TED UNIVERSITY

CMPE 492

Senior Design Project II

MESAJLA

Low Level Design Report

09/03/2025

Group Members:

- **Ahmet Tunahan Küçükgökce**
- **Altar Gürsoy**
- **Burak Eren Birinci**
- **Serdar Kemal Topkaya**

Table of Contents

1 Introduction

This report introduces the low-level design of Mesajla. Mesajla is a real-time chatting application. The most important feature that distinguishes Mesajla from other similar applications is that it allows you to use AI-generated responses during chatting. Mesajla's model, which is trained with a lot of data, shows AI-generated responses in the user interface during chatting. The user can choose one of them or change them to see new responses. It offers a nice chatting experience thanks to a user-friendly interface. You can perform an easy registration process with the sign-up option. Mesajla offers many features to its users. These include creating groups, changing profile settings, using AI-generated messages, displaying the time the messages were sent, and showing the online status of the users.

The rest of the report covers object design trade-offs, i.e., the reliability, usability, and performance of Mesajla. Then, we will show the interface documentation guideline used in the report and the engineering standards we used in the development process of the application. Finally, we will introduce the packages that make up Mesajla's front-end and back-end, and the communication between these packages, followed by the class interf aces for the front-end and back-end.

In general, this report introduces the low-level design of our application, the methods used in the application, and the implementation plan that enables Mesajla's features to be applied.

1.1. Object Design Trade-offs

While constructing this application, there are some key points to enhance the user experience and provide a reliable and secure environment. Involvement in the application of the users is also ensured by the regulations which protect the users' privacy such as GDPR. This project is built collaboratively with these regulations. To fulfill these requirements, there are some considerations while developing the application between scalability, flexibility, security and so on. Here are these decisions:

1.1.1 Modularity over Performance

While a highly modular system is preferable to increase maintainability and scalability, it also may cause performance overheads.The system is built as a microservice-based structure, using a modular system, to separate authentication, UI, chatting subsystem, database management and AI into distinct subsystems.

1.1.2 Flexibility over Complexity

Creating a flexible application to respond to making changes in the structure becomes more smoother, rather than building a complex one. The application needs to respond to the

outputs of GenAI successfully. To accomplish this, it is made more flexible to improvise the changes of the AI model.

1.1.3 Scalability over Cost

Build a system that prioritizes handling the workload but also increases the cost of the application in terms of supply. Preferring a well-scaled structure in the application helps to upgrade databases and the AI model.

1.1.4 Accuracy over Efficiency

Make the responses of the AI more accurate. However, it is also leading the system to become more time consuming.To get more accurate responses of dialogues from AI, we train its' model with large scale datasets, making it more time consuming to train and validate.

1.1.5 Both security and Usability

In order to ensure the security of the program, the application improves the usability of the authentication. Providing helpful functions for users to protect them in authentication, with respect to the GDPR.

1.1.6 Privacy over Accuracy

The subsystem of AI focuses on the privacy of user data rather than the accuracy of training the model. AI model collects user data and trains with them. However, it prioritizes the privacy of these data above than the success rate of the training.

1.2. Interface Documentation Guidelines

Following documentation style is used. Class Interfaces represented in the table style below:

Class	Class Name
Description	Description of the Class
Variables	Variables of thee Class
Methods	Methods of the Class

Table 1: Interface Documentation Guideline

1.3. Engineering Standards

The application is designed for applying software engineering standards. These standards are to check whether the fulfillment of the requirements is established while developing the project. The requirements that are set for this application cover the protection and security of the users and their personal data, the flow of tasks of classes, the efficiency and performance of the overall system and so on. Here's the engineering standards we use in our project:

1.3.1 Unified Modeling Language (UML) Standards

In this project, there are UML diagrams used to plan the architecture of the subsystems such as class, sequence and object diagrams. (See the Project Analysis Report for further information)

1.3.2. IEEE Software Engineering Standards

The project applies to some of the IEEE Standards:

- **IEEE 830-1998 (IEEE Recommended Practice for Software Requirements Specifications):** Software Requirements Specifications (SRS) are provided by the reports of this project.
- **IEEE 1471-2000 (IEEE Recommended Practice for Architectural Description for Software-Intensive Systems):** The architecture of the system, workflow of its subsystems are analyzed and explained.
- **IEEE 1016-2009 (IEEE Standard for Information Technology--Systems Design--Software Design Descriptions):** The design of the system is described in High-Level Design Report and Low-Level Design Report.

1.3.3. Cryptographic Protocol Standards

The chatting in the application is encrypted throughout the users. Third-party access from outside isn't allowed. For that, the application follows TLS 1.3 encryption standards.

1.3.4 Data Protection and Privacy

The system follows the regulations provided by General Data Protection Regulation (GDPR). It features data protection and privacy settings.

1.4. Definitions, acronyms and abbreviations

UML: Unified Modeling Language

IEEE: Institute of Electrical and Electronics Engineers

GDPR: General Data Protection Regulation

2 Packages

2.1. Front-End and Back-End Packages

This diagram is showing two main packages a Front End package and a Back End package and how the classes in those packages connected to each other.

- **Front End Package**: Contains UI classes like Login, Detail, Chat, UsersList, ChatList, and AddUser. These are responsible for displaying information to the user, handling user input, and providing the app's overall interface.
- **Back End Package**: Contains classes like ChatStore, Upload, and Firebase that handle data storage, file uploads, and communication with the database or server. These classes provide the services and data that the front end needs to function.

The lines between the classes show which classes communicate and connected with each other. The Front end classes call the Back end classes for data, information, operations and the Back end classes provide and manage those information and resources.

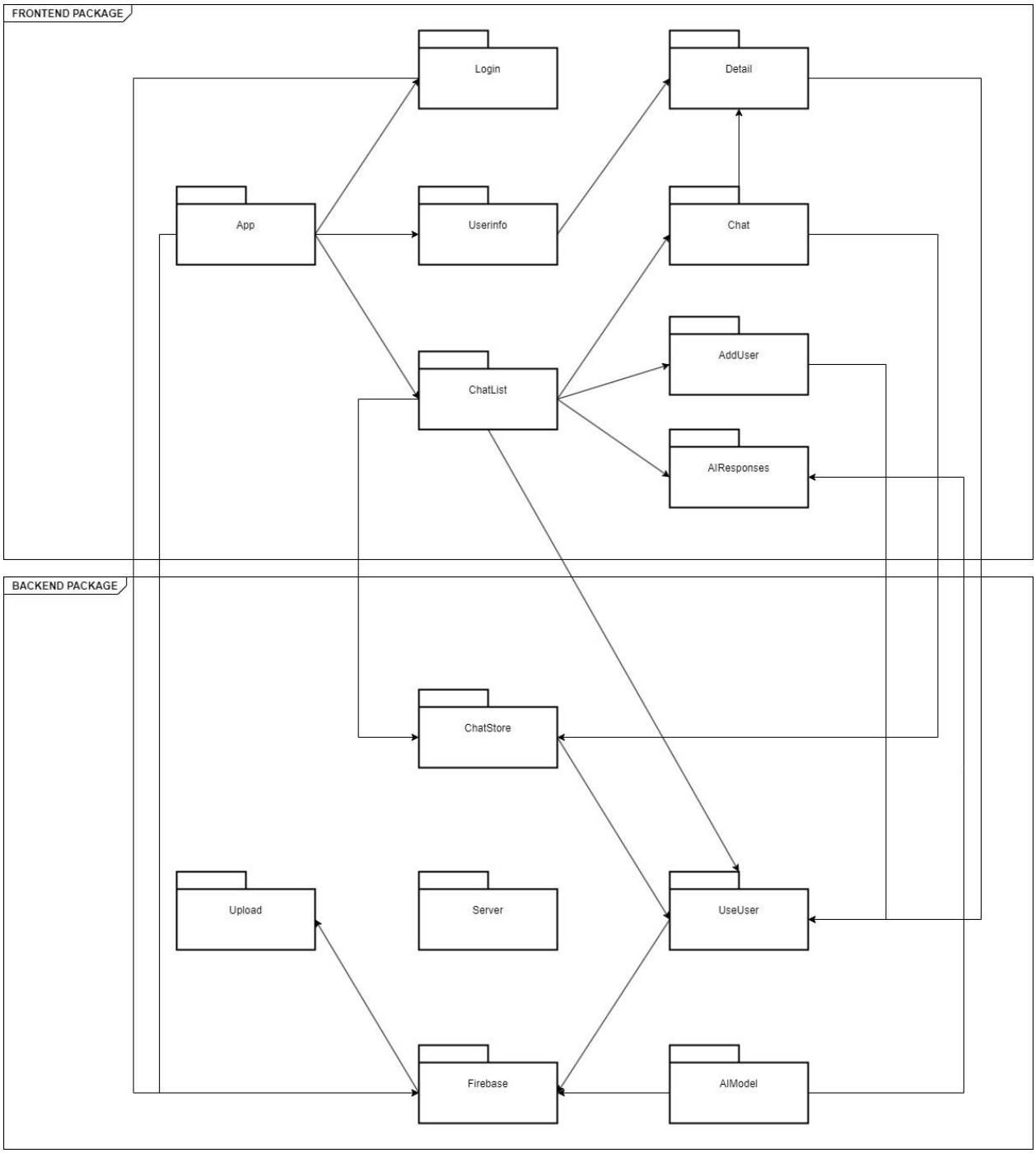

Figure 1: Front-End and Back-End Package

3 Class Interfaces

3.1. Front-End Classes

Class	Chat.jsx
Description	The Chat component manages the chat messages, allowing users to send messages and images. It updates chat data using firebase.
Variables	chat, open, text, imgcurrentUser, chatId, user, isCurrentUserBlocked, isReceiverBlockedendRef
Methods	useEffect, handleEmoji(e) ,handleImg(e) ,handleSend()

Class	Detail.jsx
Description	The Detail component manages the blocking and unblocking functions and logging out.
Variables	chatId, user, isCurrentUserBlocked, isReceiverBlocked, changeBlock, resetChat, currentUser, userDocRef
Methods	handleBlock(), handleLogout()

Class	ChatList.jsx
Description	The ChatList component displays a list of user chats, allows selecting a chat and view messages.
Variables	chats, addMode, input, currentUser, chatId, changeChat, filteredChats
Methods	handleSelect(chat)

Class	Login.jsx
Description	The Login component handles registering and logging in using Firebase Authentication.
Variables	profilepic, loading, auth, db
Methods	handleRegister(e), handleLogin(e)

Class	App.jsx
Description	The App component handles user authentication and gets user information upon login. It shows the chat interface or login screen based on the user's authentication.
Variables	currentUser, isLoading, chatId
Methods	useUserStore(), useChatStore()

Class	AddUser.jsx
Description	The AddUser component allows the user to search for and add other users by username, and it creates a new chat between the users
Variables	user, currentUser
Methods	handleSearch(e), handleAdd()

Class	Userinfo.jsx
Description	The Userinfo component shows user information such as username and profile picture. It gets the information from UseUser.
Variables	currentUser
Methods	handlepfp(e)

Class	Ai Respones
Description	This class shows the ai responses.
Variables	Responses[], response, flag
Methods	DisplayAiResponse(), UserBasesAiresponse(), deleteData()

3.2. Back-End Classes

Class	ChatStore.js
Description	The ChatStore is a store component that manages the chat state, user block status, and allows changing chats, blocking and unblocking users, and resetting chat.
Variables	chatId, user, isCurrentUserBlocked, isReceiverBlocked
Methods	changeChat(chatId, user), changeBlock(), resetChat()

Class	Firebase.js
Description	This component initializes the Firebase application and provides the authentication, Firestore, and storage to be used in the application.
Variables	firebase, app, auth, db, storage
Methods	GetConnection()

Class	UseUser.js
Description	The useUser is a component that manages the current user state and loading status, getting user information from Firestore based on the user's ID.
Variables	currentUser, isLoading
Methods	getUserInfo(uid)

Class	Upload.js
Description	The upload function is responsible for uploading a file to Firebase Storage. and returning after the upload is complete
Variables	storageRef, uploadTask, progress
Methods	uploadBytesResumable()

Class	Server
Description	Connection between backend and model
Variables	DataPath, modelData, user_input
Methods	get_response(),bag_of_words()

Class	AiModel
Description	Model that trained with data.
Variables	InputSize, outputSize, dataset, trainLoader
Methods	DataLoader()

4 Glossary

Engineering Standards: Technical documents for rules.

Database: Where data is stored.

Front-End: The visual interface of the web application.

Back-End: The system that runs on the server side of the application.

5 References

1. Institute of Electrical and Electronics Engineers. (1998). *IEEE recommended practice for software requirements specifications* (IEEE Std 830-1998). IEEE Standards Association. https://standards.ieee.org/ieee/830/1222/
2. Institute of Electrical and Electronics Engineers. (2000). *IEEE recommended practice for architectural description for software-intensive systems* (IEEE Std 1471-2000). IEEE Standards Association. https://standards.ieee.org/ieee/1471/2187/
3. Institute of Electrical and Electronics Engineers. (2009). *IEEE standard for information technology—systems design—software design descriptions* (IEEE Std 1016-2009). IEEE Standards A