DEPARTMENT OF COMPUTER SCIENCE

The Mango Messenger

DIPLOMA PROJECT

Poznan, 2021

Contents

Partner Details

Mentor's details

First name and surname	Szymon Murawski
Degree	Dr. Inz.
Date and signature	

Team members' details

First name and surname	Petro Kolosov
Course of study	Computer Science
Type of study program	Daytime
Date and signature	

First name and surname	Serhii Holishevskyi
Course of study	Computer Science
Type of study program	Daytime
Date and signature	

First name and surname	Illia Zubachov
Course of study	Computer Science
Type of study program	Daytime
Date and signature	

First name and surname	Arslanbek Temirbekov
Course of study	Computer Science
Type of study program	Daytime
Date and signature	

Chapter 1

Project Assumptions

1.1 Project description

Instant messaging systems achieve a great success and became the main mean of communication between people via an internet. Thanks to the simplicity and quickness of the message exchanging more and more people over the world start to use instant messengers on daily basis. However, such a great attention forces us to discuss another aspect of these systems, an aspect of the information security and user privacy. The high attention and wide usage of the instant messaging systems in both, commercial and non-commercial ways to be a justification for selecting the subject. The subject matter of current thesis is entire communication structure of IMS including cryptography, protocols, data storage, means of communications. As an object of research we consider the entire entity defined as instant messaging system, in context of modern world. Mainly, the research is done using qualitative data gathered from various sources, which listed in the references. We consider qualitative research as most suitable since that problem of security in IMS is quite classic and widely discussed in scientific community. Finally, we design and implement an instant messaging system that copes with the required functionalities and satisfies the defined security requirements, considering previous research.

1.2 Project objectives

1. To analyze the security and user privacy vulnerabilities of the instant messaging system and propose mitigations.

2. To provide the system requirements for instant messaging system, both functional and non-functional.

3. To propose web service (API's) architecture that fits the requirements.

4. To discuss an authorization mechanism that fits the requirements.

5. To discuss E2E Encryption and apply to the system.

6. To implement web service (API).

7. To implement web client.

8. To implement mobile client.

9. To implement desktop client.

Chapter 2

Implelentation

2.1   Project tasks

Task 1.

Task name	Implementation of the web API
Entities involved to solve the task	ASP NET Core, C#, SQL, SignalR, PostgreSQL, Entity Framework, ASP NET Core Identity, JWT Libraries, NUnit, Moq, FluentAssertions, MediatR, AutoMapper, Docker, Github Actions, Azure
Task completion outcomes	Web API deployed to Azure cloud
Star date of task execution	23-Jun-2021
End date of task execution	

Task 2.

Task name	Implementation of the web client
Entities involved to solve the task	Angular 11, TypeScript, Docker, Github Actions, Azure
Task completion outcomes	Web client deployed to Azure cloud
Star date of task execution	5-Jul-2021
End date of task execution	

Task 3.

Task name	Implementation of the mobile client
Entities involved to solve the task	
Task completion outcomes	
Star date of task execution	
End date of task execution	

Task 4.

Task name	Implementation of desktop client
Entities involved to solve the task	ElectronJS
Task completion outcomes	Previously implemented web client successfully converted to desktop app using ElectronJS
Star date of task execution	18-Jul-2021
End date of task execution	18-Jul-2021

Task 5.

Task name	Implement ECDH key exchange CLI
Entities involved to solve the task	ECDiffieHellmanCng
Task completion outcomes	ECDH key exchange CLI has been implemented
Star date of task execution	15-Nov-2021
End date of task execution	21-Nov-2021

2.2   Project implementation

2.2.1   Theoretical assumptions

Researchers at [Hindocha and Chien, 2003] conclude on the following aspects of the usage of IMS in enterprise

- Threats that affect instant messaging already exist today, including worms and vulnerabilities that can give hackers remote access to vulnerable computers.

- End users and corporations should employ basic security practices and products such as intrusion detection and antivirus to mitigate the risk.

- Corporations at the outset should assess whether instant messaging is even a business necessity.

- Enterprise versions of the instant messaging products should be utilized and administrators should be on the lookout for future enterprise security solutions that specifically address instant messaging threats.

2.2.2   Description of facts

Research [Mannan, 2005] provides analysis on the worms and other issues on IMS. Resource [Mannan and Oorschot, 2004] provides comprehensive survey on security aspects of IMS.

2.2.3 Empirical research

Security aspects of communication model. We would like to start our thesis with a general description of the communication structure between an actors among with possible security and user privacy vulnerabilities. Communication model of the instant messaging system is a quite large topic since it implies various protocols, approaches etc, therefore current discussion asserts the communication over HTTP protocol via REST API using JSON data format. Since that main task of this thesis is to implement software components that meet the specified security requirements, namely: *Web Client, Web API (Application Programming Interface), Mobile client, Desktop client* which are considered to be the actors we focus our attention to. The following diagram describes the basic concept of the system and conveys the relationships between the actors mentioned above.
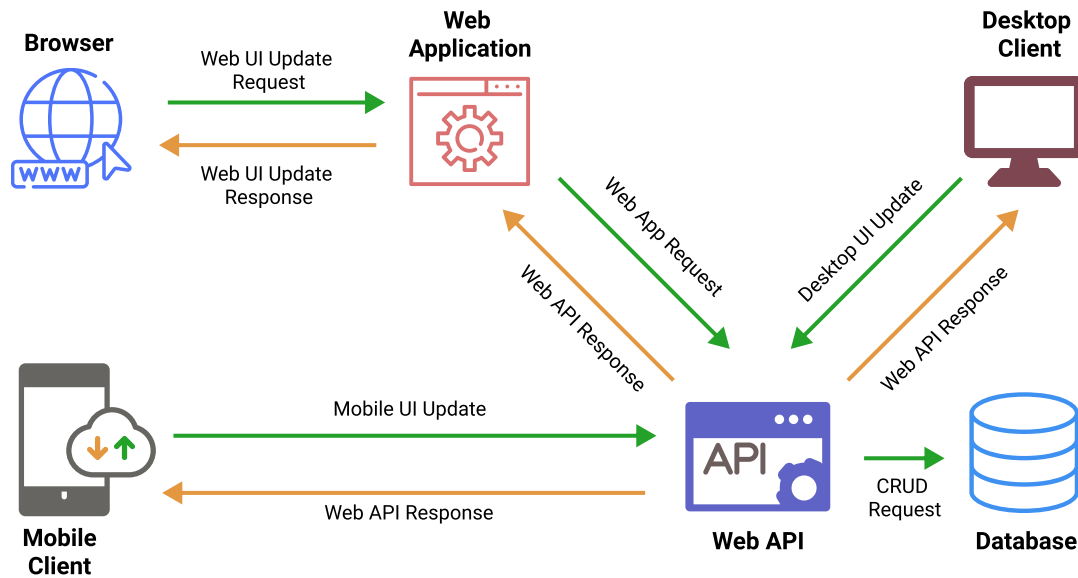

FIGURE 2.1: Database diagram.

Hence, communication between software components is organised as follows

Browser - Web Application - Web API - Database communication model

- Browser downloads application static files from Web Application server.

- Browser sends a request to Web API.

- Web API checks access rights, executes business logic referring to the Database.

- Web API responds to the Browser.

- Browser user interface is being updated.

6

Desktop Client - Web API - Database communication model

- Desktop Client sends a request to update the user interface.

- Web API checks access rights, executes business logic referring to the Database.

- Web API responds to Desktop Client.

- Desktop Client's user interface updated as per response from the Web API.

Mobile Client - Web API - Database communication model

- Mobile Client sends a request to update the user interface.

- Web API checks access rights, executes business logic referring to the Database.

- Web API responds to the desktop Mobile Client.

- Mobile Client's user interface updated as per response from the Web API.

However, such communication models are under possible security vulnerabilities, most of which are already fixed *'out of the box'* in modern web frameworks, so we discuss these are requiring additional attention. The first vulnerability that comes to mind is phishing. An attacker could launch his own web application consuming our web API, therefore it is possible to log user actions and get access to personal data or credentials. Phishing attack could be mitigated using a properly configured Cross-Origin Resource Sharing policy that will restrict the queries from the domains that do not meet the policy. Like, for instance, it is done in our project

```
public static void Configure(
IApplicationBuilder app,
IWebHostEnvironment env)
{
    ...

    app.UseCors(CorsPolicy);

    ...
}

public void ConfigureServices(IServiceCollection services)
{
    ...

    services.AddCors(options =>
```

```
    {
        options.AddPolicy(CorsPolicy, builder =>
        {
            var allowedOrigins = Configuration
                    .GetSection("AllowedOrigins")
                    .Get<string[]>();

            builder.WithOrigins(allowedOrigins)
                    .AllowAnyMethod()
                    .AllowCredentials()
                    .AllowAnyHeader();
        });
    });

    ...
}
```

The next potential vulnerability is improper TLS / SSL certificate configuration or usage of self-signed certificate, to eliminate the vulnerability of the improper TLS certificate, it is recommended to follow the instructions and best practices.

In addition, a potential vulnerability lies in the possibility of SQL injection. The SQL injection vulnerability is eliminated by using parameters in string literals of the SQL query. Also, it is necessary to pay attention to the configuration used ORM.

There is another danger that attacker may receive information about the application infrastructure through the error messages in the response from the server, thus, it is recommended to use the unified response format from the server, which, in case of an error, will not contain the details of the error.

In order to provide proper authorization, it is recommended to use the roles for users in order to restrict an unauthorized access to the resources available only to administrators.

The last but not least possible vulnerability – is a famous worm and virus spreading problem. Obviously, it is not a problem to get rid of the worms in local network with just a few devices connected, however worms are really dangerous for the huge networks, like messenger considered to be. The fight against the worms generally dependent on the end-user's security best practices education (at least user should not use the public unprotected wi-fi networks) and firewall settings of the network. However, the spread of viruses may be mitigated by the certain validation rules upon file upload, like for instance

```
public UploadDocumentCommandValidator()
{
    var allowedExtensions = new List<string>
```

```
{
    "jpg", "JPG", "txt", "TXT", "pdf",
    "PDF", "gif", "GIF", "png", "PNG"
};

RuleFor(x => x.FormFile).NotEmpty();

RuleFor(x => x.FormFile.Length)
    .LessThanOrEqualTo(10 * 1024 * 1024);

RuleFor(x => x.FormFile.FileName)
    .Cascade(CascadeMode.Stop)
    .NotEmpty()
    .Must(t =>
    {
        var validExtension = t.Split('.').Last();
        return allowedExtensions
        .Contains(validExtension);
    }).Length(1, 20);
}
```

2.2.4   System Requirements

Prior to software module implementation, it is essentially important to define the functionality module will obtain. In this section we discuss functional and non-functional requirements of secure instant messaging system from customer's prospective. Generally, there are three forms of software product requirements: business, functional, and non-functional. Business requirements [Dilworth and Kochhar, 2007] typically answer how the product will address the needs of your company and its users. They also reveal the business model of the app and what problems it can solve. Functional requirements [Malan, Bredemeyer, et al., 2001] are about functionalities that will be implemented in the application. Non-functional requirements [Chung et al., 2012] describe how these functionalities will be implemented.

Mostly common and simple way to define software product's functional requirements are User Stories. User stories [Cohn, 2004] should be understandable both to developers and to you as the client, and should be written in simple words. The most popular way of writing a user story is with the following formula:

```
"As a <user type>, I want <goal> so that <reason>."
```

Now, let's group the main features of the application as follows

- Registration

- Authentication

- Managing contacts

- Sending messages and media to individuals

- Creating and managing groups

- Sending messages and media to groups

- Viewing messages history

- Managing profile settings

In order not to overfill the document with the entire list of requirements, system's functional and non-functional requirements moved to the annexes A and B, respectively.

2.2.5   Web service architecture

Motivation.   Implementing the instant messenger system, we consider applying a well-known N-tier Monolithic Architecture [Bucchiarone et al., 2018], which provides a time-proven model that allows software developers to create flexible and reusable applications.

However, during the implementation of monolith it is very important to avoid the cases of crucial over-engineering of the system that leads to useless complication of the code base. For the developers, it is a vital point to follow the KISS [Alwin and Beattie, 2016] and YAGNI [Da Silva et al., 2018] software development principles in order not to reach thousands lines of code in a single class.

One would suggest to use nowadays popular Microservices Architecture, thinking about scalability [Brataas and Hughes, 2004], an ability of the system to handle large numbers of users distributed over geographically large areas without notably affecting the overall performance of the system. However, the effect of Microservices is being felt only for quite large and complex systems, not the case of our yet simple application. According to Martin Fowler,

> you shouldn't start a new project with microservices, even if you're sure your application will be big enough to make it worthwhile.

which is so-called *Monolith first* approach. Makes sense to begin an implementation from *Modular monolith*, a monolith with minimized coupling between the software components, where splitting to microservice won't be a time and financial expensive

operation. Following plot demonstrates the relation between the complexity and profits between monolith and microservices
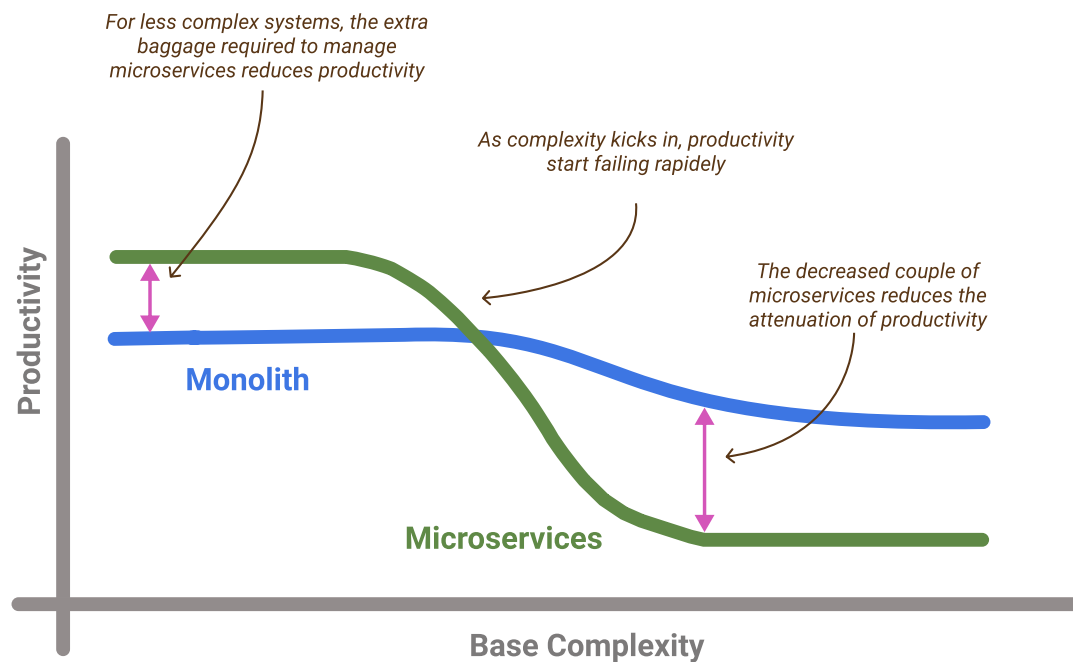

For less complex systems, the extra baggage required to manage microservices reduces productivity

As complexity kicks in, productivity start failing rapidely

The decreased couple of microservices reduces the attenuation of productivity

Productivity

Monolith

Microservices

Base Complexity

FIGURE 2.2: Relation between system complexity and architectures.
Source: Martin Fowler.

A layered architecture usually consists of Presentation layer, Business logic layer, Data access layer. By segregating the project into layers, developers reach the options to modify or add a specific layer without reworking the entire application.

- *Presentation Layer.* Graphic user interface or API gateway.

- *Application Logic.* Encapsulates the means of interaction with user. For example, push-notifications e-mail notification, sms notifications etc.

- *Business Logic.* Encapsulates the logic of clint's request handling. For example, service layer.

- *Data Access Layer.* Responsible for logging, database access and other services required to support Business Logic layer.
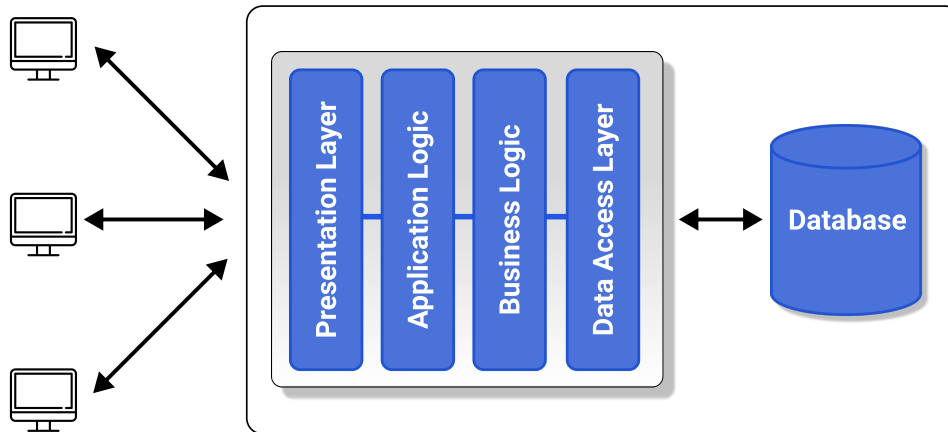
FIGURE 2.3: Monolithic architecture diagram.

Monolithic Architecture: Cons and Props. A monolith is built as a large system with a single code base and deployed as a single unit, usually behind a load balancer. Monoliths offer several advantages, particularly when it comes to operational overhead requirements. Here are some of those basic benefits:

- *Simplicity.* Monolithic architectures are simple to build and deploy. These applications can scale horizontally, by running several copies of the application behind a load balancer. With a single codebase, monolithic apps can easily handle cross-cutting concerns, such as logging, configuration management and performance monitoring. Another advantage associated with the simplicity of monolithic applications is easier deployment. When it comes to monolithic applications, you do not have to handle many deployments but just one.

- *Performance.* Components in a monolith typically share memory which is faster than service-to-service communications using IPC [Proctor, 1999] or other mechanisms.

- *Easier debugging and testing.* In contrast to the microservices, monolithic applications are much easier to debug and test. Since that monolithic application is a single indivisible unit the process of end-to-end testing is much faster.

- *Easier development.* As long as the monolithic approach is a standard way of building applications, any engineering team has the right knowledge and capabilities to develop a monolithic application.

12

However, the drawback of monolithic architectures hides in their tight coupling. Over time, monolithic components and layers become tightly coupled and entangled, effecting management, scalability and continuous deployment. Another disadvantages of the monoliths include:

- *Understanding.* When a monolithic application's code base grows up, it becomes too complicated to understand. Obviously, huge code base of monolithic app is hard to manage therefore.

- *Reliability.* Entire application down may be caused by an error in every single component.

- *Updates.* Single and large code base causes the needs to redeploy an application on every single update.

- *Technology stack.* Technology stack of the monolithic app is limited by the technologies and providers used from the beginning of development. It makes technology stack changes to be expensive in terms of finances and time.

- *Scalability.* Application's components cannot be scaled independently, an entire application should be scaled.

Minimization of services coupling. As we see, the monolith has its own disadvantages, like for instance: understanding the project structure, reliability concerns, technology stack limitations, scalability limitations. Obviously, some of these disadvantages cannot be mitigated because of the nature of the monolith. However, the complexity and coupling problem can be minimized applying certain approaches. Frequent violation of the single-responsibility principle of SOLID during implementing service components in business logics layer causes the over-complication of codebase over the time. The reason is that service components keep the huge number of methods in order to handle all possible CRUD requests to the database without any bounded context. Although, the SOLID rules are very powerful in solving designated code issues, it is necessary to apply them very carefully, since that most of them require high level of abstractions, which increases in size the code base and complicates the solution. Do not overcomplicate the solution without any reason following *Open-Closed Principle*. Schematically, the service entity is as follows
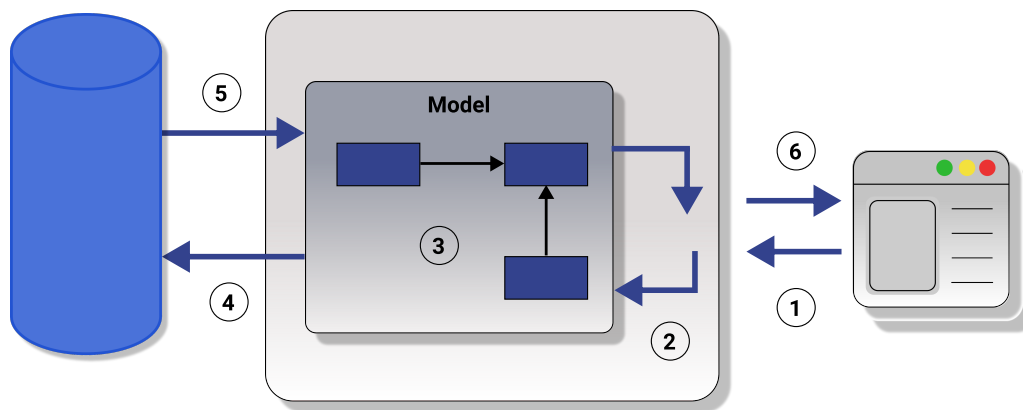
FIGURE 2.4: Service Conceptual diagram. Source: Martin Fowler.

Where the steps are

1. User makes a change in UI.

2. Change forwarded to model.

3. Model executes validation and business logic.

4. Model updates the database.

5. Model reads from database.

6. Service updates presentation from query model.

To minimize the natural disadvantages of the monolithic architecture like complexity and high tight coupling of the components we have to recall the design patterns [Rising, 1998]. In particular, the mediator pattern helps to decouple the components. Mediator – is a behavioral design pattern [Rasche et al., 2016] that allows the communication between two entities, such that entities doesn't know each other. Therefore, the program components depend only on a single mediator instance instead of being coupled to multiple of their colleagues. In context of .NET platform there are many implementations of the Mediator, the most widely known and used is the MediatR, which we use in our project.

Another mindset we are going to use in order to minimize complexity and coupling of monolith is Command-Query Responsibility Segregation (CQRS) principle. In brief, it stands that read (query) and write (command) requests should be segregated

by their responsibilities. Using CQRS and Mediator together greatly simplifies the project structure and minimizes coupling between business logic layer components. CQRS is a pattern that first described by Greg Young [Young, 2010] and its conceptual diagram as follows
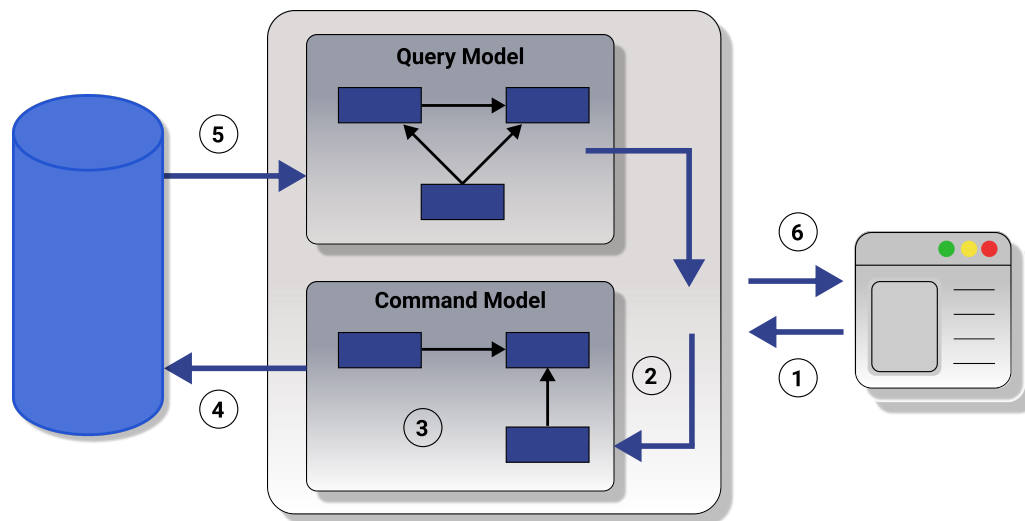

FIGURE 2.5: CQRS Conceptual diagram. Source: Martin Fowler.

1. User makes a change in UI.

2. Application routes information to command model.

3. Command model executes validation and business logic.

4. Command model updates the database.

5. Query model reads from database.

6. Query service update presentation from query model.

Despite these benefits, you should be very cautious about using CQRS. Many information systems fit well with the notion of an information base that is updated in the same way that it's read, adding CQRS to such a system can add significant complexity. I've certainly seen cases where it's made a significant drag on productivity, adding an unwarranted amount of risk to the project, even in the hands of a capable team. So while CQRS is a pattern that's good to have in the toolbox, beware that it is difficult to use well and you can easily chop off important bits if you mishandle it.

As a short conclusion, we may state that CQRS and Mediator pattern will not entirely solve the coupling problems the monolith, however will make project much

more simplistic and intuitively understood. It is worth to keep is simple, even relatively simple project may grow to the sizes of universe without proper architectural solutions.

2.2.6 Authorization mechanism

Motivation. In this section we describe the processes of Authentication and Authorization in the system. It is worth to remember the meaning of Authentication and Authorization definitions. Authentication – is the process of ascertaining that somebody really is who they claim to be [Burrows, Abadi, and Needham, 1989]. Authorization refers to rules that determine who is allowed to do what [Fagin, 1978]. For example, Adam may be authorized to create and delete databases, while Catherine is only authorized to read. The two concepts are completely orthogonal and independent, but both are central to security design, and the failure to get either one correct opens up the avenue to compromise. In terms of web apps, very crudely speaking, authentication is when you check login credentials to see if you recognize a user as logged in, and authorization is when you look up in your access control whether you allow the user to view, edit, delete or create content. Currently, there are two widely-known authentication methods, that are cookie authentication and JWT authentication.

JWT Tokens. JSON Web Token (JWT) is an open standard [Jones, Bradley, and Sakimura, 2015] that defines a compact and self-contained way for securely transmitting information between parties as a JSON object [Jones, Campbell, and Mortimore, 2015]. This information can be verified and trusted because it is digitally signed. JWTs can be signed using a secret with the HMAC [Wang et al., 2004] algorithm or a public/private key pair using RSA [Wiener, 1990] or ECDSA [Johnson, Menezes, and Vanstone, 2001]. Although JWTs can be encrypted to also provide secrecy between parties, we will focus on signed tokens. Signed tokens can verify the integrity of the claims contained within it, while encrypted tokens hide those claims from other parties. When tokens are signed using public/private key pairs, the signature also certifies that only the party holding the private key is the one that signed it. Here are some scenarios where JSON Web Tokens are useful:

- *Authorization.* This is the most common scenario for using JWT. Once the user is logged in, each subsequent request will include the JWT, allowing the user to access routes, services, and resources that are permitted with that token. Single Sign On is a feature that widely uses JWT nowadays, because of its small overhead and its ability to be easily used across different domains.

- *Information Exchange.* JSON Web Tokens are a good way of securely transmitting information between parties. Because JWTs can be signed – for example, using public/private key pairs – you can be sure the senders are who they say

they are. Additionally, as the signature is calculated using the header and the payload, you can also verify that the content hasn't been tampered with.

In its compact form, JSON Web Tokens consist of three parts separated by dots, which are Header, Payload, Signature. Therefore, a JWT typically looks like

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6I
kpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.
SflKxwRJSMeKKF2QT4fwpMeJf36POk6yJV_adQ
ssw5c
```

Let's break down the different parts.

- *Header.* Typically, consists of two parts: the type of the token, which is JWT, and the signing algorithm being used, such as HMAC SHA256 or RSA. For example,

  ```
  {
      "alg": "HS256",
      "typ": "JWT"
  }
  ```

  Then, this JSON is Base64Url encoded to form the first part of the JWT.

- *Payload.* The second part of the token is the payload, which contains the claims. Claims are statements about an entity (typically, the user) and additional data. There are three types of claims: registered, public, and private claims.

  - `Registered claims.` These are a set of predefined claims which are not mandatory but recommended, to provide a set of useful, interoperable claims. Some of them are: **iss** (issuer), **exp** (expiration time), **sub** (subject), **aud** (audience), and others. Notice that the claim names are only three characters long as JWT is meant to be compact.

  - `Public claims.` These can be defined at will by those using JWTs. But to avoid collisions they should be defined in the IANA JSON Web Token Registry or be defined as a URI that contains a collision resistant namespace.

  - `Private claims.` These are the custom claims created to share information between parties that agree on using them and are neither registered or public claims.

An example payload could be:

```
{
    "sub": "1234567890",
    "name": "John Doe",
    "admin": true
}
```

The payload is then Base64Url encoded to form the second part of the JSON Web Token. Do note that for signed tokens this information, though protected against tampering, is readable by anyone. Do not put secret information in the payload or header elements of a JWT unless it is encrypted.

- *Signature.* To create the signature part you have to take the encoded header, the encoded payload, a secret, the algorithm specified in the header, and sign that. For example if you want to use the HMAC SHA256 algorithm, the signature will be created in the following way:

```
HMACSHA256(
base64UrlEncode(header) + "." +
base64UrlEncode(payload),
secret)
```

The signature is used to verify the message wasn't changed along the way, and, in the case of tokens signed with a private key, it can also verify that the sender of the JWT is who it says it is.

- *Putting all together.* The output is three Base64-URL strings separated by dots that can be easily passed in HTML and HTTP environments, while being more compact when compared to XML-based standards such as SAML. The following shows a JWT that has the previous header and payload encoded, and it is signed with a secret.

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
eyJqdGkiOiJmZDNjNjdjNS1jNmZmLTRhNWQtY
TE2Ni05OGVjZTFiNzc1MmIiLCJyb2xlIjoiVX
NlciIsIm5iZiI6MTYzMTU1MjQ5NiwiZXhwIjo
xNjMxNTUyNzk2LCJpYXQiOjE2MzE1NTI0OTYs
ImlzcyI6Imh0dHBzOi8vbWFuZ28tbWVzc2VuZ
2VyLWFwcC5oZXJva3VhcHAuY29tIiwiYXVkIj
oiaHR0cHM6Ly9tYW5nby1tZXNzZW5nZXItYXB
```

```
wLmhlcm9rdWFwcC5jb20vYXBpIn0.
locHt8ow1lFnGGZ_aFFvXI09dD4y1r594XQF2
-6YxCw
```

As to the projects concerns, we should handle multiple client applications, e.g desktop, web, mobile etc. Therefore, HTTP cookie authorization doesn't fit our requirements, however the JWT one surely passes.

JWT Authorization. In authentication, when the user successfully logs in using their credentials, a JSON Web Token will be returned. Since tokens are credentials, great care must be taken to prevent security issues. In general, you should not keep tokens longer than required. You also should not store sensitive session data in browser storage due to lack of security. Whenever the user wants to access a protected route or resource, the user agent should send the JWT, typically in the Authorization header using the Bearer schema. The content of the header should look like the following:

```
Authorization: Bearer <token>
```

This can be, in certain cases, a stateless authorization mechanism. The server's protected routes will check for a valid JWT in the Authorization header, and if it's present, the user will be allowed to access protected resources. If the JWT contains the necessary data, the need to query the database for certain operations may be reduced, though this may not always be the case. If the token is sent in the Authorization header, Cross-Origin Resource Sharing (CORS) won't be an issue as it doesn't use cookies. Generally, the workflow is as follows

1. User provides credentials in order to authenticate to the system.

2. Server verifies user's authentication, fetches the login and password in database.

3. If authentication is successful, server creates session then writes this session to the database.

4. Server generates a pair of access token (JWT) and refresh token (GUID).

5. Server sends to client access token and refresh token.

6. Client saves the pair of access and refresh tokens.

7. User requests resource using received token passed to the request header.

8. The server check user's claims and proceeds or declines request.

The eighth point is the authorization. As a result, token stored on the client and used when it is necessary to authorize the requests. When a hacker tries to replace the data in the header or payload the token will become invalid, therefore the signature will not match the original values. So, the hacker hasn't any possibility to generate a new signature since that encryption secret key stored on the server. Access token (JWT) is used for request authorization and for storing the additional information about user like identifier, display name and others. Refresh Token (GUID) issued by server based on successful authentication results and used for get new access/refresh token pair. Also, it is worth to add a few basic rules about JWT secure usage [Degges, 2019]

- JWT should have a short lifetime, since it cannot be revoked.

- JWT should be used in a single time, e.g JWT per request.

Therefore, we consider access token's lifetime to be 5 minutes and refresh token's 7 days.

For each request client preliminarily checks access token's lifetime. If access token it expired, client sends request for updating a pair of access/refresh tokens. For more confidence, we can update tokens a few seconds earlier. That is, the case when the API receives an expired access token is practically excluded. However, we are able to consider the case of interception of the request on `401UNAUTHORIZED` http status code. The following diagram demonstrates the process of requesting the resource
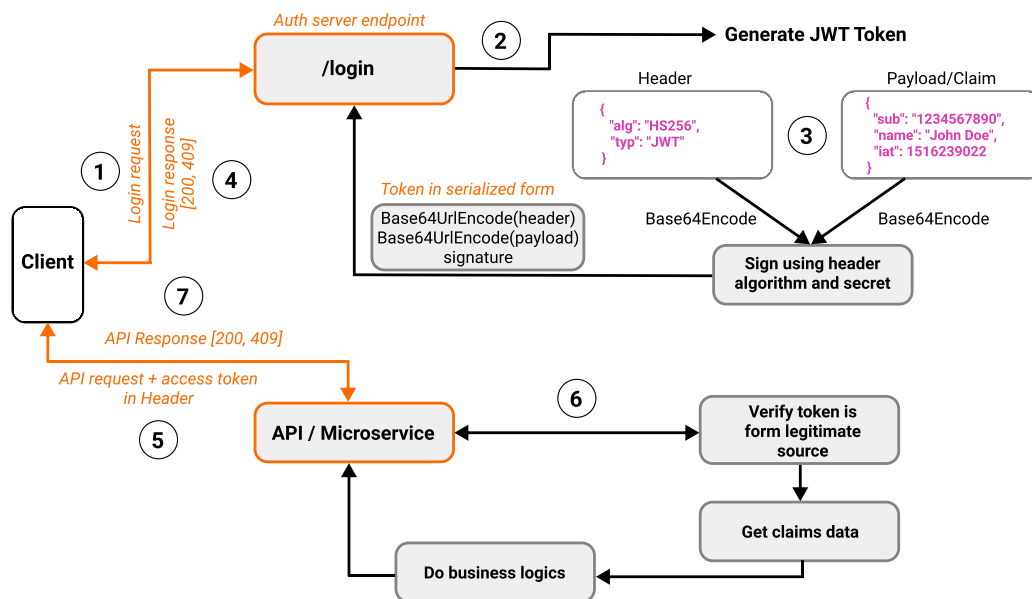

FIGURE 2.6: JWT Authentication concept diagram.

By steps, the process is

- **Step 1.** *Client* application sends POST authentication request to the *Auth server endpoint* provided user credentials in request body.

- **Step 2.** *Auth server endpoint* responses to the *Client* with the following HTTP response codes:

    - `409CONFLICT`: Invalid credentials.
    - `200SUCCESS`: Returns a pair of access and refresh tokens.

        * **Step 3.** Server generates a pair of access and refresh tokens
            · API fetches user data and claims.
            · Server creates new session instance in database.
            · Access token's Header Base64 encoded.
            · Access token's Payload with user claims is Base64 encoded.
            · Access token's Signature is generated using encoded token's Header and Payload signed by means of the algorithm (HMAC-SHA256) from the header and secret:

$$\begin{aligned} \texttt{Signature = HMACSHA256(} \\ \texttt{base64UrlEncode(header) + "."} \\ \texttt{+} \\ \texttt{base64UrlEncode(payload),} \\ \texttt{secret)} \end{aligned}$$

- **Step 4.** Access token in serialized form and refresh token (GUID) returned in response with `200SUCCESS` http status code to the *Client*.

- **Step 5.** *Client* queries the *API / Microservice* providing access token as Bearer in request header.

- **Step 6.** *API / Microservice* validates the token claims in order to authorize user

    - If authorized: *API / Microservice* handles the request, goes to **Step 7**.
    - Otherwise: returns error with `401UNAUTHORIZED` http status code.

- **Step 7.** *API / Microservice* returns response with `200SUCCESS` or `409CONFLICT` http status codes to the client, according to business logic layer implementation.

2.2.7 End-to-end encryption

Motivation. Here is the motivation.

Diffie-Hellman key exchange. Diffie-Hellman protocol is a method of asymmetric exchange of secrets for a group of two or more participants, developed in 1976 by

cryptographers Ralph Merkle and Whitfield Diffie and Martin Hellman. In contrast to symmetric key exchange, the diffie-hellman protocol eliminates the direct transfer of the shared secret. between participants, each participant computes a shared secret with its own private / public key pair. The Diffie Hellman protocol is based on a one-way function of the form

$$A = G^a mod P,$$

where A is the user's public key, a is the user's private key, P is modulus, which is at least 2048 bit prime, G - primitive root modulo P.

Thus, the safety of the Diffie-Hellman protocol is based on the discrete logarithm problem, which is unsolvable in polynomial time if the constants G and P are chosen correctly. Graphically, the essence of the Diffie Hellman protocol can be expressed through the analogy with mixing paints, as shown in image below
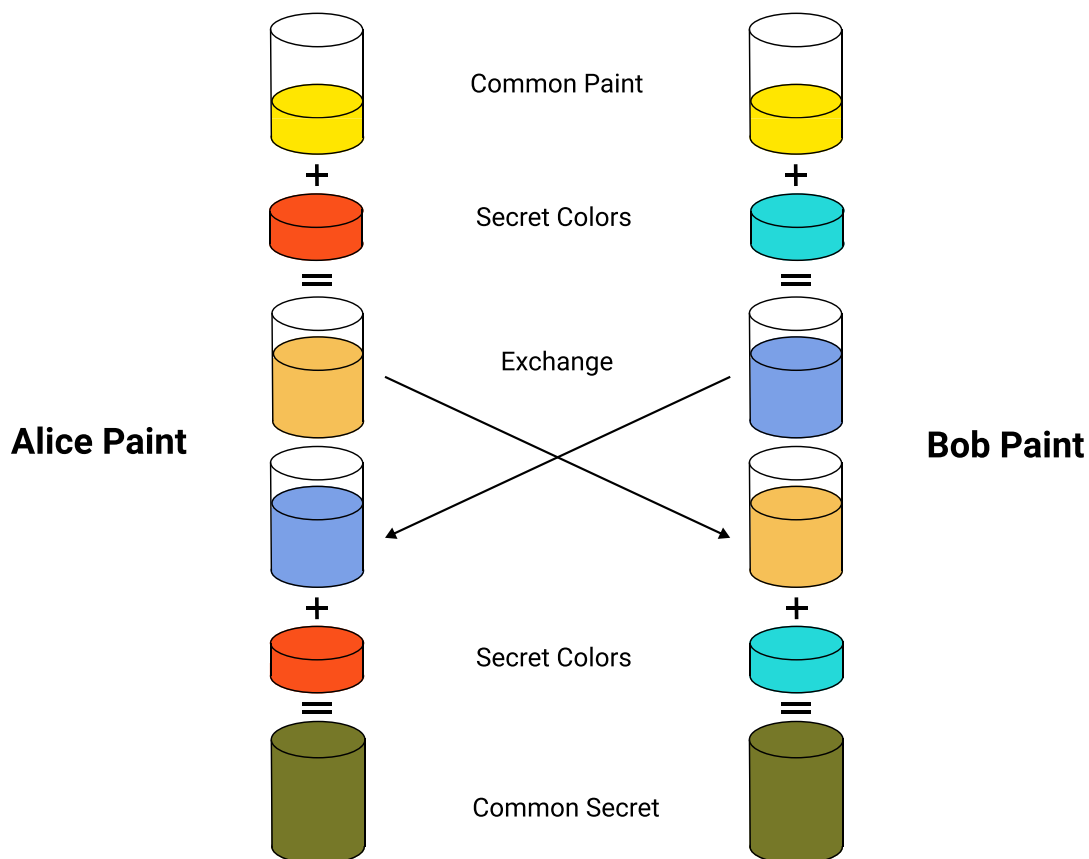
FIGURE 2.7: Illustration of the concept behind Diffie–Hellman key exchange. Source:

In contrast to the diffie-hellman based on discrete logarithm problem, there is an elliptic curve diffie hellman key exchange, which based on the elliptic curve discrete

logarithm problem. Although, the idea is quite same, but elliptic curve diffie hellman ensures the same safety as discrete logarithm diffie-hellman with lesser value of the modulus P. For instance, 521 bit modulus used in elliptic curve diffie hellman is equally safe as 2048 bit modulus in discrete logarithm diffie-hellman. To summarize, the flow of diffie hellman kay exchange is as follows

1. Given 2048 bits prime modulus P and generator G, such that G is primitive root modulo P.

2. Alice chooses her secret a.

3. Alice sends to Bob her public key $A = G^a \bmod P$.

4. Bob chooses his secret b.

5. Bob sends to Alice his public key $B = G^b \bmod P$.

6. Alice computes common secret $s = B^a \bmod P$.

7. Bob computes common secret $s = A^b \bmod P$.

8. Alice and Bob have arrived to the same value

$$A^b \bmod P = G^{ab} \bmod P$$
$$B^a \bmod P = G^{ba} \bmod P$$

Implementation. Although, the idea of DH key exchange looks quite simple, some remarks on the concrete implementation should be added. Firstly, it is necessary to implement the mechanism of key exchange request between two or more parties. As it discussed above, each user has his own private / public keys pair, so in order to perform request between parties, it should be implemented dedicate REST API endpoint, for instance the POST one `api/key-exchange-requests` which takes the body of the form

```
{
    "requestedUserId": "3fa85f64-5717-4562-b3fc-2c963f66afa6",
    "publicKey": "RUNLMSAAAAC2lkqYcTGhutQPxcjvoqUELKoy0"
}
```

So, request sender generates on the client side a key pair, keeps private on in the file system and shares the public in request to receiver. Therefore, the second party has received the key exchange request. In order to display all the key exchange requests awaiting the confirmation of decline decisions, it is worth to implement another REST endpoint such that GET `api/key-exchange-requests`, so that requested party will have the list of requests to proceed. This endpoint may return the data structure like follows

```
{
    "keyExchangeRequests": [
        {
            "requestId": "81d314c1-913f-4686-827e-ef2a65ccc370",
            "senderId": "3fa85f64-5717-4562-b3fc-2c963f66afa6",
            "senderPublicKey": "RUNLMSAAAAC2lkqYcTGhutQPxcjvoqUELKoy0"
        }
    ],
    "message": "SUCCESS",
    "success": true
}
```

Finally, requested party should be able to confirm or decline the key exchange request, the DELETE endpoint `api/key-exchange-requests` should be implemented then. The server is able to fetch the request thanks to the body endpoint takes

```
{
    "requestId": "3fa85f64-5717-4562-b3fc-2c963f66afa6",
    "confirmed": true,
    "publicKey": "string"
}
```

Therefore, an identifier of awaiting request is passed to the server among with boolean value indicating the confirmation. Under the roof of this operation are also generation of private / public keys pair for the requested party and generation of common secret stored in client's file system. As result, the initial request sender receives a public key as confirmation from requested party. Requested side may get all his public keys via the REST API using GET `api/public-keys`

```
{
    "publicKeys": [
        {
            "partnerId": "ae9e10a4-0c7e-4911-8450-4139d4a114a7",
            "partnerPublicKey": "RUNLMSAAAAAbc49wfaZ+QF9J2cu1S66bkp0"
        }
    ],
    "message": "SUCCESS",
    "success": true
}
```

Now requested participant is able to derive the common secret. In order to provide an example, a simple command line interface is implemented. We have used an Elliptic Curve Diffie-Hellman implementation `ECDiffieHellmanCng Class` from the namespace `System.Security.Cryptography` of the .NET base class library. The `P-256` curve is used.

More precisely, the following CLI commands are implemented

- `MangoAPI.DiffieHellmanConsole login SENDER_EMAIL SENDER_PASSWORD`

- `MangoAPI.DiffieHellmanConsole key-exchange RECEIVER_ID`

- `MangoAPI.DiffieHellmanConsole key-exchange-requests`

- `MangoAPI.DiffieHellmanConsole confirm-key-exchange REQUEST_ID`

- `MangoAPI.DiffieHellmanConsole print-public-keys`

- `MangoAPI.DiffieHellmanConsole create-common-secret RECEIVER_ID`

Commands are self-explanatory, therefore we skip the detailed documentation on them. An example of console output straightforward

FIGURE 2.8: Secret chat encryption concept diagram. Source:

Finally, both test accounts reached the same common secret. In order to work with our implementation of the ECDH key exchange, the user is obligated to register an account at https://messenger-mango.herokuapp.com.

2.3 Project outcomes

Please describe the achieved outcomes of the project. If possible, please provide figures showing the described outcomes. Please confront them with the objectives of the project. This part should be between 2000 and 10,000 characters long. Please use Times New Roman font, 12 pts, 1.5 spacing. Full description of solutions that were worked out and project outcomes, if any, should be presented in annexes.
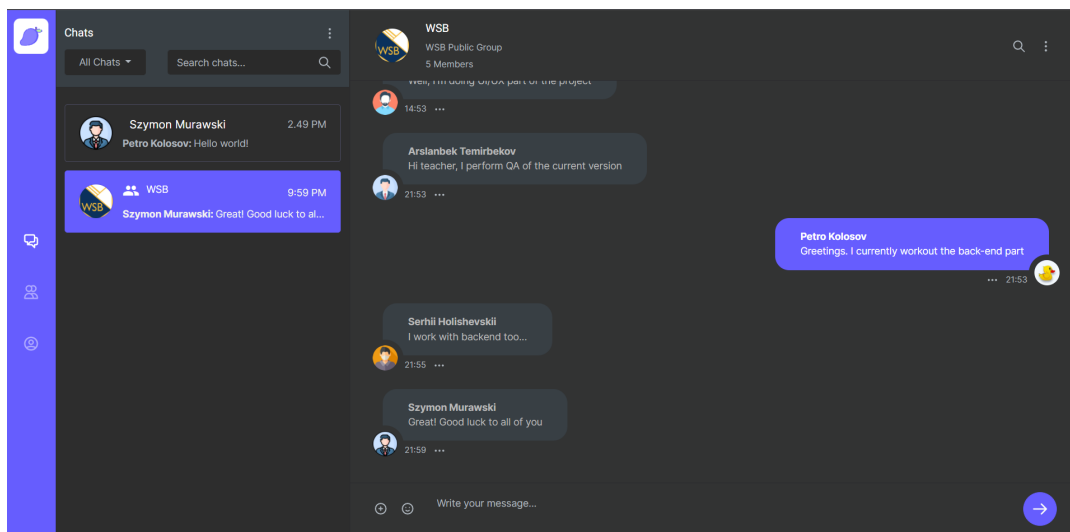

FIGURE 2.9: Secret chat encryption concept diagram. Source:

1. In the screenshot 2.9, on the left, there is a list of the user's chats.

2. In the screenshot 2.9, on the top left, there is a field for text input, entering a certain text, and clicking on the enter or on the icon with a magnifying glass the user can find the chat he is interested in, containing the entered text in its name.

3. In the screenshot 2.9, on the top left, there is an icon with three dots, by clicking on which a button will appear to create a chat, by clicking on it, you can go to the creation of a chat.

4. In the screenshot 2.9, on the top left there is a button, by clicking on which a drop-down list appears, by clicking on a specific one, you can choose to filter the displayed chats.

5. In the screenshot 2.9, on the right, there is a magnifying glass icon, by clicking on which you can search by messages in a particular chat.

6. In the screenshot 2.9, on the right, there is an icon with three dots, by clicking on which 2 buttons will appear, by clicking on the first user can (un)archive the chat, by clicking on the second user can leave the chat.

7. In the screenshot 2.9, at the bottom there is a field for entering text, entering text into it, and clicking on the enter or on the arrow icon at the bottom right,the user can send a message to a specific chat.

8. In the screenshot 2.9, below, to the left of the text message input field, there is an icon with a smiley by clicking on it, and by clicking on a specific emoji, the user can add it to his text message.

9. In the screenshot 2.9, below, to the left of the emoticon icon, there is an icon with a plus by clicking on it, then selecting the file, the user can send the selected file to a specific chat by clicking on the send button.
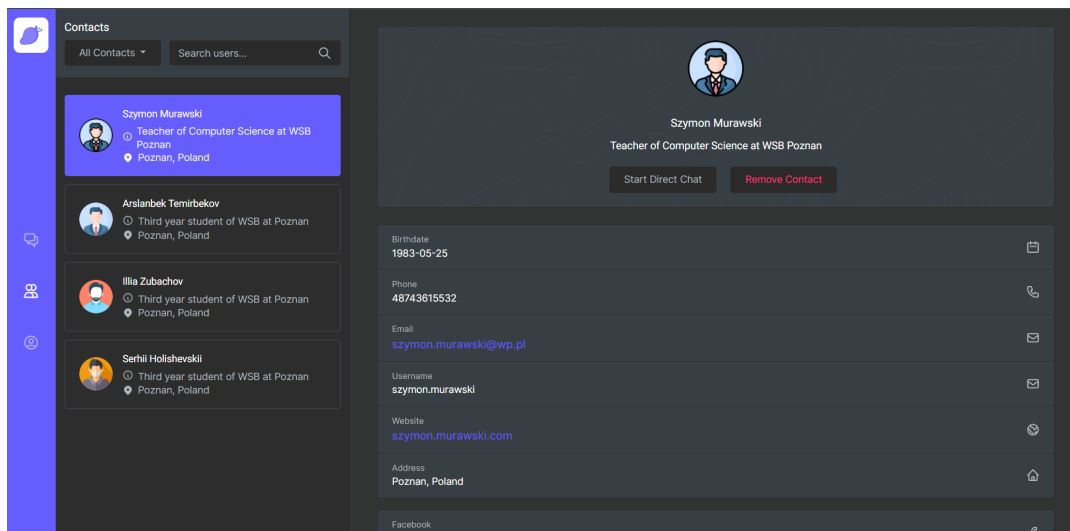

FIGURE 2.10: Secret chat encryption concept diagram. Source:

1. In the screenshot 2.10, on the left there is a list of user contacts, by clicking on a specific one, information about the contact will be displayed on the right and two buttons, the first allows you to start a chat with a specific user, and the second button removes the user from the contact list.

2. In the screenshot 2.10, on the left, there is a field for entering text, entering certain text, and clicking on the enter or on the magnifying glass, will display users who have a phone number/email/display name contains the entered text, by selecting one of the displayed, you can add it to your contacts, start with a chat and get information about it.

3. In the screenshot 2.10, on the left, there is a button for filtering contacts, by clicking on it, a drop-down list appears, in which you can select a filtering method: "All contacts", "Search results".
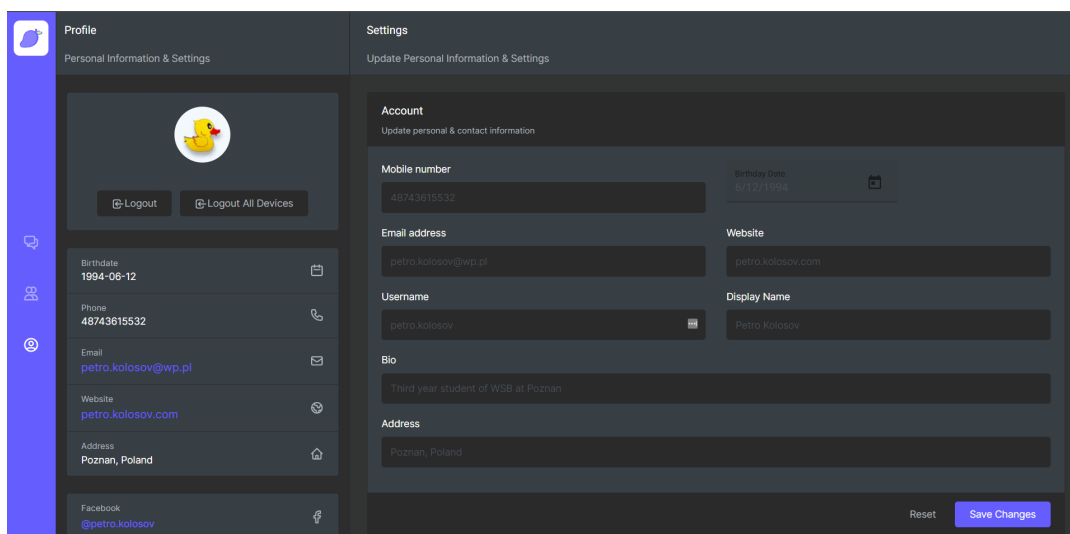

FIGURE 2.11: Secret chat encryption concept diagram. Source:

1. In the screenshot 2.11, on the left, there is the user's current information, his avatar, and two button buttons, the first one logs out of the account on this device, and the second button logs out the account from all devices.

2. In the screenshot 2.11, in the center, displayed the user settings, there are fields for updating personal data, updates of links to social networks.

2.4 Usefulness of project

The project may be used as corporate messenger in closed scope of some company under VPN as cheaper alternative to the nowadays popular Microsoft Teams. The system provides independence from the centralized Microsoft's products. Providing a various types of subscriptions and support levels current project may be considered as corporative or educational solution.

2.5 Project self-evaluation

Each of the project's Authors describes his or her skills and competencies that were developed while working on the project and identifies issues encountered while working on the project. If during the work on the project the team had not completed

any tasks planned earlier, or omitted them altogether, please specify what were these tasks and why they had not been completed. This part should not exceed 6000 characters. Please use Times New Roman font, 12 pts, 1.5 spacing.

- **Petro Kolosov.** Description of contributions.

- **Serhii Holishevskii.** Description of contributions.

- **Illia Zubachov.** Description of contributions.

- **Arslanbek Temirbekov.** Description of contributions.

Bibliography

Alwin, Duane F and Brett A Beattie (2016). "The KISS principle in survey design: question length and data quality". In: *Sociological methodology* 46.1, pp. 121–152.

Brataas, Gunnar and Peter Hughes (2004). "Exploring architectural scalability". In: *Proceedings of the 4th international workshop on Software and performance*, pp. 125–129.

Bucchiarone, Antonio et al. (2018). "From monolithic to microservices: An experience report from the banking domain". In: *Ieee Software* 35.3, pp. 50–55.

Burrows, Michael, Martin Abadi, and Roger Michael Needham (1989). "A logic of authentication". In: *Proceedings of the Royal Society of London. A. Mathematical and Physical Sciences* 426.1871, pp. 233–271.

Chung, Lawrence et al. (2012). *Non-functional requirements in software engineering*. Vol. 5. Springer Science & Business Media.

Cocks, Clifford C (1973). "A note on non-secret encryption". In: *CESG Memo*.

Cohn, Mike (2004). *User stories applied: For agile software development*. Addison-Wesley Professional.

Da Silva, Tiago Silva et al. (2018). "The evolution of agile UXD". In: *Information and Software Technology* 102, pp. 1–5.

Degges, Randall (2019). *JWTs Suck*. https://speakerdeck.com/rdegges/jwts-suck. [Online; accessed 15-August-2021].

Dilworth, John and AK Kochhar (2007). "Creation of an e-business requirements specification model". In: *Journal of Manufacturing Technology Management*.

Fagin, Ronald (1978). "On an authorization mechanism". In: *ACM Transactions on Database Systems (TODS)* 3.3, pp. 310–319.

Hindocha, Neal and Eric Chien (2003). "Malicious threats and vulnerabilities in instant messaging". In: *Virus Bulletin Conference, vb2003*.

ISO, BSEN and BRITISH STANDARD (2010). "Ergonomics of human-system interaction". In.

Johnson, Don, Alfred Menezes, and Scott Vanstone (2001). "The elliptic curve digital signature algorithm (ECDSA)". In: *International journal of information security* 1.1, pp. 36–63.

Jones, M, J Bradley, and N Sakimura (2015). "Rfc 7519: Json web token (jwt)". In: *IETF. May*.

Bibliography

Jones, Michael, Brain Campbell, and Chuck Mortimore (2015). "JSON Web Token (JWT) profile for OAuth 2.0 client authentication and authorization Grants". In: *May-2015.{Online}. Available: https://tools. ietf. org/html/rfc7523*.

Malan, Ruth, Dana Bredemeyer, et al. (2001). "Functional requirements and use cases". In: *Bredemeyer Consulting*.

Mannan, Mohammad and Paul C van Oorschot (2004). "Secure public instant messaging: A survey". In: *Proceedings of Privacy, Security and Trust*, p. 95.

Mannan, Mohammad Abdul (2005). "Secure public instant messaging". PhD thesis. Carleton University.

Proctor, Frederick M (1999). "Linux, Real-Time Linux, & IPC." In: *Dr. Dobb's Journal: Software Tools for the Professional Programmer* 24.11, pp. 32–36.

Rasche, Peter et al. (2016). "Building and Exchanging Competence Interdisciplinarily: Design Patterns as Domain Mediator". In: *Proceedings of the International Symposium on Human Factors and Ergonomics in Health Care*. Vol. 5. 1. SAGE Publications Sage CA: Los Angeles, CA, pp. 19–24.

Rising, Linda (1998). "Design patterns: Elements of reusable architectures". In: *The Patterns Handbook: Techniques, Strategies and Applications*, pp. 9–13.

Rivest, Ronald L, Adi Shamir, and Leonard Adleman (1978). "A method for obtaining digital signatures and public-key cryptosystems". In: *Communications of the ACM* 21.2, pp. 120–126.

Wang, Mao-Yin et al. (2004). "An HMAC processor with integrated SHA-1 and MD5 algorithms". In: *ASP-DAC 2004: Asia and South Pacific Design Automation Conference 2004 (IEEE Cat. No. 04EX753)*. IEEE, pp. 456–458.

Wiener, Michael J (1990). "Cryptanalysis of short RSA secret exponents". In: *IEEE Transactions on Information theory* 36.3, pp. 553–558.

Young, Greg (2010). "CQRS and Event Sourcing". In: *URL: http://codebetter. com/gregyoung/2010/02/13/cqrs-and-event-sourcing*.

Appendix A

List of functional requirements

Registration user stories

- As an unregistered user, I want to tap "Register" so that I see the registration form and register myself.

- As an unregistered user, I want to use my phone number to register so that my account is tied to my phone number.

- As an unregistered user, I want to use my e-mail address to register so that my account is tied to my e-mail address.

- As an unregistered user, I want to add a display name during registration so that other users can find me using it.

- As an unregistered user, I want to choose how to receive the registration confirmation via SMS or e-mail so that notification is sent to me via SMS or e-email.

- As an unregistered user, I want to receive the registration confirmation via SMS or Email so that I can activate my account.

- As a registered user, I want to confirm my email address so that I get confirmation link via email I provided.

- As a registered user, I want to confirm my phone number so that I use specified form to do it.

Authentication user stories

- As a registered user, I want to authenticate myself using both combinations email-password and phone-password so that I use the specified form with two inputs.

- As a registered user, I want to restore my password if I forget it so that I use specified form and restore my password.

- As an authenticated user, I want my session on each device to least 7 days so that after 7 days of inactivity device will be logged out automatically.

Managing contacts user stories

- As an authorized user, I want to see my contact list so that there is a list of users who are my contacts.

- As an authorized user, I want to search users so that I write user display name or phone number of e-mail address to specified input, click "Search user" button and see results.

- As an authorized user, I want to add other user to my contact list so that I click "Add contact" button on user profile and add him to my contact list.

- As an authorized user, I want user search input to accept empty or whitespace queries so that all users displayed as search result.

- As an authorized user, I want to remove the user from my contact list so that I click "Remove contact" button on user profile and remove him from my contact list.

- As an authorized user, I want to navigate to private chat with the user from my contact list so that I click "Message" button at user profile and get navigated to the private chat with him.

Sending messages and media to individuals user stories

- As an authorized user, I want to send a text message so that another user sees my message.

- As an authorized user, I want to add an attachment to the message so that another user sees the message with attachment.

- As an authorized user, I want to add an emoji to the message so that another user sees the message with emoji.

- As an authorized user, I want to tap "Edit" on my message so that message I edited is changed immediately in the chat.

- As an authorized user, I want to tap "Delete" on my message so that message immediately disappears from the chat.

- As an authorized user, I want to share secret messages with users from my contact list so that our are messages encrypted for anyone else including system administrators.

- As an authorized user, I want each new message in private chats I participate to be displayed immediately in real-time so that I do not reload page.

Creating and managing groups user stories

- As a registered user, I want to tap "Create channel" so that I create a new channel of the one of the types: Private channel, Public channel, Readonly channel.

- As a registered user, I want to tap "Start direct chat" so that I create a new direct chat with specified user.

- As a registered user, I want to tap "Start secret chat" so that I create a new secret chat with specified user.

- As a registered user, I want to join public groups so that I click button "Join group" to join the group.

- As a registered user, I want to tap "Archive" so that I archive the specified chat or channel.

- As a registered user, I want to tap "Un-archive" so that I un-archive the specified archived chat or channel.

- As a registered user, I want my secret chats to be device-specific so that I can see a secret chat only on the device that I used to start this chat.

Sending messages and media to groups user stories

- As an authorized user, I want to send a text message so that other members of the group see the message I sent.

- As an authorized user, I want to add an attachment to the message so that other members of the group see the message with attachment I sent.

- As an authorized user, I want to add an emoji to the message so that other members of the group see the message with emoji I sent.

- As an authorized user, I want to tap "Edit" on my message so that other members of the group see the message I edited.

- As an authorized user, I want to tap "Delete" on my message so that my message is deleted for all members of the group.

- As an authorized user, I want to search public groups by title so that I enter display name to specified field, click button "Search chats" and see results.

- As an authorized user, I want each new message in groups I participate to be displayed immediately in real-time so that I do not reload page.

Viewing messages history user stories

- As an authorized user, I want to view a message history of particular chat or group so that I see a list of my active chats on the UI.

- As an authorized user, I want to search messages in particular chat so that I see the results in messages window of the chat.

Managing profile settings user stories

- As an authorized user, I want to update my personal information in profile settings so that other users my updated personal information.

- As an authorized user, I want to update my social network links in profile settings so that other users my updated social media.

- As an authorized user, I want to change my profile picture so that all other users will see updated one.

- As an authorized user, I want reset password, so that my password will change.

- As an authorized user, I want to tap "Logout" button so that current device will be logged out from the system.

- As an authorized user, I want to tap "Logout all" button so that all my authorized devices will be logged out from the system.

Appendix B

List of non-functional requirements

- **NFR01.** Graphic user interface of the system should be well organized. To fulfill this requirement, we follow an ISO 9241–161:2010 (en) Ergonomics of human-system interaction standard [ISO and STANDARD, 2010].

- **NFR02.** The system should have well performance, which meant to respond it at least 1 second. User should have a device with at least 6 GB RAM and CPU with 1.8 GHZ, 100 Mbps internet connection. Server must have the following hardware: Intel 2.4 GHz 8 Cores server processor, 64GB DDR4 (4x16GB) memory, NVME or SAS server disk with a minimum capacity of 1.6 TB.

- **NFR03.** The unique, unambiguous identifier of users in the system is the username. It is set in the profile settings.

- **NFR04.** The UI must be well displayed with the following browsers, in the versions current at the date of receipt of the system or, depending on technical possibilities, with the latest versions that support correct operation of the system:

  - Google Chrome 72.0.36.
  - Mozilla Firefox 64.0.2.
  - Microsoft Edge 17.17134.

- **NFR05.** The system shall force users to use passwords with a minimum length of 8 characters and using at least one capital letter and one number and one special symbol.

- **NFR06.** The UI must be compatible to use on mobile device screens with a minimum width of 600 pixels.

- **NFR07.** The UI must be compatible to use on desktop or laptop device screens with a minimum display width of 1024 pixels.

Appendix C

RSA Algorithm comments

C.1 Details

C.1.1 One way functions

One way function – is a function that is easy to compute on every input, but hard to invert given the image of a random input. For instance, the function

$$f(m) = m^e \bmod N \equiv C$$

where e, N are public constants is one-away function, because it is easy to compute C given m, however it is hard to compute m given C.

C.1.2 Euler's totient theorem

Given a number N and its prime factorization $p_1^{e_1} \cdot p_2^{e_2} \cdots p_k^{e_k}$, then Euler's totient function $\phi(N)$ is defined as

$$\phi(N) = (p_1^{e_1} - p_1^{e_1-1}) \cdot (p_2^{e_2} - p_2^{e_2-1}) \cdots (p_k^{e_k} - p_k^{e_k-1})$$

In particular, for positive number M such that its factorization is $p1 \cdot p2$, the $\phi(M)$ is

$$\phi(M) = (p_1 - 1) \cdot (p_2 - 1)$$

Euler's theorem relates the modular division and exponent as follows, given number m, then

$$m^{\phi(N)} = 1 \bmod N$$

It means that reminder of division $m^{\phi(N)}$ by N is always 1. By the equality $1^K = 1$

$$M^{K \cdot \phi(N)} = 1 \bmod N$$

If we multiply both parts by M, we get

$$M \cdot M^{K \cdot \phi(N)} = M^{K \cdot \phi(N)+1} = M \bmod N$$

C.2 RSA Encryption algorithm

The RSA algorithm is named after Ron Rivest, Adi Shamir and Len Adleman, who invented it in 1977 [Rivest, Shamir, and Adleman, 1978]. The basic technique was first discovered in 1973 by Clifford Cocks [Cocks, 1973] of CESG (part of the British GCHQ) but this was a secret until 1997. The patent taken out by RSA Labs has expired.

Historically, the process of encryption is considered to be symmetric one. That means that prior the communication, the sides conclude on the common key to be used in encryption. This process is similar to the first sharing keys and only after that the locked chest with the message. Such approach is highly cost since it requires to share the defined keys between each actor if the number of actors is greater than 2. Much more simpler is to think about secured communication channel that in terms of asymmetric encryption. The real life example would be if Alice shares with all actors an opened lock having key. So that Bob receives an opened lock, writes letter to Alice, puts letter to the chest, locks this chest with received from Alice lock. This way, only Alice will be able to open the chest and to read the letter. This is an idea of the asymmetric encryption. However, such a simple from first glance idea requires complex number theory approach. A concept of opened lock may be interpreted in terms of one-way functions. One way function – is a function that is easy to compute on every input, but hard to invert given the image of a random input. Thus, it is much simpler to close the lock without key, but very difficult to open lock trying the combinations of the key. For instance, the function

$$f(m) = m^e \bmod N = C$$

where e, N are public constants is one-away function, because it is easy to compute C given m, however it is hard to compute m given C. So, assume that Alice defines two positive integer constants e, N and sends it to Bob. Bob encrypts the secret message m using $f(m)$

$$f(m) = m^e \bmod N = C$$

Then Bob sends encrypted message C to the Alice. Given C Alice must fetch the Bob's message m. In order to decrypt C, Alice has to compute

$$C^d \bmod N = m^{ed} \bmod N \equiv m,$$

where e for encryption and d for decryption. Now the problem is to define such d that it is hard to the listener to fetch it. In order to define the secret d, Alice chooses two enough big prime numbers: P, Q, let's say around 150 digits both. Then Alice multiplies these two prime numbers in order to get N

$$N = P \cdot Q$$

The N is around 300 digits. Now Alice can share N with anyone, since it takes decades to find its prime factorization by the fundamental problem of prime factorization. Next, it is very important to know such a function, which depends on the knowledge of factorization of N. Such function is an Euler's totient function. Given a number N and its prime factorization $p_1^{e_1} \cdot p_2^{e_2} \cdots p_k^{e_k}$, the Euler's totient function $\phi(N)$ is defined as

$$\phi(N) = (p_1^{e_1} - p_1^{e_1-1}) \cdot (p_2^{e_2} - p_2^{e_2-1}) \cdots (p_k^{e_k} - p_k^{e_k-1})$$

In particular, for positive number M such that its factorization is $p1 \cdot p2$, the $\phi(M)$ is

$$\phi(M) = (p_1 - 1) \cdot (p_2 - 1)$$

Euler's theorem relates the modular division and exponent as follows, given number m, then

$$m^{\phi(N)} = 1 \bmod N$$

It means that reminder of division $m^{\phi(N)}$ by N is always 1. By the equality $1^K = 1$

$$M^{K \cdot \phi(N)} = 1 \bmod N$$

If we multiply both parts by M, we get

$$M \cdot M^{K \cdot \phi(N)} = M^{K \cdot \phi(N)+1} = M \bmod N$$

It follows that Alice is able to define the secret d as follows

$$e \cdot d = K \cdot \phi(N) + 1$$
$$d = \frac{K \cdot \phi(N) + 1}{e}$$

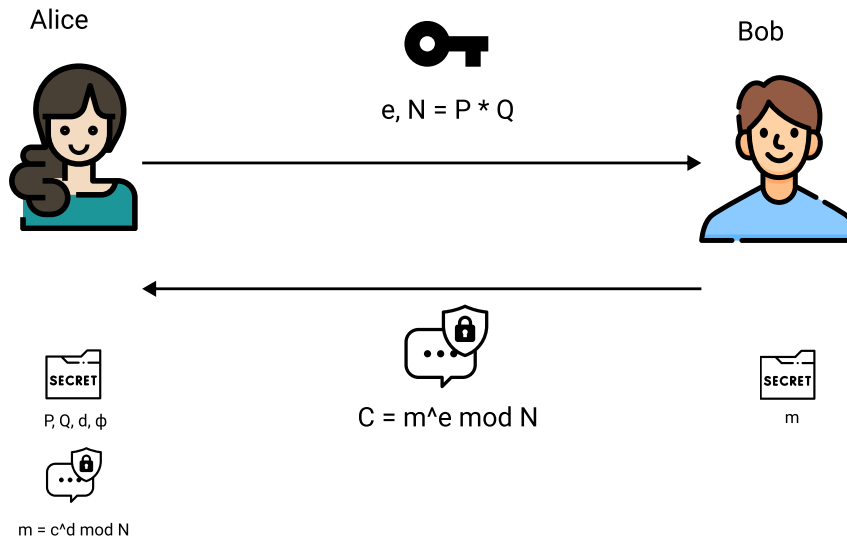The following image demonstrates the concept of RSA approach

FIGURE C.1: Secret chat encryption concept diagram. Source:

To summarize, the process by the steps is as follows

- Alice defines the large secret prime numbers P, Q.

- Alice computes $N = P \cdot Q$ and $\phi = (P-1)(Q-1)$

- Alice chooses an integer e, $1 < e < \phi$ such that $\gcd(e, \phi) = 1$.

- Alice computes secret exponent d, $1 < d < \phi$ such that $ed \equiv 1 \bmod \phi$.

- Alice shares public key (N, e) with Bob and keeps private key (d, p, q) is secret.

- Bob defines the message m, encrypts it as $C = m^e \bmod N$.

- Bob sends C to Alice.

- Alice decrypts C using her secret d, so she gets m

$$m = C^d \bmod N$$

Security of the RSA approach is based on the complexity of fundamental problem of prime factorization, which takes decades to solve having enough large number.