



CS 411 SOFTWARE ARCHITECTURE PROJECT #2

Ayberk Yaşa	21801847, Section 1
Cemhan Kaan Özaltan	21902695, Section 1
Fatih Kaplama	21802755, Section 1
Görkem Ayten	21802399, Section 1

1. Introduction	4
2. Analysis of Instant Messaging Platforms	4
2.1. Problem Definition and Domain	4
2.2. Major Technical Problems and Proposed Solutions	4
2.2.1. Security and Privacy	4
2.2.2. Scalability	4
2.2.3. Interoperability	5
2.2.4. User Experience	5
2.2.5. Reliability	5
2.3. Stakeholders	5
2.3.1. Customers	5
2.3.2. Project Development Team	5
2.3.2.1. Project Managers	5
2.3.2.2. Business Analysts	5
2.3.2.3. Developers and Engineers	5
2.3.2.4. UI/UX Designers	6
2.3.3. Investors	6
2.3.4. Governments	6
2.3.5. Competitors	6
2.3.6. Companies	6
2.4. Pros and Cons of Existing IM Platforms	6
2.4.1. Meta WhatsApp	6
2.4.2 Telegram	7
2.4.3 Viber	7
2.4.4 Signal	8
3. Proposed Instant Messaging Product	8
3.1. Quality Requirements	9
3.1.1. Security	9
3.1.2. Data Privacy	9
3.1.3. High Performance	9
3.1.4. Reliability	10
3.2. Main Requirements	10
3.2.1. Real-Time Text Transmission	10
3.2.2. Emojis	10
3.2.3. Voice over IP	10
3.2.4. Video Chat	11
3.3. High Level Architecture	11
3.3.1. Logical View	12
3.3.1.1. Generalization Module Style	14

3.3.2. Process View	14
3.3.2.1. Publish-Subscribe C&C Style	15
3.3.3. Development (Implementation) View	16
3.3.3.3. Layered Module Style	17
3.3.4. Physical (Deployment) View	17
3.3.4.1. Deployment Allocation Style	17
3.3.5. Scenarios (Use Case) View	18
3.4. Architectural Patterns	18
3.4.1. Publish-Subscribe Pattern	18
3.4.2. Event-Bus Pattern	19
3.4.3. Model-View-Controller Pattern	19
3.5. Final Product	20
3.5.1. Sign In Screen	20
3.5.2. Sign Up Screen	21
3.5.3. Messaging Screen	21
References	22

1. Introduction

In this report, an analysis of existing instant messaging applications which already exist in the market, followed by an instant messenger web app proposal called WeText. The high level architecture of this system will be described, in addition to the used architectural patterns.

2. Analysis of Instant Messaging Platforms

2.1. Problem Definition and Domain

Instant messaging (IM) is a type of online communication offering real-time text transmissions over the internet. Users are allowed to send and receive messages without having to refresh the page or wait for a response by using this technology. IM applications solve the problem of delay in communication commonly associated with asynchronous online communication like email. This involves designing and developing software that enables text-based communication with user-friendly interfaces and features such as sending and receiving files, sharing videos and images, and engaging in group conversations. It allows people to have real-time conversations and get immediate feedback that can be useful in personal and business life where quick communication is necessary. The domain of IM applications includes computer science, software engineering, human-computer interaction, and communication studies.

2.2. Major Technical Problems and Proposed Solutions

While the technical problems of the instant messaging application may vary depending on its intended use, the major problems can be listed as follows:

2.2.1. Security and Privacy

Ensuring that user data is secure and protected from unauthorized access and that users have control over who can see their messages and information. Proposed solutions might include encryption, authentication, and access control mechanisms.

2.2.2. Scalability

Ensuring that the instant messaging application can handle large volumes of traffic and users without crashing or experiencing significant delays. Proposed solutions might include distributed systems, load balancing, and efficient data storage and retrieval techniques.

2.2.3. Interoperability

Ensuring that the instant messaging application can work with other applications and systems, and that users can communicate with people using different technologies. Proposed solutions might include open standards and protocols, as well as APIs and other integration tools.

2.2.4. User Experience

Ensuring that the instant messaging application is easy to use and understand, and that it provides a seamless and intuitive experience for users. Proposed solutions might include user research, usability testing, and user-centered design.

2.2.5. Reliability

Ensuring that the instant messaging application is available and functioning at all times and that users can rely on it for communication. Proposed solutions might include fault tolerance, redundant systems, and regular maintenance and updates.

2.3. Stakeholders

2.3.1. Customers

The individuals who use instant messaging applications to communicate with other people.

2.3.2. Project Development Team

2.3.2.1. Project Managers

Project managers manage the entire project development process, taking into account the interests and needs of all stakeholders. Their primary goal is to deliver a high-quality product on time and within budget, leaving customers satisfied. They lead a development team and oversee project implementation processes, making adjustments as needed.

2.3.2.2. Business Analysts

Business analysts analyze the customers' ideas, communicate with the developers, and determine the scope and requirements of the project. They are responsible for predicting the project budget and time.

2.3.2.3. Developers and Engineers

The frontend and backend developers who are responsible for implementing the core functionalities and logic of the apps in addition to devops engineers, who are

responsible for deploying the end product to the appropriate environments, all hold a stake in this domain.

2.3.2.4. UI/UX Designers

Designers who aim to make more usable and well produced messengers are also involved in this domain.

2.3.3. Investors

The individuals or organizations that provide financial support for the development and growth of instant messaging applications.

2.3.4. Governments

The regulatory bodies that oversee the use of instant messaging technology and ensure it is used in accordance with laws and regulations.

2.3.5. Competitors

Other companies or organizations that offer similar technologies or services in the same market.

2.3.6. Companies

Other companies or organizations that work with the instant messaging application to offer complementary services or products.

2.4. Pros and Cons of Existing IM Platforms

2.4.1. Meta WhatsApp

Pros:

- It is free to use, with no costs for sending messages or making calls.
- It has a user-friendly interface and is easy to set up and use.
- It is widely used, with over 2 billion users worldwide.
- It offers end-to-end encryption for enhanced security and privacy.
- Share end-to-end encrypted live location
- Audio and video call
- Support for Siri
- Sending a message to multiple contacts at once using the broadcast list.
- Deleting the mistakenly sent message
- When the disappearing messages feature is activated, messages will disappear after 7 days.
- Muting notifications for chat
- Watching YouTube videos in chat.
- Hands-Free voice note recording.

- WhatsApp Business

Cons:

- It is owned by Facebook, which has been criticized for its handling of user data and privacy.
- It has faced controversy over its privacy policies and data-sharing practices.
- It is not available in some countries due to government restrictions or censorship.
- It has had security issues and vulnerabilities in the past, which may make some users concerned about using it.
- It has limited features compared to some other instant messaging applications, such as Telegram or Signal.
- You cannot contact a person on WhatsApp if you don't have their cell phone number.
- File size limitation. You cannot share a media file bigger than 16MB or file bigger than 100MB.
- No virtual assistant to guide users.

2.4.2 Telegram

Pros:

- It is free to use, with no costs for sending messages or making calls.
- It offers end-to-end encryption for enhanced security and privacy.
- It has a user-friendly interface and is easy to set up and use.
- It allows users to send and receive messages, make calls, share videos and photos, and engage in group conversations.
- It has additional features, such as the ability to create secret chats and self-destructing messages.

Cons:

- It is not as widely used as some other instant messaging applications, such as WhatsApp or Facebook Messenger.
- It is not available in some countries due to government restrictions or censorship.
- It has faced criticism over its security and encryption protocols.
- It has had issues with spam and bots, which can be annoying for users.
- It has faced controversy over its association with the Russian government and its founders.

2.4.3 Viber

Pros:

- It is free to use, with no costs for sending messages or making calls.
- It has a user-friendly interface and is easy to set up and use.

- It allows users to send and receive messages, make calls, share videos and photos, and engage in group conversations.
- It offers end-to-end encryption for enhanced security and privacy.
- It has additional features, such as the ability to share location and create custom stickers.

Cons:

- It is not as widely used as some other instant messaging applications, such as WhatsApp or Facebook Messenger.
- It is not available in some countries due to government restrictions or censorship.
- It has faced criticism over its security and encryption protocols.
- It has had issues with spam and bots, which can be annoying for users.
- It has been acquired by Rakuten, a Japanese e-commerce company, which has raised concerns about data privacy and user control.

2.4.4 Signal

Pros:

- It is free to use, with no costs for sending messages or making calls.
- It offers end-to-end encryption for enhanced security and privacy.
- It has a user-friendly interface and is easy to set up and use.
- It allows users to send and receive messages, make calls, share videos and photos, and engage in group conversations.
- It has additional features, such as the ability to set expiration times on messages and use disappearing messages.

Cons:

- It is not as widely used as some other instant messaging applications, such as WhatsApp or Facebook Messenger.
- It is not available in some countries due to government restrictions or censorship.
- It has faced controversy over its association with the nonprofit organization the Signal Foundation and its founder, Brian Acton.
- It has had issues with spam and bots, which can be annoying for users.
- It has limited features compared to some other instant messaging applications, such as Telegram or WhatsApp.

3. Proposed Instant Messaging Product

We propose WeText, an instant messaging web app which satisfies the general needs which are satisfied by the market standard apps listed above and offers the customers with the most immediate ways of online communication. In the following

sections, the general requirements of WeText will be explained in detail, followed by an architecture overview.



Fig. 1. Logo of WeText.

3.1. Quality Requirements

3.1.1. Security

In order to prevent scamming attempts and other such illicit activities through events such as account hijackings, an instant messenger app would be expected to have security as a quality requirement [1]. In order to satisfy this need, <name> must have high authenticity, where the user who is currently sending and receiving messages is authenticated with a unique token to verify that the transmitted messages belong to the correct user. <name> must also have high data integrity, where the customer data and the cloud system is regularly checked with penetration tests to prevent breaches. In a more general sense, WeText must be ready for different cyber attacks such as malware, man-in-the-middle attack, denial-of-service attack, and phishing [2].

3.1.2. Data Privacy

Along with the emergence of GDPR, all websites, including and especially instant messengers and similar platforms, must handle customer data appropriately [3]. This means that customers must have the right to know what data will be collected by WeText and how it will be processed. Customers must also be able to delete their accounts whenever they want, in compliance with modern regulations. By satisfying these requirements and not using customer data for any purpose of which the customer is not aware of, data privacy will be ensured by WeText.

3.1.3. High Performance

Speed is an important factor in the instant messaging industry. The attention span of users generally expires after 2 seconds of waiting while using a web app [4]. This means that WeText must satisfy a baseline in several performance metrics to prevent users from getting distracted and leaving. The most important metrics to

consider are: response time, which evaluates the time elapsed for a response to reach the server, get processed, and delivered to the client (peak response time may also be considered); requests per second, which evaluates the capacity of the system for satisfying a sufficient number of HTTP requests in a second; and transaction time comparison, where the expected and actual time taken to respond to a request is evaluated to determine the performance state of the system [5].

3.1.4. Reliability

WeText must also be reliable and deliver its features as the user would expect. This means that the WeText system must be as fail-proof as possible. Availability is considered, which evaluates the ability of a system to satisfy a request at any given time, which requires that it is not failed or under maintenance at the current time []. In general, the behavior of WeText will be considered in the context of how accurately it can satisfy user requests and the probability with which it will work without failure in a given time frame [6].

3.2. Main Requirements

3.2.1. Real-Time Text Transmission

As the most fundamental requirement of an instant messenger, WeChat must support real-time text transmission. This transmission format, as opposed to teletype where a turn based system is implemented, allows both parties to type messages at the same time [7]. This method has several advantages, such as allowing instant communication without a turn-based system, working better over IP, providing a broader range of text characters, etc. WeText must therefore support real-time text transmission by using the push technology. Here, a transaction, which is the delivery of a message from the sender to the recipient, is initiated by the sender's device which acts as the "server" which pushes the request. The message is then received by the client with the counterpart of "push," which is "get." [8]

3.2.2. Emojis

Emojis are used to express ideas and feelings over text messages, and have become integral components of text communication and languages over the recent years. Therefore, WeText must support this feature. In order to include emojis in our app, we will use their corresponding unicode values and utilize Unicode Emoji data files [9].

3.2.3. Voice over IP

Voice over internet protocol (VoIP) is a technology which allows making audio calls through a broadband internet connection as an alternative to making calls through an analog landline. The system works as illustrated below:



Fig. 2. VoIP diagram.

As voice chat is an important feature for a messenger application, where people are able to make international calls for a small fee, it is important for WeText to utilize it. With this technology, the sender's voice is converted into a digital signal and transported through the internet to the receiver in a simultaneous and reciprocal fashion. If a regular phone number is being used, the received digital signal would finally be converted to a phone signal before reaching the recipient [10]. Since WeText is an account based web app, this step will not be necessary.

3.2.4. Video Chat

Video chat is also an important feature which is expected from a messenger app. There are several types of video chat software, such as Zoom for conference calls with several people, Discord for entertainment and streaming, and WhatsApp for mostly one-to-one video calls. Since the domain of WeText is the most similar with WhatsApp's video call feature, this feature will be similar for our proposal. Using the WebRTC (real-time communication) protocol, peering and streaming video data independent of a server in a secure way is possible [11]. Therefore, our proposal will utilize this technology to satisfy the video chat feature.

3.3. High Level Architecture

The general architecture of the proposed system and the view model will be outlined in this section. We will use the 4+1 View Model in our architecture, where the whole system will be analyzed by different views which focus on different problem sets, and are taken on by the perspectives of different stakeholders. The main structure of our views is as follows:

	Logical View	Process View	Development View	Physical View	Scenarios
Developer / Engineer					
UI/UX Designer					
Project Manager					
Customer					

Fig. 3. View model matrix.

Here, the views are listed on the horizontal axis and the views of different stakeholders (in a generalized way) of the proposed system are given on the vertical axis. Different stakeholders focus on different views at different rates according to the context of their perspectives. For example, developers/engineers will primarily focus on the development and physical views. These views are explained in more detail as follows:

3.3.1. Logical View

This view focuses on the main functionality of the system and is described with static UML diagrams such as a class diagram [12]. The customers, who are the main driving force in determining the requirements of such an instant messenger, have a large influence on the logical view where they shape the general outline of the system with their demands. Through this process, the outline of the model classes and the data structure of WeText is determined. For example, as per the explained requirements, features such as real-time text transmission and emojis are determined according to this view (other requirements are not included in the actual implementation according to the guidelines). A class diagram of the system structure and the functionalities it provides is as follows:

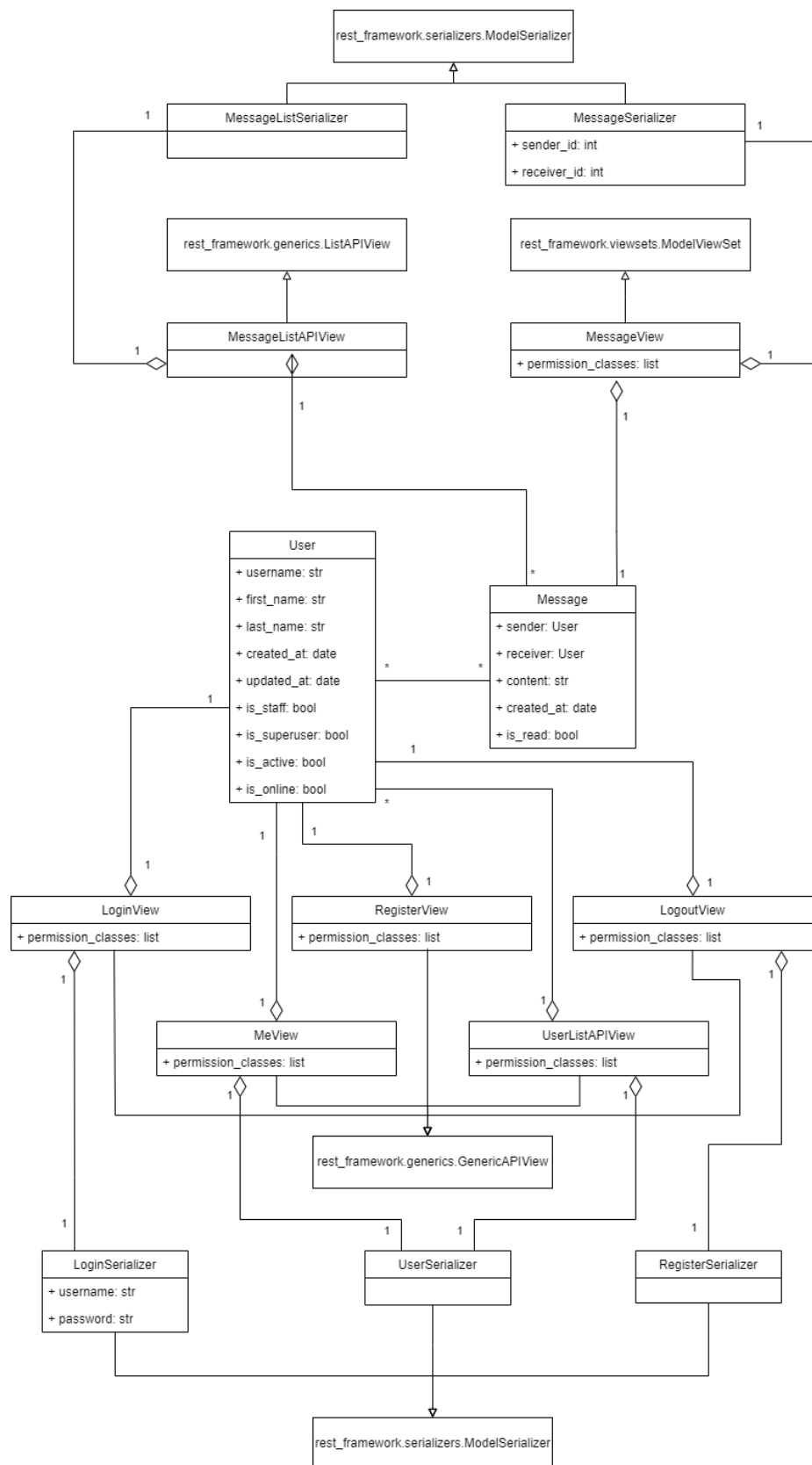


Fig. 4. Class diagram.

3.3.1.1. Generalization Module Style

This style captures the general attributes between modules and classes. The details of how it is used in our system are included with UML notation in the above Fig. 4. A code example of generalization where the MessageView class extends ModelViewSet is as follows:

```
class MessageView(ModelViewSet):
    serializer_class = MessageSerializer
    queryset = Message.objects.select_related('sender', 'receiver')
    permission_classes = (IsAuthenticated, )

    def create(self, request, *args, **kwargs):
        if str(request.user.id) != str(request.data.get("sender_id", None)):
            raise Exception("Only sender can create a message")

        serializer = self.serializer_class(data=request.data)
        serializer.is_valid(raise_exception=True)
        serializer.save()

        handle_request(serializer)

        return Response(serializer.data, status=status.HTTP_201_CREATED)
```

Fig. 5. Code example of generalization.

3.3.2. Process View

The process view focuses on the execution environment of the software system. In a sense, it describes the concurrency and synchronization state of a running system. Component & connector (C&C) styles represent such elements, i.e. the ones which have a runtime presence such as remote servers and memory-related components, and their interactions. A sequence diagram describing the process view is as follows:

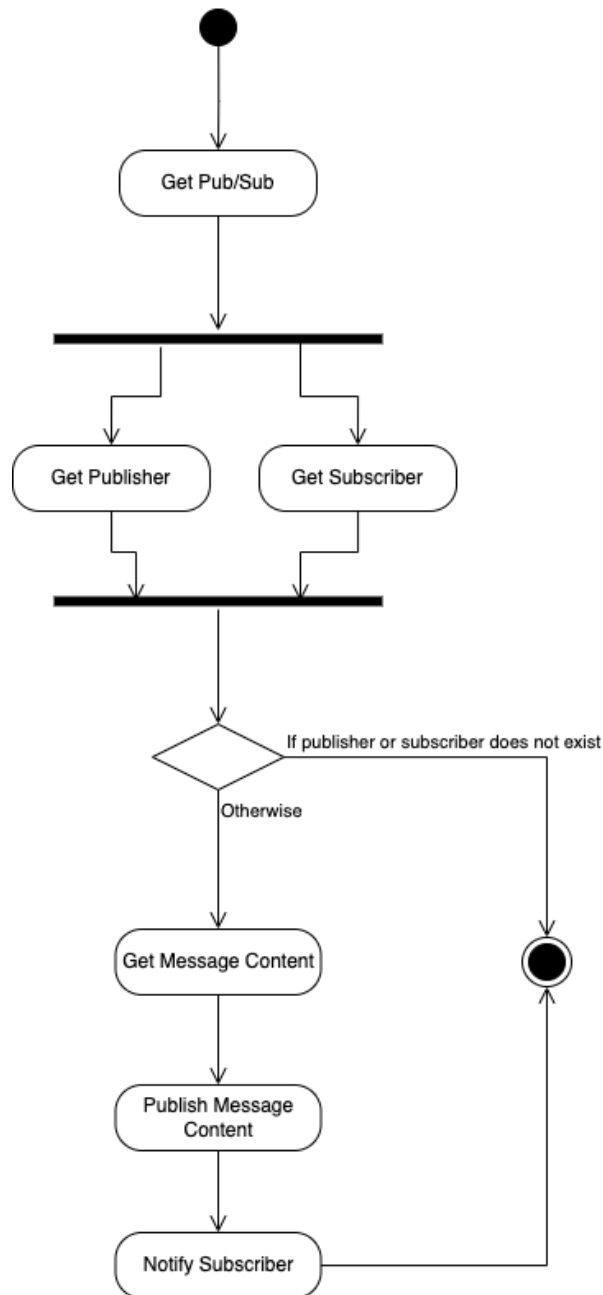


Fig. 6. Activity diagram.

3.3.2.1. Publish-Subscribe C&C Style

In the implementation of WeText, this C&C style will be used to create an interchangeable client-server structure in the delivery of instant messages. Due to the need for asynchronous communication between components to ensure that WeText satisfies real-time text transmission. In this style, a sender publishes an event and an arbitrary number of subscribers receive and process that event. In our case, there is a one-to-one correspondence between publishers and subscribers, and the processing consists of displaying the message to the receiver on the frontend. The below figure demonstrates the “publish” stage of this style.

```

def handle_request(serializer):
    notification = {
        "message": serializer.data.get("content"),
        "from": serializer.data.get("sender_id"),
        "receiver": serializer.data.get("receiver_id")
    }
    headers = {
        "Content-Type": 'application/json',
    }
    try:
        requests.post(settings.SOCKET_SERVER, json.dumps(notification), headers=headers)
    except Exception as e:
        pass
    return True

```

Fig. 7. Code example of handling message publish.

3.3.3. Development (Implementation) View

The development view illustrates the environment in which the system is implemented and its static structure. This view is the closest one to the developer/engineer's perspective as it directly deals with software management. Components, modules, and packages are important building blocks of this view, and are explained in this package diagram:

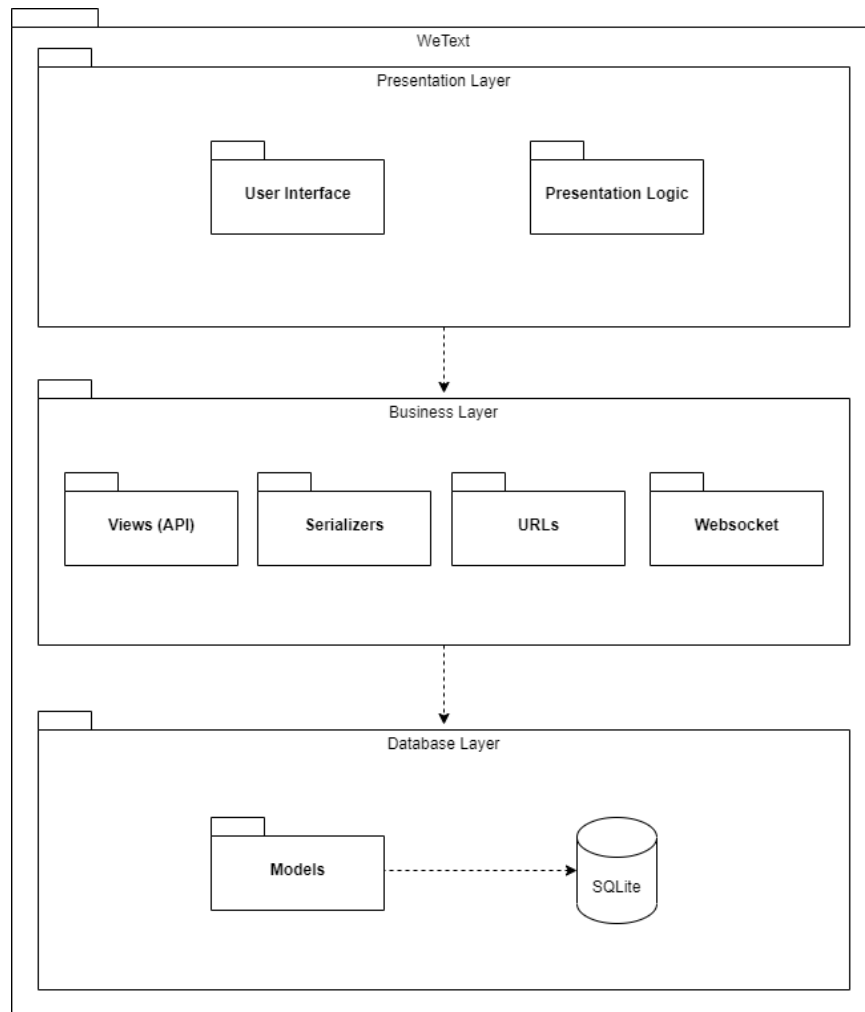


Fig. 8. Package diagram.

3.3.3.3. Layered Module Style

This module style is useful for explaining the linear (single-directional) dependencies between highly related module sets and is closely related with the MVC design pattern used in our proposed system [13]. In WeText, we have presentation, business, and database layers.

3.3.4. Physical (Deployment) View

This view of our proposed system is related to its deployment to a real-world environment and is described below.

3.3.4.1. Deployment Allocation Style

This style primarily describes the relationship between software and hardware architecture. The deployment of our system is as follows:

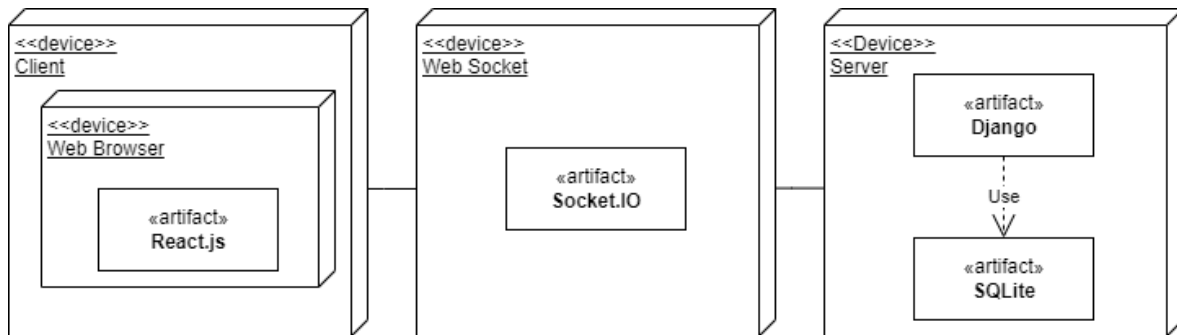


Fig. 9. Deployment diagram.

3.3.5. Scenarios (Use Case) View

This view describes the entire use case set of our system and the relations of these use cases with the overall architecture. The following use case diagram describes this in more detail:

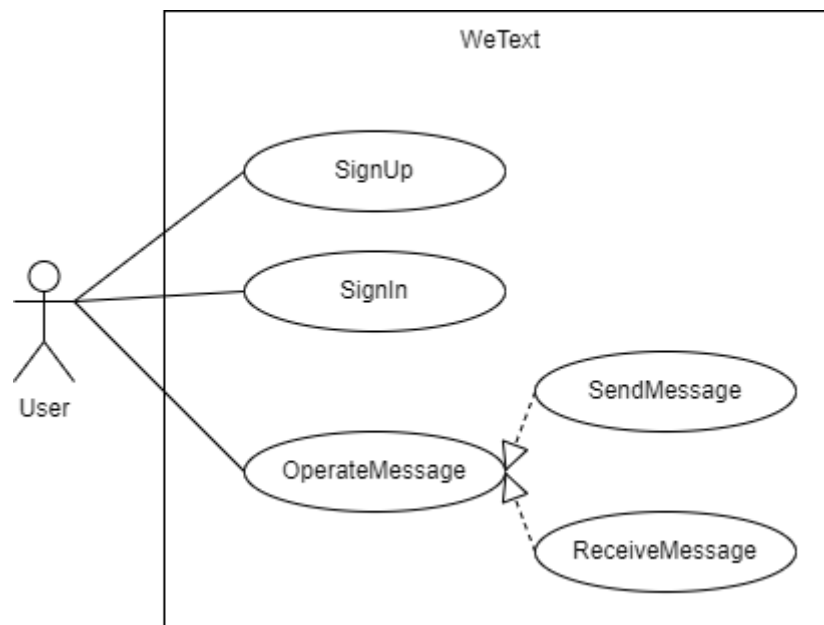


Fig. 10. Use-case diagram.

3.4. Architectural Patterns

3.4.1. Publish-Subscribe Pattern

Similar to the C&C style explained in Section 3.3.2, this architectural style is for supporting information exchange between publishers and subscribers. This architectural pattern is based on the event-bus pattern, which is explained in the following section, and extends it by providing a more flexible and scalable system. This is made possible due to the fact that, unlike the event-bus pattern, components need not be aware of each other's identities for sending and receiving messages [14]. In our system, each message receiver subscribes to a channel that belongs to a

user pair who happen to be texting. When the sender publishes a message, the subscriber receives it through the channel thanks to the websocket. The code of the websocket as implemented with Express.js and Socket.IO is as follows:

```
app = express();

const server = http.createServer(app).listen(port, () => { });

app.use(cors());

app.use(express.static(path.join(__dirname, "client")));

app.use(bodyParser.json());

app.post("/server", (req, res) => {
  io.emit("command", req.body);
  console.log(req.body);
  res.status(201).json({ status: "reached" });
});

let io = socket.listen(server, {
  cors: {
    origin: 'http://localhost:3000'
  }
});

io.on("connection", socket => {
  socket.on("command", data => {
    io.emit("command", data);
  });
});
```

Fig. 11. Code of the websocket.

3.4.2. Event-Bus Pattern

This pattern has 4 main building blocks, namely “event source, event listener, channel, and event bus.” Here, listeners subscribe to channels on an event bus, to which event sources publish events that trigger certain events. This pattern forms the foundation of the previously explained publish-subscribe pattern [14]. In our proposed system, the events are the message publishes which are sent to user pair channels, and the websocket acts as the event bus. This forms the foundation of how the publish-subscribe pattern is used in our system.

3.4.3. Model-View-Controller Pattern

This pattern is known as MVC. It divides the program logic into three different interconnected elements. It decouples the components and allows efficient code reuse [13]. The model contains the main data. Each entity has its own model class. The view displays the information of models to the user. This representation can be in different formats like JSON or XML. The controller handles the input coming from the user and converts it to commands for view and model. In WeText, we use this

architectural pattern, where Django's Models, such as the "User" model implementation, are our models. The views are the frontend components implemented using React with which the user interacts. Finally, the controllers are LoginView, RegisterView, RefreshView, and LogoutView classes located in views.py (views are controllers in Django).

3.5. Final Product

The screens of our final product are as follows:

3.5.1. Sign In Screen

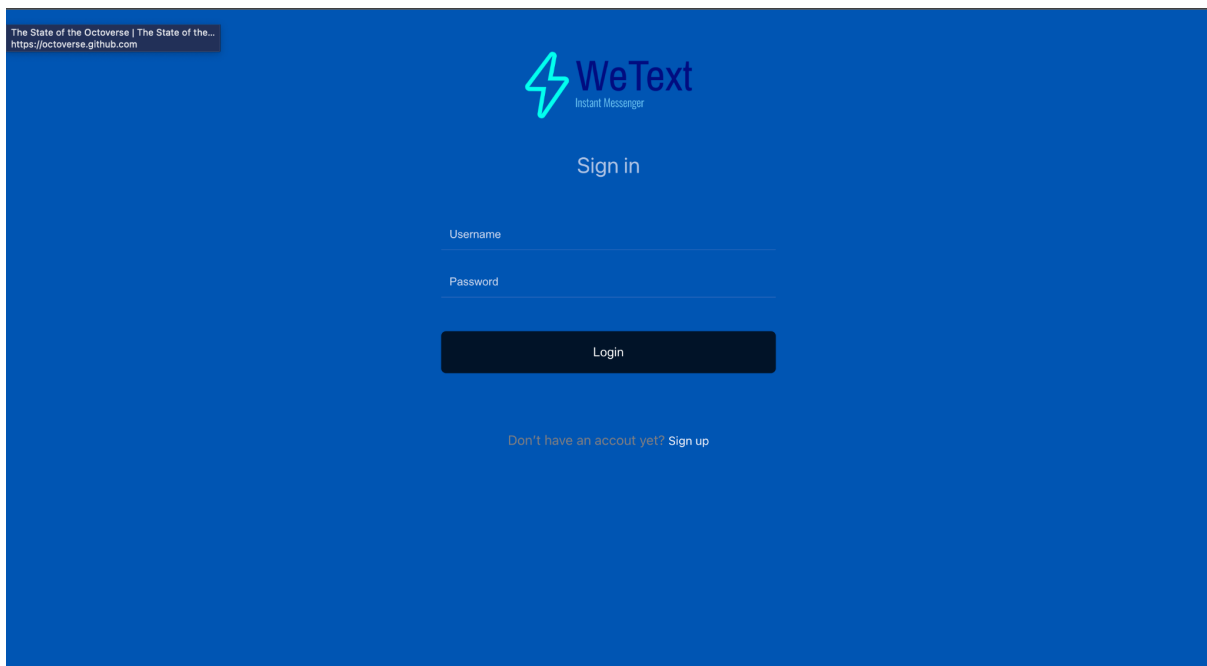
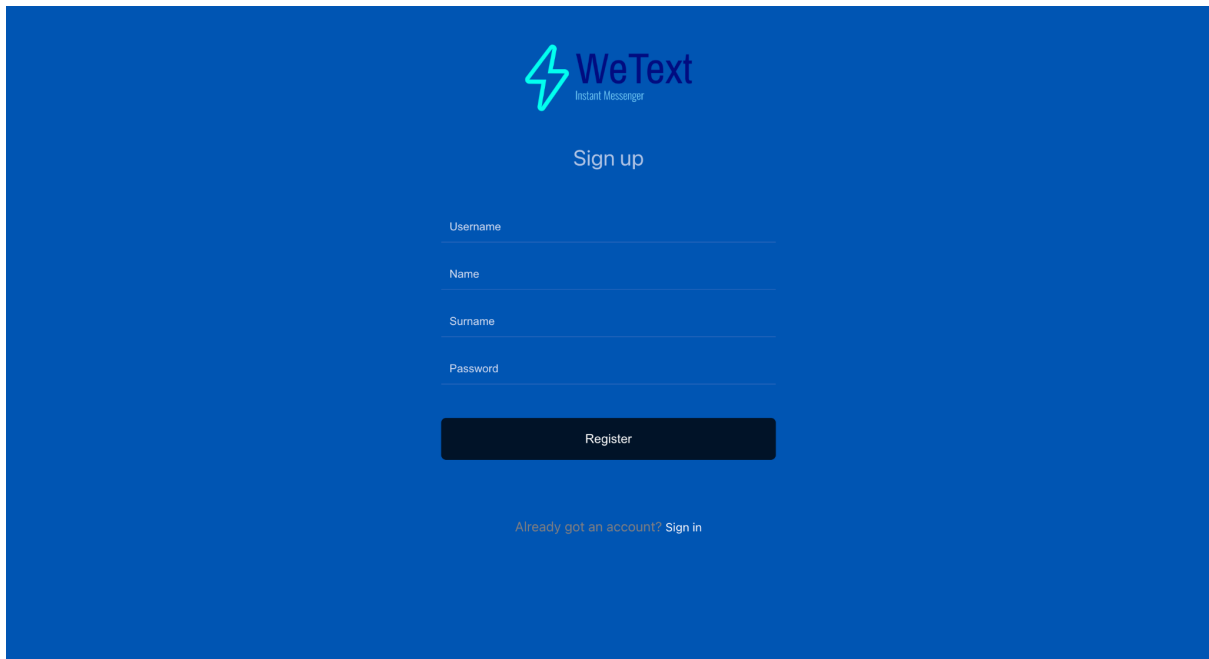


Fig. 12. Sign in screen.

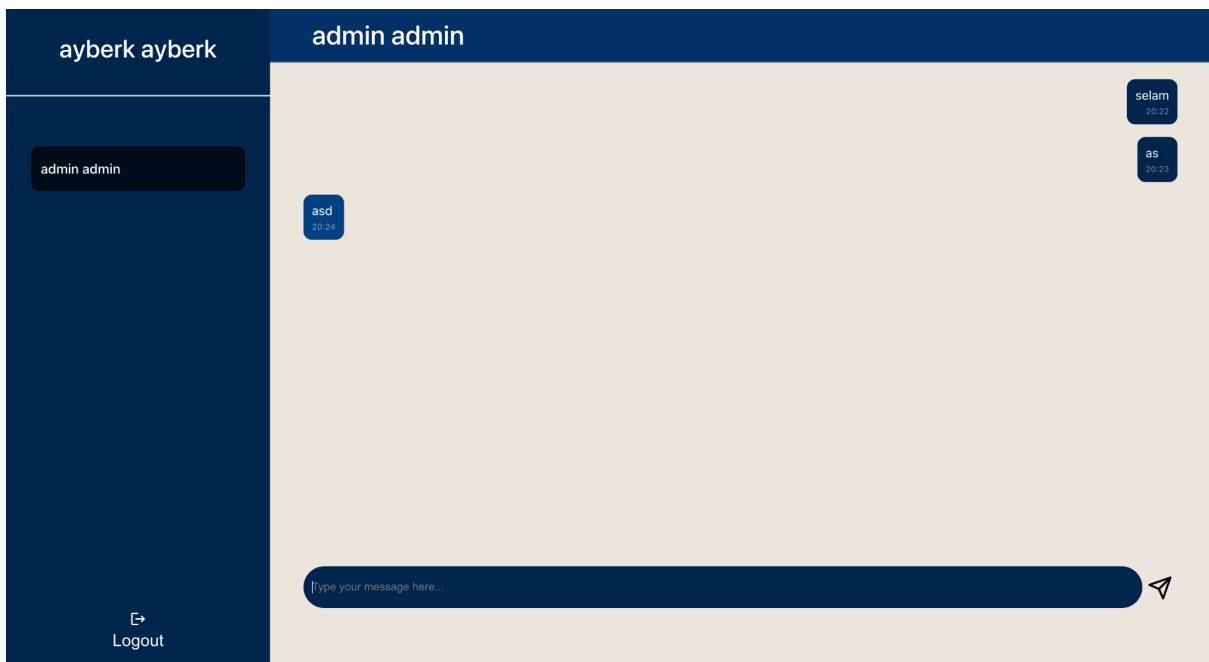
3.5.2. Sign Up Screen



The sign-up screen for WeText Instant Messenger features a blue background. At the top center is the logo, which consists of a green lightning bolt icon followed by the text "WeText" in a bold, sans-serif font, with "Instant Messenger" in a smaller font below it. Below the logo, the text "Sign up" is centered. Underneath, there are four input fields labeled "Username", "Name", "Surname", and "Password" from top to bottom. A dark blue "Register" button is positioned below these fields. At the bottom center, there is a link that says "Already got an account? Sign in".

Fig. 13. Sign up screen.

3.5.3. Messaging Screen



The messaging screen has a dark blue header bar. On the left side of the header, the text "ayberk ayberk" is displayed. On the right side, "admin admin" is displayed. Below the header, the screen is divided into three main sections. On the left is a dark blue sidebar containing a button labeled "admin admin" and a "Logout" button at the bottom. The central area is a light beige chat window. It contains three messages: a blue bubble on the left with the text "asd" and a timestamp of "20:24"; a blue bubble on the right with the text "selam" and a timestamp of "20:23"; and another blue bubble on the right with the text "as" and a timestamp of "20:23". At the bottom of the chat window is a dark blue input field with the placeholder text "Type your message here..." and a paper plane icon to its right.

Fig. 14. Messaging screen.

References

- [1] “What is software security?,” *Techopedia.com*. [Online]. Available: <https://www.techopedia.com/definition/24866/software-security>. [Accessed: 12-Dec-2022].
- [2] “Top 10 common types of cybersecurity attacks,” *Datto Security Solutions*. [Online]. Available: <https://www.datto.com/blog/common-types-of-cyber-security-attacks>. [Accessed: 12-Dec-2022].
- [3] “Official Legal Text,” *General Data Protection Regulation (GDPR)*, 27-Sep-2022. [Online]. Available: <https://gdpr-info.eu/>. [Accessed: 12-Dec-2022].
- [4] “The Psychology of Web Performance,” *The Uptrends Blog*, 09-Aug-2018. [Online]. Available: <https://blog.uptrends.com/web-performance/the-psychology-of-web-performance/>. [Accessed: 12-Dec-2022].
- [5] “Software performance testing metrics for more effective testing,” *UTOR*, 28-May-2021. [Online]. Available: <https://u-tor.com/topic/performance-testing-metrics>. [Accessed: 12-Dec-2022].
- [6] A. Iannino and J. D. Musa, “Software reliability,” *Advances in Computers*, pp. 85–170, 1990.
- [7] “Real-time text: Improving accessible telecommunications,” *Federal Communications Commission*, 27-Jan-2021. [Online]. Available: <https://www.fcc.gov/consumers/guides/real-time-text-improving-accessible-telecommunications>. [Accessed: 12-Dec-2022].
- [8] M. Thomson, E. Damaggio, and B. Raymor, “Generic event delivery using HTTP PUSH,” *IETF Datatracker*, 08-Dec-2016. [Online]. Available: <https://datatracker.ietf.org/doc/html/draft-ietf-webpush-protocol-12>. [Accessed: 12-Dec-2022].
- [9] “Full emoji list, V15.0 - Unicode.” [Online]. Available: <https://unicode.org/emoji/charts/full-emoji-list.html>. [Accessed: 12-Dec-2022].
- [10] “Voice over internet protocol (VoIP),” *Federal Communications Commission*, 01-Nov-2015. [Online]. Available: <https://www.fcc.gov/general/voice-over-internet-protocol-voip>. [Accessed: 12-Dec-2022].
- [11] “How to make a video chat app in 2022 making no mistake,” *Cleveroad Inc.* [Online]. Available: <https://www.cleveroad.com/blog/build-video-chat-app/>. [Accessed: 12-Dec-2022].
- [12] P. B. Kruchten, “The 4+1 view model of Architecture,” *IEEE Software*, vol. 12, no. 6, pp. 42–50, 1995.
- [13] “10 common software architectural patterns in a Nutshell.” [Online]. Available:

<https://towardsdatascience.com/10-common-software-architectural-patterns-in-a-nutshell-a0b47a1e9013>. [Accessed: 12-Dec-2022].

- [14] “Everything you need to know about publish/subscribe,” *Ably Realtime*. [Online]. Available: <https://ably.com/topic/pub-sub>. [Accessed: 12-Dec-2022].