending greeting message, how ever this wasn't properly implemented.

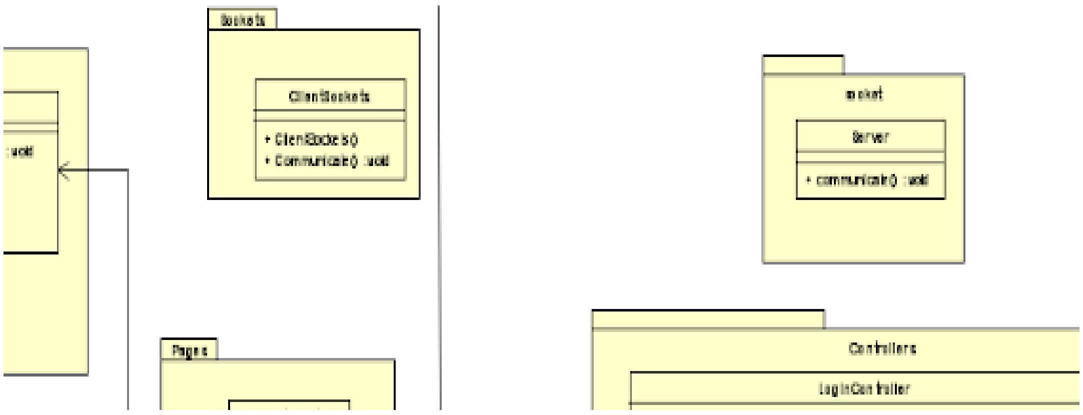


Figure 7: Sockets

SMTP (simple mail transfer protocol) was used to send users verification emails.

Security

Done by: Marwan

The security and integrity of any application important. Without putting any thought into securing one's application the person will reach a point when the application fails or because of consumer feedback, people will stop using it. Since this application is dealing with user sensitive information, we should assume that at points there will be threats regarding the safety of the users' data. One way to protect them is by keeping the sensitive data in our secure cloud database. If the attackers gain access to our user sensitive information, it will breach the system's security standards and would put at risk all of other users. There are many types of cybersecurity threats, but the most common ones are:

- Denial-of-service (DoS) and distributed denial-of-service (DDoS) attacks.
- Man-in-the-middle (MitM) attack.
- Phishing and spear phishing attacks.
- Drive-by attack.
- Password attack.
- SQL injection attack.

Denial-of-service attacks are the ones carried out by people who intend to make a service or machine (computer) unavailable for intended users. These attacks flood the target system with unneeded requests making the service temporarily or indefinitely unavailable for legitimate requests.

Man-in-the-middle attacks are carried out when two parties are communicating with each other through a seemingly secure connection, but when in-fact they are being transmitted through a third party. This is specifically a threat while the two intended parties are using an unsecure WiFi connection where the attacker is in range to intercept the messages being sent and relay them to the intended destination. With this kind of an attack the two parties think they are messaging each other, while in fact their messages are being relayed or possibly altered.

Phishing and spear phishing attacks are attempts to steal sensitive information from unaware users. Attackers that use "Phishing attacks" disguise themselves as trusted service providers to steal credit card details, passwords or another user relevant information. Usually, attackers send out mass emails to people using the same site in order to trick them into opening a provided link that will transfer them to their desired service. When clicked, the link will transfer them to a different webpage. Although the webpage looks and acts legitimate, it in fact serves the attackers needs. The users enter the information needed (Passwords, usernames, credit card details etc.) and send it to the site. While the user thinks its information is being treated securely, it never made it to the desired destination rather to the attacker that scammed the users. Password attacks or password cracking is the process of getting a hold of a user's password by trying different possible combinations. There are different methods on how to get a user's password. Some of them include Dictionary attack, Brute force, Rainbow table, Phishing etc.

The Brute force method means simply guessing passwords until you find the correct one. This might take minutes and it might take weeks. Due to the reason passwords are getting longer and more complex, "Brute force" gets less useful because it takes more and more time to actually "crack "the password. People are switching to the "Dictionary " method to make it more simple to use.

The Dictionary method requires a text file containing words people might use as passwords. The longer this file is, the higher the chance that one of them might be the correct one. Although this works in cases when users choose simple passwords, word combination or simply substituting the letter “I” for a 1. But in a perfect scenario where users choose safe passwords by combining capital letters with small letters and adding multiple digits throughout the password, this method starts to get less and less useful due to the fact that much more possibilities occur and need to be added to the text file.

SQL injection attacks are targeting the user’s database in order to get valued information like passwords, credit card details editing the syntax that either stores or retrieves values from the database. A common way of doing this would be retrieving information by means of the Logon function. When the user is asked for a username and password to log in, an attacker might pass a sql query to the username tab and retrieve wanted information from there.

STRIDE model

Done by: Marwan

STRIDE is a threat model developed by Microsoft to identify computer security threats in six different categories. Thus, creating the mnemonic STRIDE

Spoofing is when a person or program masks their identity and impersonates a different person or service.
Tempering is when information is being manipulated in order to gain advantage over someone or to change privileges. If an attacker would be able to change the URLs the redirected site might contain harmful scripts or malware.
Repudiation is when someone challenges the authenticity of a certain thing, like a key card.
Information disclosure is when an attacker can see information otherwise denied to them. These would be personal pictures or data files that would be kept private. In this project this

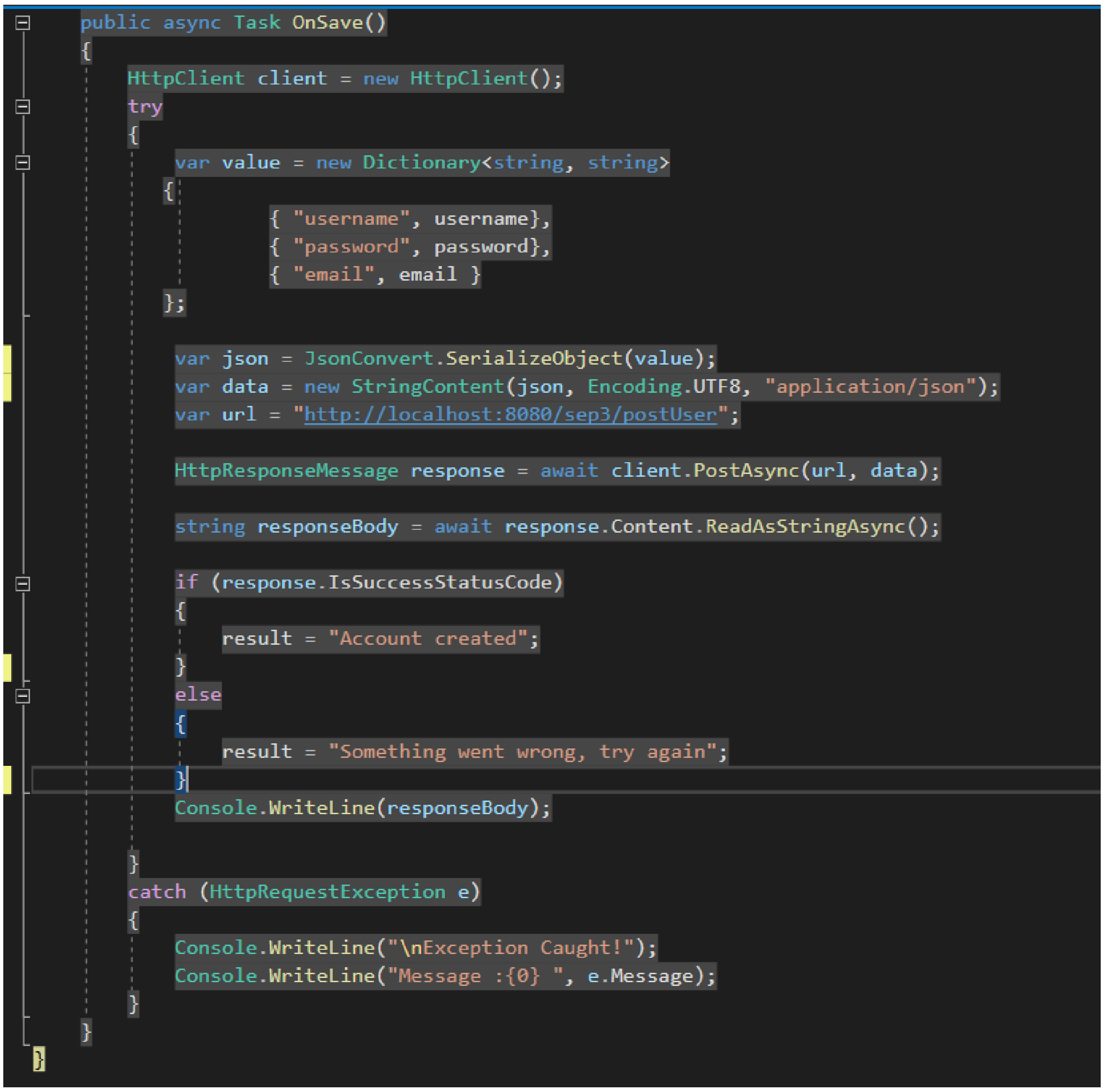
is a major issue given the system will contain personal information and communications that people want to stay hidden and not seen or read by others.

Denial of service is a major risk for this application since its main purpose is to communicate with other people. These kinds of attacks are usually carried out by activists that want to keep people silent or deny them the opportunity to express themselves by spreading their opinions to others.

Elevation of privilege is when a person receives more privileges in a system where he in fact should not have the privileges. This may result in a simple user receiving admin privileges and is able now to add, edit or copy information they otherwise would not get access to. In this application this will create a major issue as no one, even administrators, should be able to see or alter the any user's information.

Implementation

Done by: Marwan

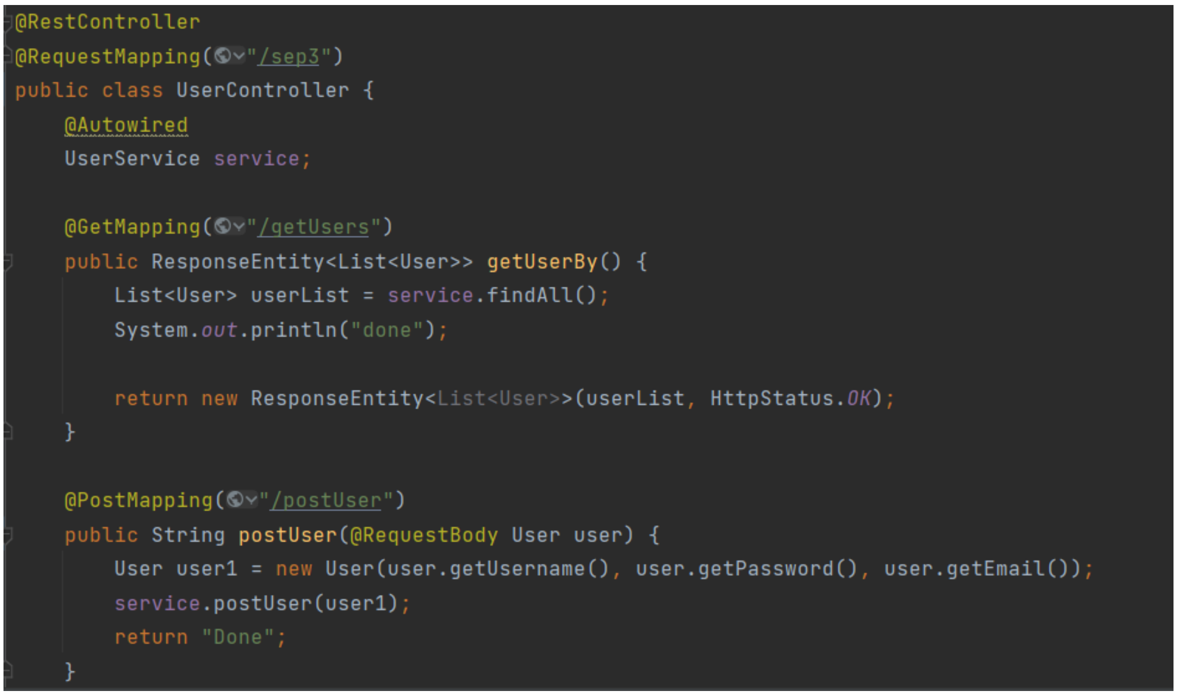


```
public async Task OnSave()
{
    HttpClient client = new HttpClient();
    try
    {
        var value = new Dictionary<string, string>
       {
                { "username", username},
                { "password", password},
                { "email", email }
       };

        var json = JsonConvert.SerializeObject(value);
        var data = new StringContent(json, Encoding.UTF8, "application/json");
        var url = "http://localhost:8080/sep3/postUser";

        HttpResponseMessage response = await client.PostAsync(url, data);

        string responseBody = await response.Content.ReadAsStringAsync();

        if (response.IsSuccessStatusCode)
        {
            result = "Account created";
        }
        else
        {
            result = "Something went wrong, try again";
        }
        Console.WriteLine(responseBody);

    }
    catch (HttpRequestException e)
    {
        Console.WriteLine("\nException Caught!");
        Console.WriteLine("Message :{0} ", e.Message);
    }
}
}
```

Figure 8: AccountCreation FrontEnd

In the figure above the onSave method is triggered when the user clicks the button to create an an account. It first registers that this is an HttpClient and initialize it on click. This takes care of creating and destroying the client after the process is finished to avoid port exhaustion when many clients are created, and they were handled properly.

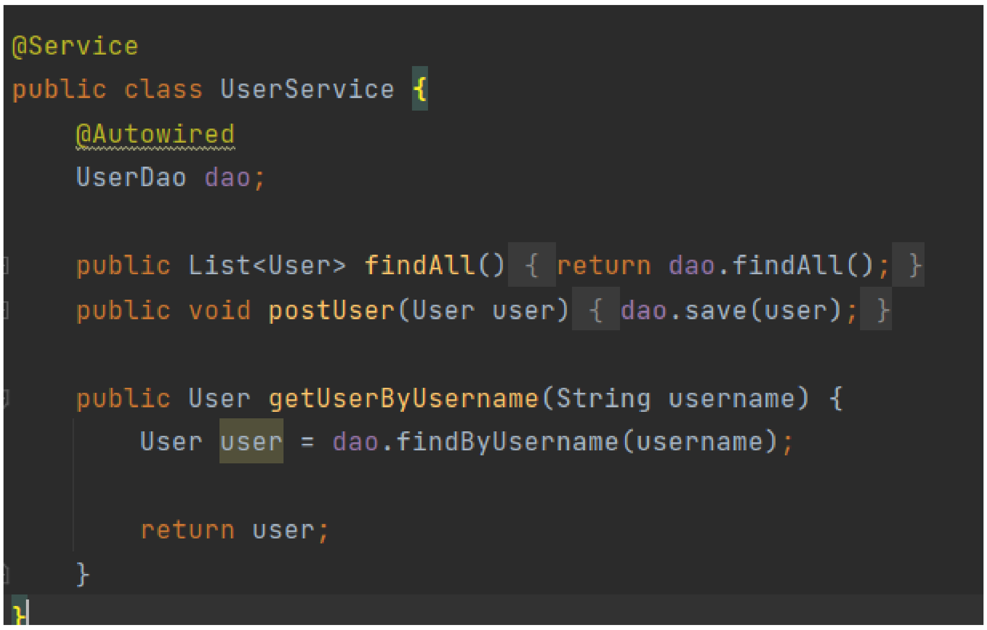
Surrounded by try and catch, the client now creates a json file containing the information entered by the user. Then it calls the appropriet web service to send this information to the server side attached with the contents.
If the server returns that everything is okay the user gets notified that the account was created otherwise it would send a warning that something is wrong.



```
@RestController
@RequestMapping("/sep3")
public class UserController {
    @Autowired
    UserService service;

    @GetMapping("/getUsers")
    public ResponseEntity<List<User>> getUserBy() {
        List<User> userList = service.findAll();
        System.out.println("done");

        return new ResponseEntity<List<User>>(userList, HttpStatus.OK);
    }

    @PostMapping("/postUser")
    public String postUser(@RequestBody User user) {
        User user1 = new User(user.getUsername(), user.getPassword(), user.getEmail());
        service.postUser(user1);
        return "Done";
    }
```

Figure 9: UserController

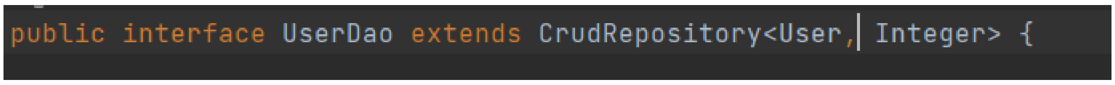
The figure above illustrate how User will be created in the server side. After receiving the information through the restAPI the postUser method creates an instance of User and saves it locally before it gets pushed to the database.



```java
@Service
public class UserService {
    @Autowired
    UserDao dao;

    public List<User> findAll() { return dao.findAll(); }
    public void postUser(User user) { dao.save(user); }

    public User getUserByUsername(String username) {
        User user = dao.findByUsername(username);

        return user;
    }
}
```

Figure 10: User Service class

Service class is used to write business logic in a different layer, separated from @RestController class file. It calls the UserDao interface which is extends CRUD repository which will act as a database repository making CRUD operations possible from this class.



```java
public interface UserDao extends CrudRepository<User, Integer> {
```

Figure 11: UserDao

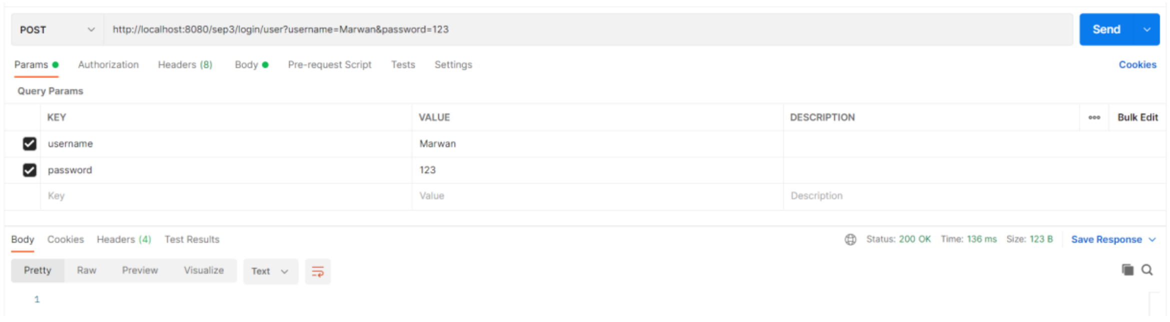
When the UserDao is called the user will be created in MS SQL server

Test

Done by: Marwan

Black box testing

Restful web services used in the system, has been tested using postman to perform manual test to ensure that everything is working as it was intended.
In the figure below, several HTPP verbs have been tested.



POST http://localhost:8080/sep3/login/user?username=Marwan&password=123 Send

Params ● Authorization Headers (8) Body ● Pre-request Script Tests Settings Cookies

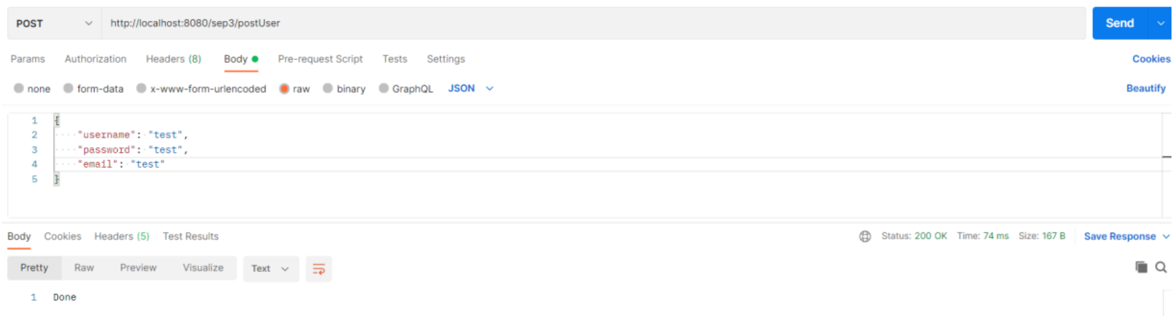
Query Params

	KEY	VALUE	DESCRIPTION
✓	username	Marwan	
✓	password	123	
	Key	Value	Description

Body Cookies Headers (4) Test Results — Status: 200 OK Time: 136 ms Size: 123 B Save Response

Pretty Raw Preview Visualize Text

Figure 12:login



POST http://localhost:8080/sep3/postUser Send

Params Authorization Headers (8) Body ● Pre-request Script Tests Settings Cookies

none form-data x-www-form-urlencoded ● raw binary GraphQL JSON Beautify

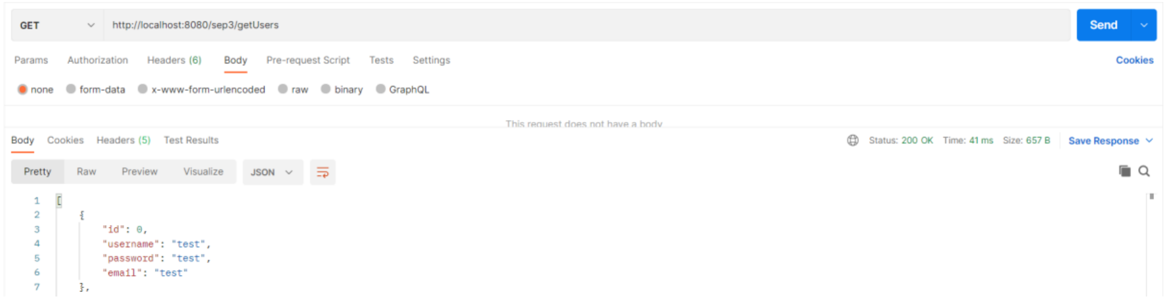
```
{
    "username": "test",
    "password": "test",
    "email": "test"
}
```

Body Cookies Headers (5) Test Results — Status: 200 OK Time: 74 ms Size: 167 B Save Response

Pretty Raw Preview Visualize Text

```
Done
```

Figure 13: creating user



GET http://localhost:8080/sep3/getUsers Send

Params Authorization Headers (6) Body Pre-request Script Tests Settings Cookies

● none form-data x-www-form-urlencoded raw binary GraphQL

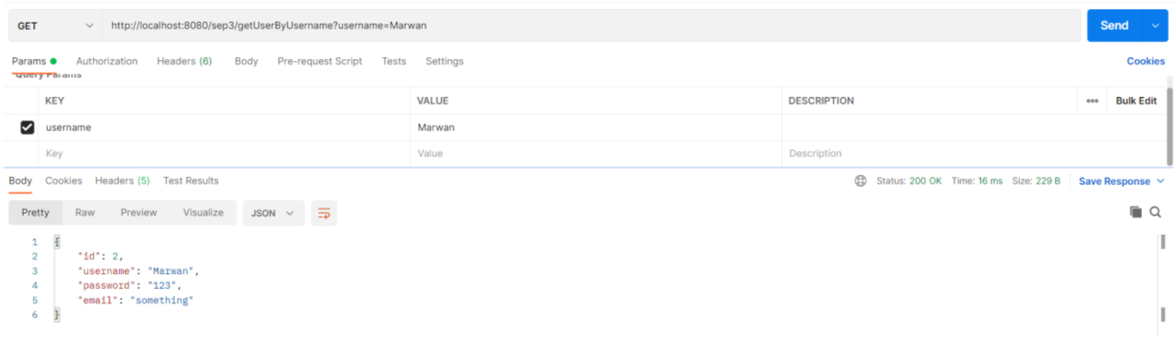
This request does not have a body

Body Cookies Headers (5) Test Results — Status: 200 OK Time: 41 ms Size: 657 B Save Response

Pretty Raw Preview Visualize JSON

```
[
    {
        "id": 0,
        "username": "test",
        "password": "test",
        "email": "test"
    },
```

Figure 14:get all the Users



GET http://localhost:8080/sep3/getUserByUsername?username=Marwan Send

Params ● Authorization Headers (6) Body Pre-request Script Tests Settings Cookies

Query Params

	KEY	VALUE	DESCRIPTION
✓	username	Marwan	
	Key	Value	Description

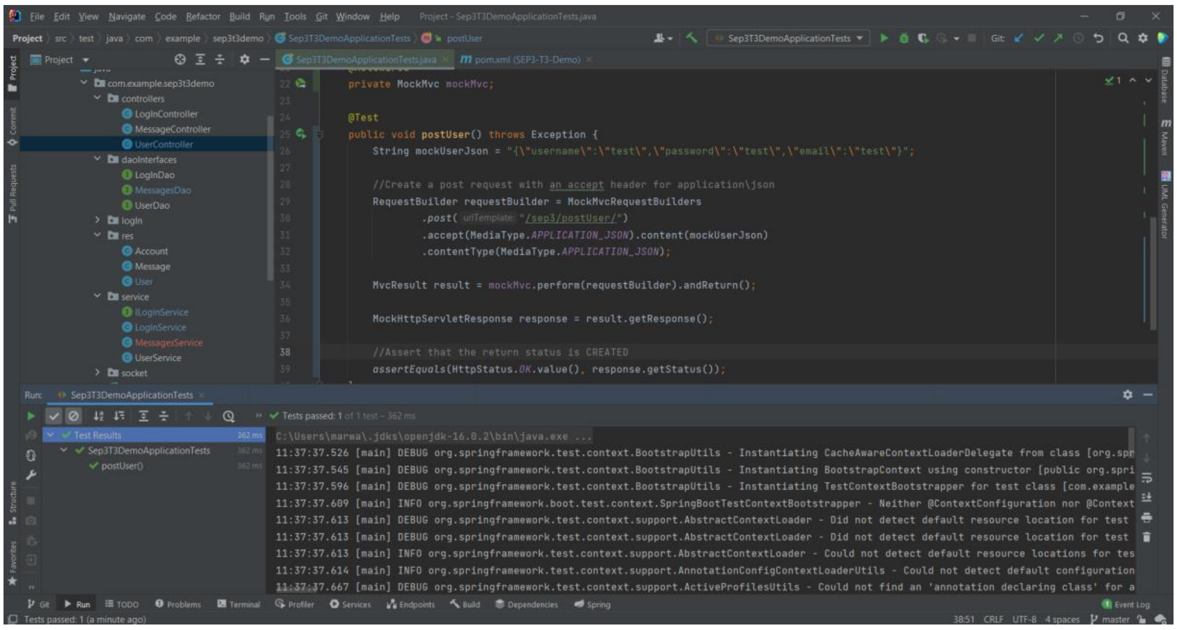
Body Cookies Headers (5) Test Results — Status: 200 OK Time: 16 ms Size: 229 B Save Response

Pretty Raw Preview Visualize JSON

```
{
    "id": 2,
    "username": "Marwan",
    "password": "123",
    "email": "something"
}
```

Figure 15: getting a user by username

Junit testing



```java
    private MockMvc mockMvc;

    @Test
    public void postUser() throws Exception {
        String mockUserJson = "{\"username\":\"test\",\"password\":\"test\",\"email\":\"test\"}";

        //Create a post request with an accept header for application\json
        RequestBuilder requestBuilder = MockMvcRequestBuilders
                .post( urlTemplate: "/sep3/postUser/")
                .accept(MediaType.APPLICATION_JSON).content(mockUserJson)
                .contentType(MediaType.APPLICATION_JSON);

        MvcResult result = mockMvc.perform(requestBuilder).andReturn();

        MockHttpServletResponse response = result.getResponse();

        //Assert that the return status is CREATED
        assertEquals(HttpStatus.OK.value(), response.getStatus());
```

```
Tests passed: 1 of 1 test – 362 ms
C:\Users\marwa\.jdks\openjdk-16.0.2\bin\java.exe ...
11:37:37.526 [main] DEBUG org.springframework.test.context.BootstrapUtils - Instantiating CacheAwareContextLoaderDelegate from class [org.spr
11:37:37.545 [main] DEBUG org.springframework.test.context.BootstrapUtils - Instantiating BootstrapContext using constructor [public org.spri
11:37:37.596 [main] DEBUG org.springframework.test.context.BootstrapUtils - Instantiating TestContextBootstrapper for test class [com.example
11:37:37.609 [main] INFO org.springframework.boot.test.context.SpringBootTestContextBootstrapper - Neither @ContextConfiguration nor @Context
11:37:37.613 [main] DEBUG org.springframework.test.context.support.AbstractContextLoader - Did not detect default resource location for test
11:37:37.613 [main] DEBUG org.springframework.test.context.support.AbstractContextLoader - Did not detect default resource location for test
11:37:37.613 [main] INFO org.springframework.test.context.support.AbstractContextLoader - Could not detect default resource locations for tes
11:37:37.614 [main] INFO org.springframework.test.context.support.AnnotationConfigContextLoaderUtils - Could not detect default configuration
11:37:37.667 [main] DEBUG org.springframework.test.context.support.ActiveProfilesUtils - Could not find an 'annotation declaring class' for a
```

Figure 16: Junit testing on post user

JUNIT test has been performed on mock data, as shown above.

Results and Discussion

Done by: Marwan

As a result of this project the 2 tiers forming the back end were working properly, all the created services worked, however the front-end lacks in both functionality and polish. As a whole system it can perform a login operation, create users and send messages.

Conclusions

Done by: Marwan

The System is a web application developed for people to send messages. The Project was not finished, although the finished parts work. Not all the requirements were implemented. But they can be given a better timeframe and resources.

Project future

Done by: Marwan

In the future the project can be expanded to handle more than just text messages, it can also be adopted into an android application and a security layer is necessary for global competition with other similar in functionality systems.

Sources of information

Banger, D., 2014. A Basic Non-Functional Requirements Checklist « Thoughts from the Systems front line.... Available at: https://dalbanger.wordpress.com/2014/01/08/a-basic-non-functional-requirements-checklist/ [Accessed January 31, 2017].

Business Analyst Learnings, 2013. MoSCoW : Requirements Prioritization Technique — Business Analyst Learnings. , pp.1–5. Available at: https://businessanalystlearnings.com/ba-techniques/2013/3/5/moscow-technique-requirements-prioritization [Accessed January 31, 2017].

Dawson, C.W., 2009. *Projects in Computing and Information Systems*, Available at: http://www.sentimentaltoday.net/National_Academy_Press/0321263553.Addison.Wesley.Publishing.Company.Projects.in.Computing.and.Information.Systems.A.Students.Guide.Jun.2005.pdf.

Gamma, E. et al., 2002. *Design Patterns – Elements of Reusable Object-Oriented Software*, Available at: http://books.google.com/books?id=JPOaP7cyk6wC&pg=PA78&dq=intitle:Design+Patterns+Elements+of+Reusable+Object+Oriented+Software&hl=&cd=3&source=gbs_api%5Cnpapers2://publication/uuid/944613AA-7124-44A4-B86F-C7B2123344F3.

IEEE Computer Society, 2008. *IEEE Std 829-2008, IEEE Standard for Software and System Test Documentation*,

Larman, C., 2004. *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development*,

Mendeley.com, 2016. Homepage | Mendeley. Available at: https://www.mendeley.com/ [Accessed February 2, 2017].

YourCoach, S.M.A.R.T. goal setting | SMART | Coaching tools | YourCoach Gent. Available at: http://www.yourcoach.be/en/coaching-tools/smart-goal-setting.php [Accessed August 19, 2017].

Rohit Joshi, 2015; Java Design Patterns, [Last accessed 14/03/2018] via link;

http://enos.itcollege.ee/~jpoial/java/naited/Java-Design-Patterns.pdf

Thomas Connolly, Carolyn Begg, 2015; Database Systems A Practical Approach to

Design, Implementation and Management, [Last accessed 18/05/2018] via the link;
http://people.stfx.ca/x2011/x2011asx/5th%20Year/Database/Database%20Management%20Textbook.pdf

Hibernate:
https://hibernate.org/orm/documentation/5.4/

Spring
https://docs.spring.io/spring/docs/current/spring-framework-reference/index.html

JPA
https://docs.spring.io/spring-data/jpa/docs/current/reference/html/

Oracle
https://www.oracle.com/technetwork/java/javaee/documentation/index.html

Appendices

Appendix 1 Project Description

Appendix 2 Use case description

Appendix 3 Diagrams

Appendix 4 Scenarios

Appendix 6 Contract