UNIVERZA NA PRIMORSKEM
FAKULTETA ZA MATEMATIKO, NARAVOSLOVJE IN
INFORMACIJSKE TEHNOLOGIJE

ZAKLJUČNA NALOGA
(FINAL PROJECT PAPER)

APLIKACIJA ZA KLEPET V PHP
(CHAT ROOMS APPLICATION IN PHP)

NIKITA SIZOV

2022
ZAKLJUČNA NALOGA (FINAL PROJECT PAPER)
SIZOV

UNIVERZA NA PRIMORSKEM
FAKULTETA ZA MATEMATIKO, NARAVOSLOVJE IN
INFORMACIJSKE TEHNOLOGIJE

Zaključna naloga
(Final project paper)

Aplikacija za klepet v PHP

(Chat rooms application in PHP)

Ime in priimek: Nikita Sizov
Študijski program: Računalništvo in informatika
Mentor: doc. dr. Peter Rogelj

Koper, maj 2022

Ključna dokumentacijska informacija

Ime in PRIIMEK: Nikita SIZOV

Naslov zaključne naloge: Aplikacija za klepetalnice v PHP

Kraj: Koper

Leto: 2022

Število listov: 52 Število slik: 19 Število tabel: 5

Število referenc: 16

Mentor: doc. dr. Peter Rogelj

Ključne besede:

Problem komunikacije študentov, prototip, faze razvoja programske opreme, testiranje.

Izvleček:

V tem delu obravnavamo probleme študentske komunikacije in ponujamo rešitev v obliki klepetalnice. Predstavljamo razvojni proces programske aplikacije, ki pripelja do prototipa progamske opreme. Prototip je zasnovan na spletu in vključuje vse osnovne funkcionalnosti aplikacije za klepet, vendar ni integriran v univerzitetno okolje. Predstavljamo vse faze razvoja programske opreme, od začetne definicije problema, študije izvedljivosti, analize zahtev, načrtovanja sistema, načrtovanja komponent, implementacije in testiranja. Analiza projekta bo razkrila vse glavne pomanjkljivosti in težave te rešitve. Na koncu povzamemo rezultate projekta in uspešnost prototipa, pri čemer se osredotočimo na njegovo sposobnost reševanja komunikacijskega problema.

Key document information

Name and SURNAME: Nikita SIZOV

Title of the final project paper: Chat rooms application in PHP

Place: Koper

Year: 2022

Number of pages: 52 Number of figures: 19 Number of tables: 5

Number of references: 16

Mentor: Assistant Professor Peter Rogelj, PhD

Keywords:

Student communication problem, prototype, software development phases, testing.

Abstract:

In this work, we discuss the problems of student communication and offer a solution in the form of a chat application. We present the development process of the software application that results in a software prototype. The prototype is web based and includes all the basic functionalities of the chat application, but it is not integrated into the university environment. We present all the software development phases, from the initial problem definition, feasibility study, requirements analysis, system design, component design, implementation, and testing. The analysis of the project will reveal all the main disadvantages and problems with this solution. In the end, we summarize the outcomes of the project and the success of the prototype, focusing on its ability to solve the communication problem.

ACKNOWLEDGEMENTS

I would like to express my deep and sincere gratitude to my project supervisor, Assistant Professor Peter Rogelj, PhD, for giving me the opportunity to do a thesis on this topic and providing invaluable guidance throughout this work. Thanks to his advice on methodology, adjustment of the set goals, and identification of problems, the work has acquired a more correct and deeper look. I am also grateful to Assistant Professor Dr. Klen Čopič Pucihar, PhD, whose lectures inspired me to learn more about creating websites and improving existing systems.

My completion of this project could not have been accomplished without the support of my good friends, students of the University of Primorska, Olga Pugacheva and Olexandr Babenko. They provided the necessary assistance in the editing of this work and gave me moral support.

I am very grateful to all these people.

LIST OF CONTENTS

TABLE OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

1 INTRODUCTION

Information transmission is one of the important aspects of work efficiency in all areas. For students, there is always an urgent question of how they can transfer information to each other quickly and conveniently. Since many problems are solved by a communicative method, this aspect is very important and affects the student's efficiency.

Not so long ago, we started to use electronic mail as the main means of communication at work and while studying. People could exchange not only text messages, but also documents, pictures, and other files. This approach made the information exchange much easier. However, e-mail represents a means of sending information rather than communicating. Nowadays, for these purposes, we use instant messengers. The use of messengers allows students to create groups in which they can discuss studies, exchange thoughts and ideas, send important files, and convey information. Thanks to this approach, students can be more immersed in the studying process, quickly solve problems related to their studies, and help others.

However, there are also disadvantages. To transfer any information through the messenger, it is necessary that the person should be in the contacts list. And the process of finding a person through acquaintances may not always be successful. There is also another problem: people use different messengers, such as Telegram or WhatsApp, and they don’t want to use something else. In this case, messengers are not an effective means of communication and information transfer.

One of the faculties of the University of Primorska, FAMNIT, has a system of communication between professors and students. Each subject can have a forum tab where students are able to ask questions about the subject, and discuss the educational process with teachers. Unfortunately, this method has proven to be ineffective. Teachers show little activity and don't have any discussions there. Students prefer to communicate with professors by mail instead of the forum. Also, many subjects do not have this forum, as teachers don’t include it. This contributes to the low activity of students there. As a result, the forum is not used for discussions, but only for public questions, such as indicating the date of the exam. And this is very bad, since a lot of students’ problems remain unresolved. Therefore, it is necessary to implement a system in which they could simply communicate, discuss topics related to their studies, solve problems with their studies, ask each other questions, as it works in messengers. A single platform for all students and teachers.

The idea of the diploma work is to implement a prototype of a website where students will be able to communicate and transfer information about their studies through chat rooms. A system in which students can go under their student ID will allow them to communicate with each other, create group rooms, and discuss subjects. This approach can solve many problems:

- It will no longer be necessary to form groups in messengers, exchange contacts, or register in new messengers. Each student will already be included in the chat room of their course.

- Students will have the option to chat with everybody. This option will allow them to solve various problems with studying and exchanging information. Since there will be all students in the system, it is much more likely that there will be people who already know the solution to the problem.
- Professors will be able to create discussions in which not only students of this course can participate, but also other students and teachers. It will help teachers to establish contact with students.
- Students can create a discussion if they want others to know about the problem.
- It will be possible to create chats for project discussions.
- Also, the system can help to conduct surveys or organize different events via chat.

To develop such a project, we will use modern methods and technologies. Web frameworks will simplify our development tasks, as well as make shorter the development time. Therefore, in this work, we will also determine which frameworks we need, what are their positive and negative sides, and how they simplify project creation. Also, we will analyze the implementation of the entire project code, highlight the key development points, and determine the need for some tools.

The result of the work will be a prototype of a website implemented by a web framework and used to visually demonstrate the capabilities of this framework. In the end, we will determine how successful its implementation is, and how important it is to use the framework.

2 FEASIBILITY STUDY

2.1 Financial feasibility

The chat rooms system is a web application. Accordingly, for the implementation and further maintenance of this project, we need a web server. This is no problem for developing the application, as it will be possible to use a local server and save the project code in the git repository. For further implementation, the system can be set up on the university's servers for free. Since a large number of functions are not planned in such a system, we can say that one person is enough for support of the application. Also, despite the fact that the platform will need to support a large number of users, it can be said that it won't need to allocate a huge amount of space on the servers. Since chats don't take up much space, and with annual cleaning, hard drives won't be clogged with information. So, we can conclude that a system like this is financially feasible.

2.2 Technical feasibility

To develop such a project, we may need a local server to test the changes in the application and eventually upload and build the project. Or we can use our system as a server and build everything on it. If the system were located on the servers of the university, there would be no need to separately add new users and delete old ones, since the system shares a common user base with the university.

For implementation, we can use such technologies as web frameworks and libraries, which can speed up the development process. It would be a good idea to use a git to back up the project. Every technology is freely available, and the required technical skills are readily available. The time constraints of project development and the ease of implementation using these technologies are synchronized with this, it is clear that the project is technically feasible.

2.3 Legal feasibility

Such a system complies with all GDPR standards. All personal data of users will also be stored on the server of the university. No personal data will be published in the website, only the real names of the users. The system will be protected by means of the framework from unauthorized entry by persons not related to the educational process. Also, if the data in the chat is prohibited by the authorities, is invalid or does not relate to the topic, then it will be deleted by moderator.

3 REQUIREMENT ANALYSIS

3.1 Functional and non-functional requirements

For a developer, knowing exactly what features and functionalities should be in the system is quite challenging. To avoid any misunderstandings, it is important to do the requirements analysis. It is a very critical process that enables the success of a system or software project to be assessed. Requirements are generally split into two types: Functional and Non-functional requirements.

Functional requirements of the system:

1. The system allows users to log into the account. It can only be used by authorized users. The user can be students and teachers.
2. Users can enter rooms from the public and private list of rooms on the site. The prototype aims to implement a list with public chat rooms. However, it will also be necessary to make a list with private rooms. Since we create a prototype on a local machine, we do not pursue this goal.
3. The users can create public and private chat rooms with custom names and descriptions. The description is necessary so that users can immediately understand the topic of discussion in the room.
4. The user can change the name and description of the room that he created.
5. The user can chat in any public room on the site.
6. The system allows users to find chats by search.

Non-functional requirements of the system:

1. The application must be well optimized on any device (PC, mobile, tablet).
2. The chat must support an unlimited number of users.
3. The system must be secure and support modern protocols and certificates such as HTTPs, sFTP/FTPs, SSL.
4. The system should support different types of users: students, teachers, and moderators. Moderators support the system and delete posts in case of violations.
5. The system should handle approximately 200k messages per day.
6. The system should be able to process about 100 parallel connections with the server.
7. Opening pages should take no more than 6 seconds.

3.2 Use Case diagram

It is a good decision to use the “Use Case Diagram”. The purpose of this diagram is to capture the core functionalities of a system and visualize the interactions. With the help of Use Case Diagram, we can find all the interactions with the actors. It is helpful for analyzing the requirements and modeling the basic idea behind the system.

We consider a Use Case diagram at the sea level in which we describe the goals of the user in the system. In order to better understand the diagram, we need a **Use Case Description**:

Since the application can be used by both teachers and students, we combine them into one actor - the user. They have equal functionality in the system, so this is permissible. The user is the main actor of the system, who has access to all the main functionality of the platform.

We have identified six cases for the user:

- Login case – user needs to auth in the system, since the website cannot be used without authorization.
- Write message case – user has opportunity to send messages to the chat using text field and send button.
- Receive message case – After sending a message by any user, every user of the chat receives this message in real-time.
- Creation a room case – If the user can't find a chat with the desired topic, he can create a chat room. To do this, he needs to specify the name and description of the room.
- Search a chat room case - the user has the opportunity to view the list of rooms and select the desired one.

Another actor of the system is Moderator. The purpose of this actor is to support the system, delete and edit all information on the site. He directly interacts with the database and the server.

We have identified three cases for the moderator:

- Deleting a room case - If any rules are violated in the chat, the moderator has to delete these records.
- Update information case - It is necessary to maintain the system, update the information on the dashboard, correct the data the name and description of the chat rooms.
- Create private rooms - Some private chat rooms, such as the course room, will need to be pre-defined by a moderator. So, he needs to create some of them.

Use Case Name:	Login
Actor:	User
System requirements:	• User has to have a university account. • Website is opened.
Triggers:	• Click to the "LOGIN" button. • Open any page without being logged id.
Main flows:	1. The login page opens with username and password fields. 2. User enters his login data and clicks the "LOGIN" button below. 3. The system validates the user data. 4. User gets access to the platform. Opens the main page.
Postconditions:	The user can use the system only while he is authorized in it. If he logs out of his account, he needs to log in again.
Alternative flows:	2a If one of the fields is empty: 1. The system displays a message that all fields must be filled in.

	3a If the entered data is incorrect: 1. The system displays a message that the password or id is incorrect.

Use Case Name:	Search a chat room
Actor:	User
System requirements:	• User must be authorized in the system.
Triger:	Open chat rooms list page by clicking the link in the header.
Main flows:	1. User scroll the page, check the name and description of the rooms. 2. After the user has found a suitable room, he opens it by clicking.
Postconditions:	The user opens the chat room of interest and leaves the chat rooms list.
Alternative flows:	1a If user can't find a suitable room: 1. User clicks on search field. 2. He enters the name of the desired room. 3. Opens the room that appears 3a If the chat that the user has opened is not suitable: 1. The user returns to the previous page. 2. Chooses another room. 3b If the chat that the user has opened is not suitable: 1. The user selects another room using the list of rooms on the chat page.

Use Case Name:	Write a message
Actor:	User
System requirements:	• User authorized in the system. • The chat is opened.
Triger:	Click on the "SEND" button that is prepared with a text field in the chat and is located at the bottom.
Main flows:	1. User writes a message to the text field. 2. User sends the message by selecting the "SEND" button or clicking "Enter".
Postconditions:	All chat users get this message and it saves to the database.
Alternative flows:	2a If the text field is empty: 1. The message isn't sent and other users don't receive it.

Use Case Name:	Receive a message
Actor:	User
System requirements:	• User authorized in the system. • The chat is opened.
Triger:	Some of the users send message.
Main flows:	1. A new message appears in the chat. 2. The users check it in the bottom of the messages list.
Postconditions:	The user doesn't receive any unnecessary notifications. Chat includes all user messages in the format: username and its message.

Use Case Name:	Create a chat room
Actor:	User
System requirements:	• User authorized in the system.
Triger:	Click on “CREATE” button.
Main flows:	1. User opens creation room page. 2. User enters the room information: title and description and clicks the “CREATE” button. 3. System validates the data. 4. User gets successful creation message.
Postconditions:	• The room data saves in a database. • The chat adds to the chat rooms list.
Alternative flows:	2a If one of the fields is not filled: 1. User receives a message about the need to fill in all the fields. 4a If the data not corresponds to system requirements: 1. User receives an error message.

Use Case Name:	Delete room
Actor:	Moderator
System requirements:	• Actor must have a valid login and password to access the database.
Triger:	Enters to PostgreSQL Admin
Main flows:	1. The moderator connects to database system. 2. The moderator checks the rooms records. 3. The moderator removes the records by choosing delete option.
Postconditions:	All records associated with the room are deleted from the system.
Alternative flows:	3a If there are no problems: 1. Nothing is deleted.

Use Case Name:	Update information
Actor:	Moderator
System requirements:	• Actor must have a valid login and password to access the database.
Triger:	Get update request.
Main flows:	1. The moderator connects to database system. 2. The moderator checks the records. 3. The moderator edits the necessary records.
Postconditions:	All information in the system are updated.
Alternative flows:	3a If it needs to change the information that is not in the database: 1. The moderator opens the projects files. 2. The moderator edits information.

Use Case Name:	Create private rooms
Actor:	Moderator
System requirements:	• Actor must have a valid login and password to access the database.

	• There is a need to create a private room in the system.
Triger:	Get creation chat request.
Main flows:	1. The moderator connects to database system. 2. The moderator creates a closed room. 3. The moderator adds users to this room.
Postconditions:	A private room appears in the system, which can only be entered by its members.

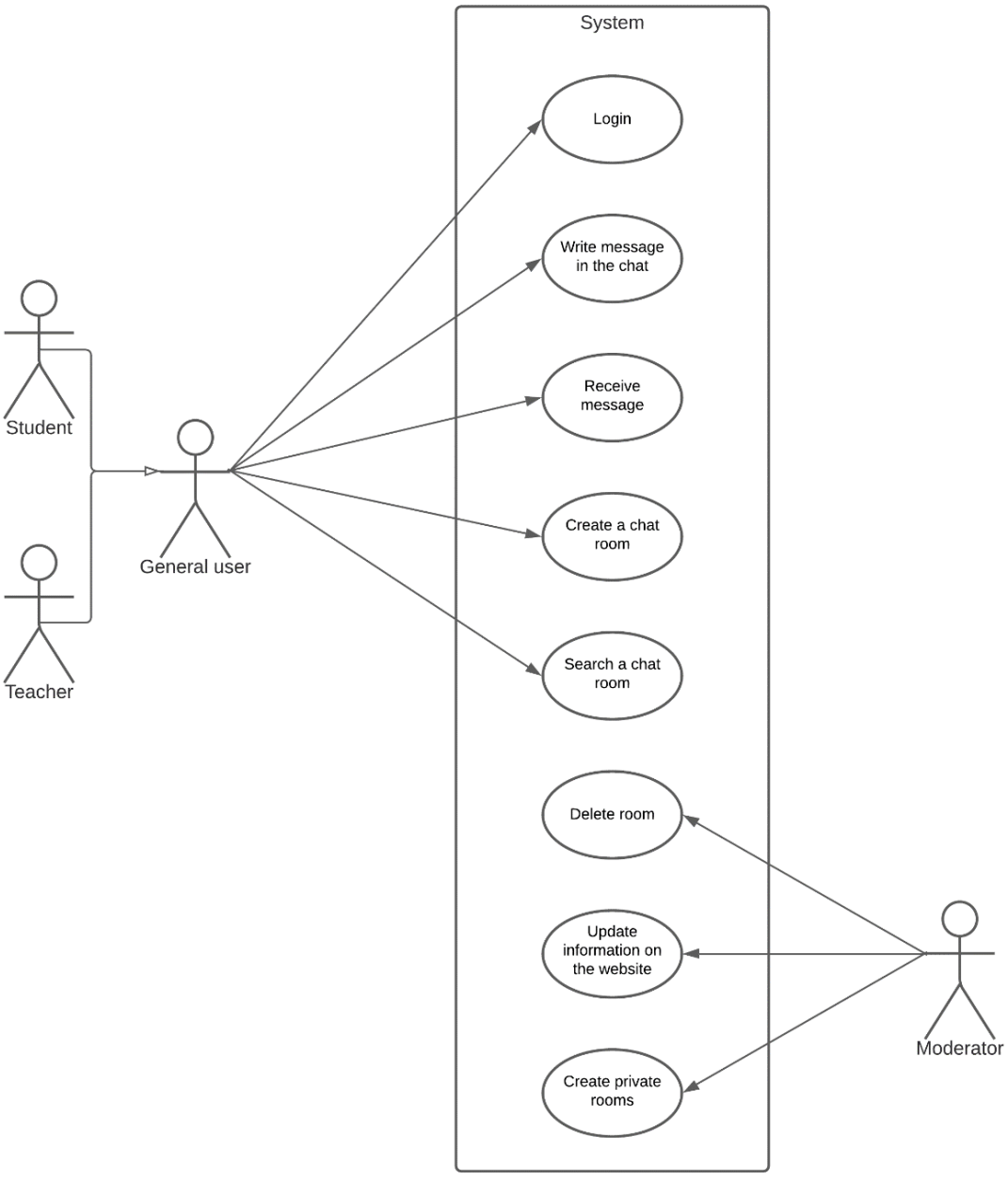

Figure 1: Use Case diagram

3.3 Structure of the website

One of the main stages of website creation is the development of its structure. It determines the location and relationship of all elements of the website: sections, subsections, and pages. The structure should be logical. It is necessary to highlight the main sections and subsections in it. Each document should refer to its own section. The most popular and optimal option for most sites is a tree structure. Each subsequent page is a part of the previous one. The multi-level structure allows highlighting more important pages for review.

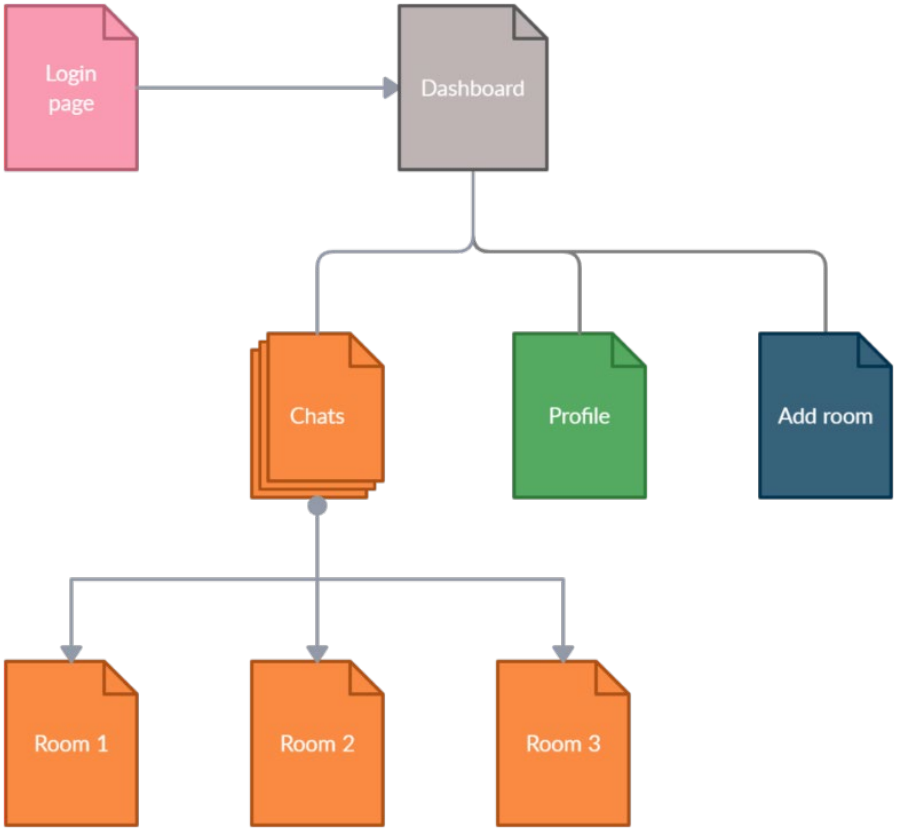

Figure 2: Website structure

The first page that the user will see is the login page for the system. The user who isn't logged in doesn't have the right to view the pages of the site. When trying to open other pages, the user will be redirected to the login page.
After authorization, the user goes to the Dashboard. This is the start page, which can contain a user's greeting, news blocks, event notifications, etc. It also contains a navigation bar to open other pages. Each user has their own personal profile page, which includes password change, and logout options. The chats page should contain a list of chat rooms that users can freely enter. If a room with the desired topic hasn't been selected, the user can create a room on the page Add room page. Each room contains a chat form, with the history of all correspondence, which is stored in the database.

4 SYSTEM DESIGN

4.1 Technical aspects

A prototype of the system will be built as a website using a **PHP7**. It is much faster than its predecessor PHP5.6. The performance test for various frameworks PHP7 removed many deprecated functions, introduced support for strong typing, added new methods, operators, Unicode encoding syntax, group use declaration, many new syntax features, improved exception handling, anonymous functions.

The **PhpStorm** IDE is ideal for working with such frameworks as Laravel, Symfony, Drupal, WordPress, Zend Framework, Magento, Joomla!, CakePHP, Yii, and others. It indexes all project code, supports many languages, autocompletion, code refactoring, error highlighting, built-in tools for working with SFTP, console, databases, etc. An important aspect is that a full version of **PhpStorm** is freely available for students, teachers and for open-source projects.

To simplify the development, the PHP framework **Laravel** was chosen for the back-end. It uses the **MVC** model, which is a project design pattern that divides the application into data model, presentation information, and control information parts. The framework supports many different database management services, Composer, and a number of other libraries are supplied as part of the framework. All source code of the framework conforms to PSR standards, which serves the standardization of programming concepts in PHP. Laravel has its own development host, detailed functionality documentation with detailed examples and supports PHP7.

For the front-end development **Vue.js** framework was chosen. The progressive and high flexible tool, one of the most popular **JavaScript** frameworks today.

PostgreSQL DB is an object-relational database, that is, it contains technologies that implement an object-oriented approach. It has some advantages over MySQL:

- Subqueries can be written separately by giving them names.
- A large number of different types of data, including the ability to add custom types.
- Constraints on the database itself to ensure data integrity. The ability to restrict by a certain range, condition, etc.
- The json type and full-fledged work with it.
- Stored procedures in any language.
- Replication is fast and consistent.
- If configured correctly, queries are executed many times faster.
- Full-text search.
- Sequences. MySQL only has AUTO_INCREMENT per table field, iterating one at a time. In PostgreSQL, this mechanism lives separately from the table, which can be used for a wide variety of needs, and you can also create them in loops.

For creating a project, the version control system (**Git**) will be actively used. Git has both the ability to control the console and a graphical shell, is integrated into many IDEs, has high performance.

For the construction of chat channels, **pusher** services and API will be used. It will facilitate the implementation of channels, as well as speed up the development process. Advantages:

- Immediately scale to billions of messages
- Industry-leading uptime
- Flexible and versatile API
- SDKs and libraries for every language

The **webserver** will be implemented on a local machine. From the moment a user accesses the website, to the moment that user leaves the website all of the data should be encrypted. An easy way to implement this is to use HTTPS instead of plain old HTTP to power the website. A supporting SSL certificate is needed for security. Also, it avoids search engines from flagging the site as “not secure”. It’s necessary to support different scaling in the website to quickly and easily scale the page with the chat into a comfortable state. For the development and testing of the project, there is no need for a large space on hard drive. However, once implemented, it will be necessary to use several terabytes as data will be accumulated daily. In order for the system not to become clogged, it will be necessary to clean the rooms in a timely manner. An annual cleansing will be more than enough.

Since the application gathers users’ data, the system is obliged to protect personal data from unauthorized or illegal processing, destruction and damage, and comply with all standards of GDPR.

4.2 Deployment diagram

In order to better describe all the important processes and their integration into the system, we use the Deployment UML diagram. Deployment diagrams help model the hardware topology of a system compared to other UML diagram types, which mostly outline the logical components of a system. Since we plan to use a third-party service to support real-time sending of messages, the deployment diagram is the best choice to describe this process.

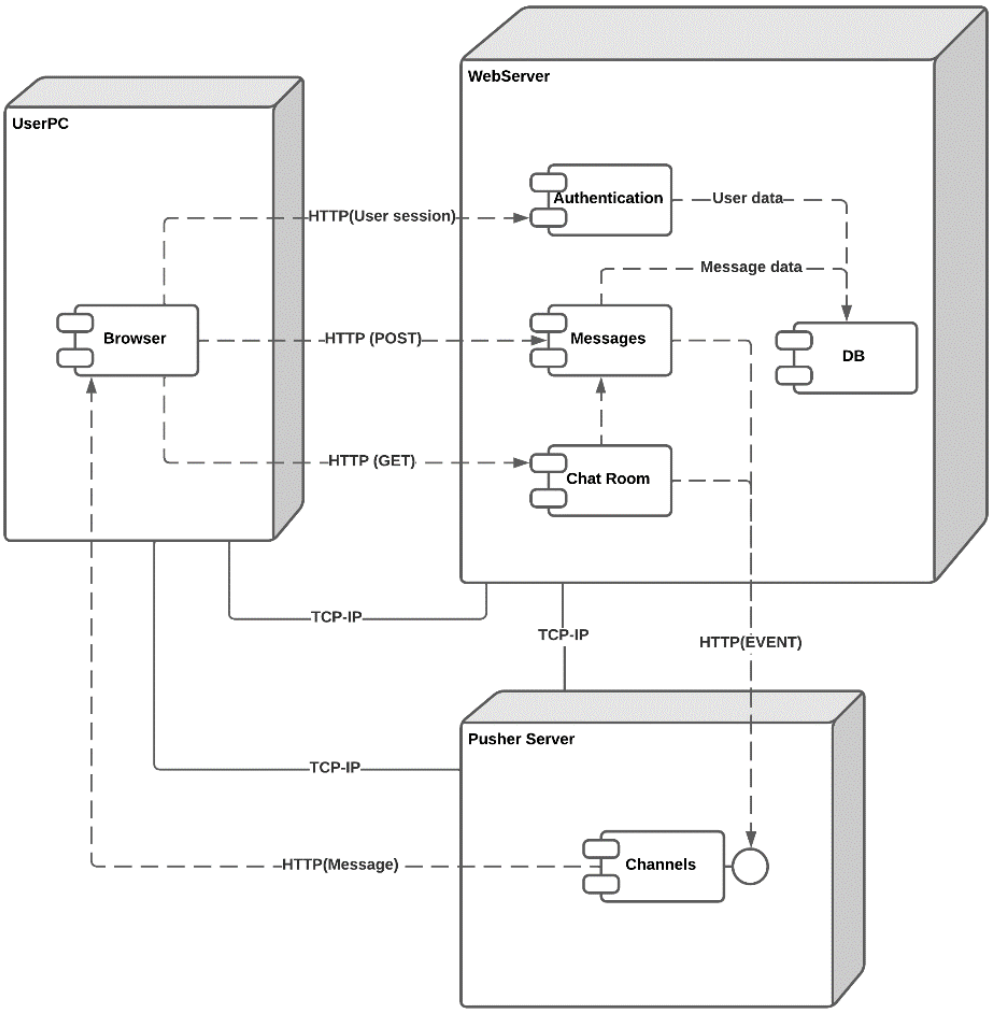

Figure 3: Deployment diagram

We have identified 3 necessary nodes for the deployment diagram: user pc, web server, and pusher. Since all interaction between them takes place over the Internet, we connect nodes with a TCP-IP protocol. The pusher works through a system of channels and receives events from the server. Events can be of 3 types: subscribe, unsubscribe, or broadcast. For any user action in the chat, the server sends the corresponding event to the pusher server.

For a better understanding, we will consider the standard case. If any user action is performed on the site, the user's session is checked by the authentication component. If the session is verified, the system allows the action to be performed. The user enters the chat room, the session successfully verifies, his messages are loaded, and the server sends an event subscribed to the pusher. Then he sends a message to the chat, where it is processed by the post method, the database is updated, and the server sends a broadcast event with the user's message. The Pusher broadcasts this message to all channel subscribers. After that, the user leaves the chat, and the server sends an unsubscribe event.

4.3 MVC

To create the website, it was decided to use the MVC model, which is the most popular model today. The **Model-View-Controller (MVC)** framework is an architectural pattern that separates an application into three main logical components: Model, View, and Controller. Each architecture component is built to handle specific development aspects of an application. MVC separates the logic and presentation layers from each other.

A model is a representation of an object and business logic in our database. It responds to view requests and updates in response to controller instructions. It is also the lowest level of templates responsible for maintaining data. The View represents the interface through which the user interacts with our application. Views are created based on data collected from model data. It prompts the model to provide information to send the output view to the user. The controller is a part of the application that handles user interaction. The controller sends commands to the model to update its state and sends commands to its associated view to change the page content.

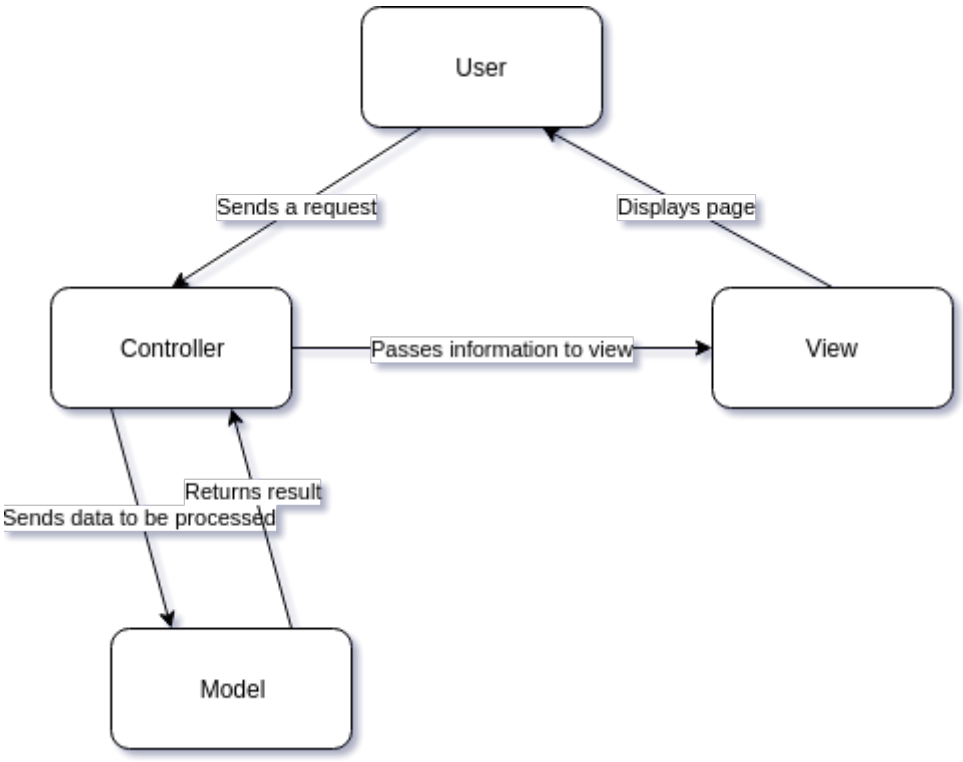

Figure 4: MVC [1]

Figure 4: MVC [1] demonstrates the processing of a request using the MVC model. After receiving an HTTP request, the server accesses the route component, which contains a list of all routes and determines which controller should be called, after which it sends a request to the desired controller. The controller accepts the request and calls the appropriate method. The called method determines the data of the model based on the business logic and calls the

required view. After the view is created, the controller sends the model data in the form of attributes to the view, which is eventually displayed in the browser.

Advantages of using MVC

1. Easy code maintenance, which is easy to extend and grow.
2. Using the MVC allows easy modification of the entire application. Adding and updating the new types of views is simplified in the MVC pattern as a single section is independent of the other sections. Any changes in a certain section of the application will never affect the entire structure.
3. Development of the various components can be performed in parallel.
4. As there is segregation of the code among the three levels, developing web applications using the MVC model allows one developer to work on a particular section while another can work on any other section simultaneously. It works well for web apps that are supported by large teams of web designers and developers.
5. All classes and objects are independent of each other, so it is possible to test them separately.

Disadvantages of using MVC

1. Increased complexity and inefficiency of data.
2. There is a need for multiple programmers to conduct parallel programming for a big project. However, one person is enough to implement the prototype of our project, since its functionality is limited.
3. Knowledge of multiple technologies is required.
4. Maintenance of lots of codes in the controller.
5. Not suitable for small applications. It has an adverse effect on the application's performance.

In general, the use of MVC increases the amount of manual coding, however, it allows to cover most of the logic with tests, reduces the connectivity between components, and can serve as a basis for implementing a template mechanism. Also, there are a large number of web frameworks using the MVC pattern: Ruby on Rails, Django, CakePHP, Yii, CherryPy, Spring MVC, Catalyst, Rails, Zend Framework, CodeIgniter, Laravel, Fuel PHP, Symphony. Using frameworks with this model will speed up the development process and make it easier to write code.

4.4 Database design

Database design facilitates the design, implementation, and maintenance of a data management system. A properly designed database is easy to maintain, improves data consistency, and is economical in terms of storage space. During the design of the database, it is decided how the data items are related and what data should be stored.

The main objective of database design is to produce a logical design model of the proposed database system. The logical model concentrates on the data requirements and the data to be

stored independent of physical considerations. It does not concern itself with how the data will be stored or where it will be stored physically.

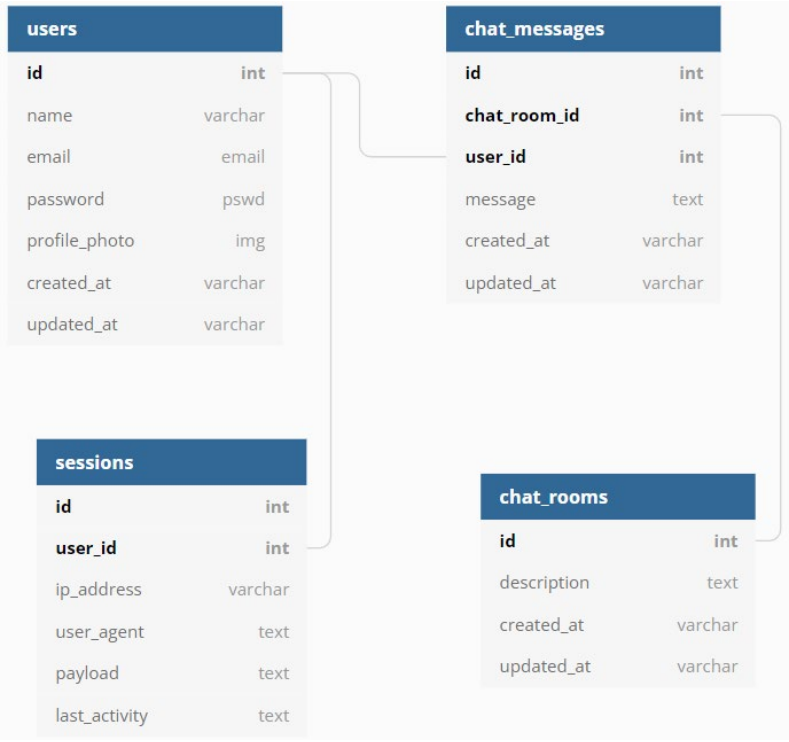

Figure 5: Database design

In the projected design, tables were created and separated, and the relationship between them was built. These are the most important tables that will contain all the relevant data.

In the table of users, we define their personal data and login information, which is essential for user authentication. The chat rooms table contains information about rooms and their unique id, it is used in chat_messages table. This table contains all messages written by users and reference keys to other tables. It is needed to understand in which room and which user was written the message. The table session is needed to identify users on the site. It contains information about the user's session and the reference key for their table. Each user must have a unique session.

Such a design will allow us to efficiently use and store data, save storage space, and simplify the development of the system.

4.5 Selection of back-end framework

To start developing a prototype, we need to decide which development tools will be most helpful. Since, we have a huge number of technologies, it is necessary to understand which tool will allow us to create an application faster and easily.

Web frameworks (or "web application frameworks") are software tools that make it easy to create, maintain, and scale web applications. Frameworks provide libraries that simplify common web development tasks, including routing URLs to appropriate handlers, interacting

with databases, maintaining sessions and authorizing users, formatting the output (e.g. HTML, JSON, XML), and improving web security.

Numerous web frameworks exist for almost every programming language we might want to use. With so many options, it can be difficult to determine which framework provides the best starting point for the project. The website [2] "Statistics and data" contains statistics on the use of various back-end frameworks from 2012 to 2021. These are indicative statistics for us, since it is important to know what tools are in demand and what modern development standards they set.

Based on their data, it can be seen that the Laravel framework has been in the lead for several years since February 2018.

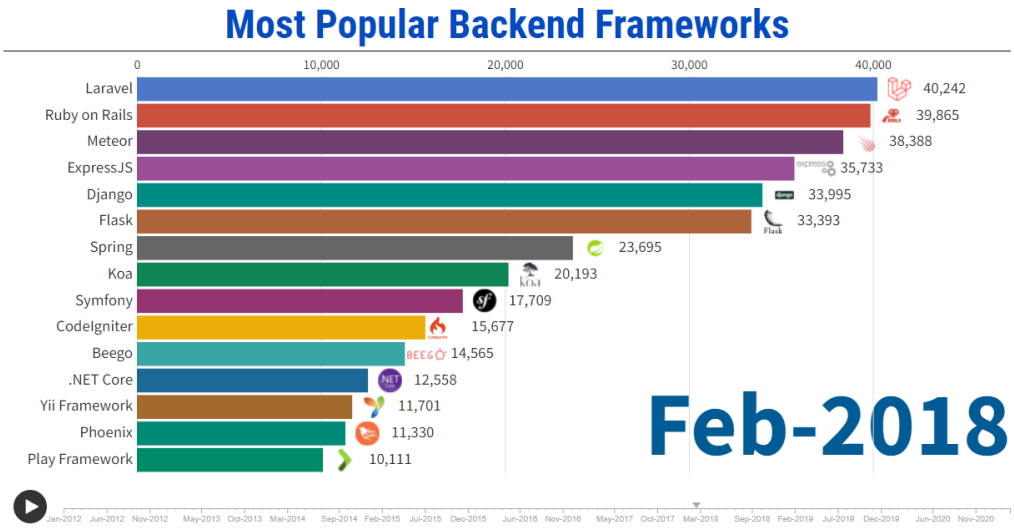

Figure 6: Back-end frameworks 2018 according to [2]

Now, in 2021, the lead of Laravel is even wider, and the number of sites developed using it has increased by half.

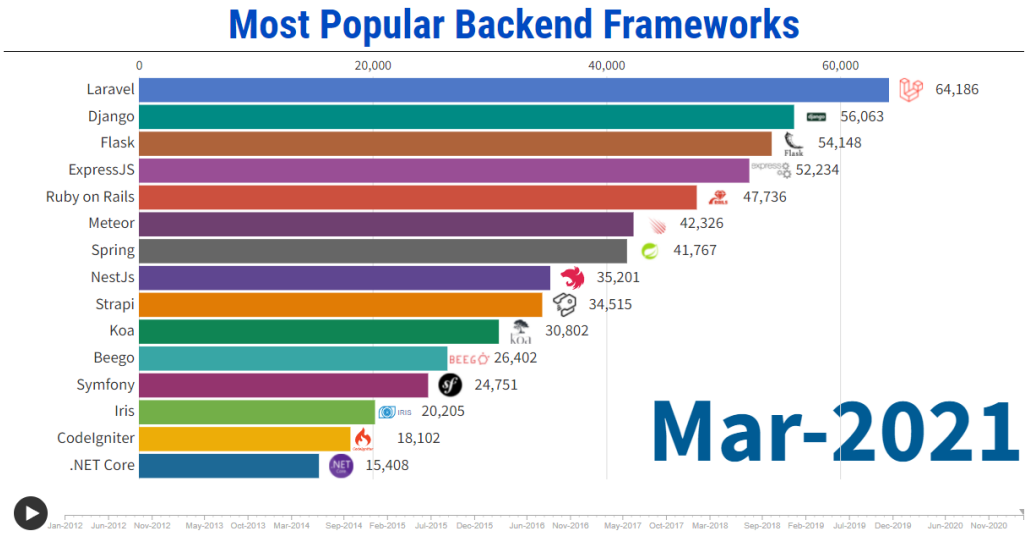

Figure 7: Back-end frameworks 2021 according to [2]

Based on this data, we can say that it is the most popular back-end framework today. In order to understand Laravel better, it is necessary to consider all its positive and negative sides and compare it with other frameworks.

Laravel is a unique PHP framework developed by Taylor Otwell for creating web applications using the Model-View-Controller pattern. To create Laravel, the main components of another well-known framework, Symfony, were used. In the second version of Symfony, the creator made the constituent parts of the framework that were independent of each other. This change permitted anyone to disassemble it safely into components and assemble anything from them. Laravel was created from such components. The original idea of the creation was to develop an alternative to Codeigniter. This framework did not have all the necessary features to build web applications.

4.5.1 Advantages of Laravel

1. **High security.** There are two main security concerns: SQL injection and cross-site scripting. The framework protects against the first ORM which excludes the possibility of SQL queries and normalizes all parameters when building them. Anything that can harm the data is removed from them. The second is solved by escaping forbidden html tags and outputting the escaped string as plain text without the possibility of its execution.
2. **Caching.** By default, caching is enabled at the file system level. Developers can change this behavior and use non-SQL databases: REDIS, Memcache, or APC. They store data in the form of key-value pairs and do so in the server's RAM. Due to this, the data access time is reduced and enables developers to cache any amount of data.
3. **Authentication.** Laravel has support authorization tools like LDAP API and libraries like OAuth. They implement a mechanism for authorizing users with a few commands and it is available immediately after the project’s deployment.
4. **Blade templating engine.** It allows to standardize and reuse the same template across different parts of the application. However, the template engine doesn’t restrict the developer from using native PHP templating techniques. The Blade templating engine is fairly lightweight and doesn’t reduce web page generation performance.
5. **Database migrations.** The developer can easily change the data structure and roll back changes in case of an error. This comes in handy in group development, where all participants in the process can especially update the local database with the new structure. Also, developers can populate the databases with test data. All this eliminates the situation when a team member makes changes to the database and breaks all the applications of colleagues because their database does not contain what the code is working with.
6. **Unit tests.** When developing projects of medium to high complexity, manual testing takes a lot of time. Each new feature is potentially fraught with bugs, sometimes in the most unexpected places. Units, or unit tests, allow testing of software components within a system. In these tests, the developer connects the component under test, function, class, and passes necessary for process input data into it, and then receives and analyzes the output data for adequacy**.** Laravel has PHPUnit tests built-in out of

the box. For each application, settings are created in the phpunit.xml file. In addition to unit tests, there are also functional tests, where the system is tested by the user, emulating all his actions. Such tests are conducted with the participation of the browser.

7. **Integration with mail service.** Laravel provides a simple API on top of the SwiftMailer library. It also has drivers for SMTP, Mailgun, Mandrill, SparkPost, Amazon SES, PHP "mail" and "sendmail" functions that allow the application to send mail locally or via cloud services. In addition, Laravel provides support for sending notifications through many other delivery channels, including SMS and Slack.
8. **Dealing with Errors and Exceptions.** When creating a new application, Laravel already has configured error and exception handlers and in addition, an integrated logging library Monolog which contains many handlers. The way the developer handles errors and exceptions can have a big impact on the usability of the application. The most common source of errors is user forms. The system determines the location of the error and notifies the user about it.
9. **Documentation.** It is believed that Laravel is developer-friendly due to documentation. Every version of Laravel is shipped with the proper documentation in which the developer is able to find good and detailed explanations of classes, coding styles, methods, etc.

4.5.2 Disadvantages of Laravel

1. **Upgrades are problematic.** This is not solely a Laravel problem, but PHP frameworks do show problems for long-term support versions as the upgrades may turn problematic. So, the developers are advised to take precautions before upgrading a mobile application/website.
2. **The Composer is not strong enough.** As Laravel is a new framework, therefore it is difficult for developers to deal with it. Also, the composer of Laravel is not strong enough in comparison with npm (node.js), pip (for python), etc.
3. **Lack of Inbuilt Support.** When compared to other frameworks such as Ruby on Rails and Django, Laravel has limited inbuilt support due to its lightweight. This issue can be resolved with third-party tools.
4. **Complex.** The framework has a large base of various libraries, methods, and configurations. The developer should have some experience to get started with building an application. However, thanks to Laravel's support and the huge online developer community, these initial steps can be painless.

4.5.3 Comparisons

To identify other limitations of Laravel, it is necessary to compare it with another popular framework.

The second most popular framework today is **Django**. It is a python-based web framework. It was released in 2005, and is developed and maintained by the Django Software Foundation. Django works on the MVT architecture. It's highly adaptable to different types of projects in multiple industries and involves several pre-made feature packages. Django is an excellent tool for creating complex applications since Python's performance is higher than PHP's.

Table 1: Comparisons of Django and Laravel

Differences	Django	Laravel
Speed	Python is a high-performance language that provides swift execution. This makes Django naturally faster as it allows developers to speed up web applications without taking up much time and effort on their part.	Laravel has rich and robust features but these additional features make it slower than other PHP-based web frameworks. Many developers have to find other ways to improve the speed of the web applications they developed. However, PHP has great improvements in version 7 when it comes to performance (as benchmarked), though it doesn't follow that the framework is also fast.
Routing	Routing in Django is much harder because it has no built-in Application Programming Interface (API) so you have to use a library to work around it and implement the routing.	Laravel provides an easier way to create routes which are to call the method you want to use such as to get, post, patch, and delete. Then supply the route URL and the view or controller method.
Admin panel	Django has an easy-to-use admin panel that can be activated with just one command. The admin panel allows you to create a superuser and register the models which you want to manipulate through the admin.	Laravel doesn't come with an admin panel so you have to use a third-party library or create your own library from scratch.
Templating	Django has the Jinja template engine that allows customization of globals, filters, tags, and tests. Jinja allows the template designer to call functions with arguments on objects.	Laravel comes with Blade – a powerful and fully featured template engine.

Security:	Django's user authentication system offers a safe way to manage user accounts and passwords and ensures the developers stay away from making common errors such as cross-site scripting, cross-site request, clickjacking, and forgery.	Laravel uses hashed and salted passwords which means that the password would never be saved as plain text in a database. It also uses the "Bcrypt Hashing Algorithm" to create an encrypted representation of a password.

After analyzing the main differences between these two frameworks, we can say that Laravel has no significant drawbacks compared to Django. They are really two completely different tools, written in different languages, using different packages and different methods. The main advantages of Django are the speed of its work and a ready-made admin panel from the package. However, the performance of the site still depends more on its configuration and there are many methods in Laravel to improve this. So, we can say that both frameworks put themselves in the best light and meet all the most important standards today.

4.5.4 Conclusion

The best web framework for back-end development is the one that assists the developers to have a good start by eradicating the requirement to develop and configure everything from scratch. Laravel is one of the best tools on the web framework market today. It has huge functionality, good documentation, various libraries, and a large community.

4.6 Selection of front-end framework

User interface development is also a big and time-consuming job. In order to speed up the development of our project, it is necessary to use modern solutions in the development market - front-end framework. This type of framework includes certain development instruments, such as a grid making it simple to place and position the UI design components, pre-defined font settings, and website standard building blocks (i.e., side panels, buttons, navigation bars, etc.). There are a bunch of front-end frameworks on the web, and most of them run on JavaScript as their source language. The website [3] "Statistics and data" contains statistics on the use of various fronted frameworks from 2012 to 2020.

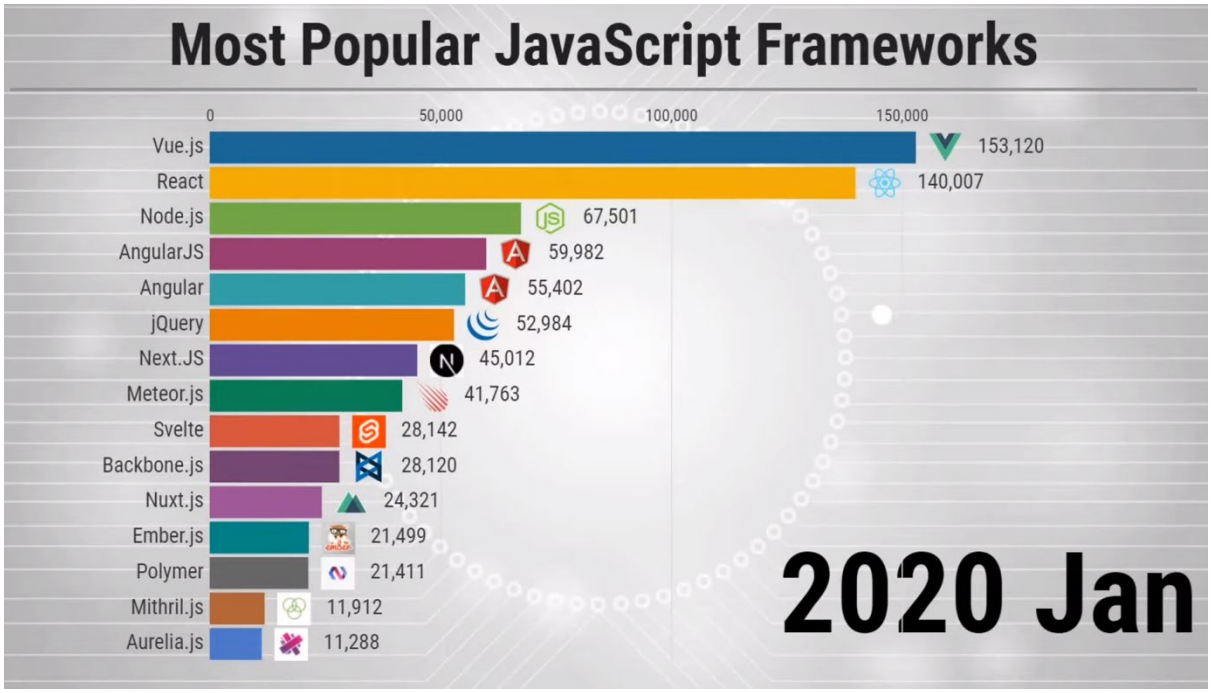

Figure 8: JavaScript Frameworks according to [3]

According to statistics, we can highlight frameworks like Vue.js, React, Node.js, Angular. These tools are very popular now and are used in the development of many web applications. To choose the right front-end framework, it is important to consider how it works in conjunction with our back-end framework.

4.6.1 Angular

Angular is a single framework that is based on TypeScript. Formally released in 2016, Angular was established by Google to link the gap between the mounting demands of technology and conventional notions that displayed the results.

Table 2: Pros and cons of Angular

Pros	Cons
Making the coding procedure easier due to its refactoring services and enhanced navigation	The learning effort
The component-based pattern of Angular sanctions forms a user interface with single components	The CLI documentation is not fairly defined
Angular Material reorganizes Material Design interface production	Angular complication

One of the top reasons to use Laravel with Angular is – they both have similar frameworks with slightly different purposes. They both use an MVC-inspired application structure and templating engine. However, this leads to certain conflicts between both frameworks. In a

project can be some hurdles – Laravel Blade and Angular use the same {{}} brackets to define variables. This causes conflict and creates confusion when coding to build an application.

4.6.2 React

React is the open-source framework developed and created by Facebook. This framework is one of the best UI frameworks today used by a majority of the front-end developers according to Stack Overflow Developer's survey 2021.

Table 3: Pros and cons of React

Pros	Cons
Virtual DOM. Instead of rewriting the entire DOM tree every time a change is made by a user, a virtual DOM only updates the element being manipulated.	As a result of rapid development, educational documentation and resources can be sparse in covering the latest updates and changes.
One-way data flow. Parent data cannot be affected by changes in child elements. This creates a more stable code and gives developers more control over large and complex projects.	The pace of ReactJS's development is a common disadvantage that is brought up, but it should be noted that in recent years React's core API has gotten a lot more stable.
Saving time using reusable components.	The primary complaint about JSX is its difficulty. JSX doesn't negatively impact the performance or User Interface capabilities of ReactJS. This con is more of a preference than anything else.
An open-source library with a diversity of tools.	

When using Laravel with React, React can be used for front-end view, and Laravel can handle creating REST API endpoints. React is known for its excellent and fast front-end performance, and it is highly dynamic and loads on demand. But there are some difficulties that arise from using Laravel and React together.

React is great for small-scale projects for JavaScript experts. However, it creates issues when scaling. React can be difficult to comprehend for people coming from a non-JavaScript background. With React, library dependencies increase which can complicate the entire project. Moreover, React projects can be time-consuming to set up.

4.6.3 Vue.js

Vue is an open-source model-view-view Model (MVVM) front-end JavaScript framework. It is used for building single-page applications and user interfaces. Vue is one of the very few

JavaScript frameworks that comes with compressive documentation. The entire documentation is easy to understand. As long as the developer is comfortable with basic HTML and JavaScript, they can develop apps fluently on Vue. The framework is not all that different from Angular and React as well, it does share some common principles with them.

Table 4: Pros and cons of Vue

Pros	Cons
Lightweight. Vue is convenient to use for developing projects of various complexity - both small personal projects and multifunctional applications.	Vue is still very new and evolving fast. It doesn't yet have the widespread support of its fellow frameworks, as it is not as popular as React or Angular.
The Vue framework uses HTML to render objects as well as a template system to facilitate its integration with existing applications. Compared to its competitors, Vue is a flexible framework, as it allows you to write templates at your own discretion using HTML, JSX, or JS.	While the ecosystem is pretty wide, and there are all the required tools to start developing with Vue, it's still not big. It has a limited number of plugins and packages.
Vue handles two-way reactive data-binding like a charm. When data is changed, the DOM has also changed accordingly.	
Vue allows for easy code reusability through the components	

Laravel and Vue are the most popular pair, that used by developers today. Frameworks are based on different programming languages, which confuses many people as to how both of them would support each other. However, they support each other in more ways than one. Laravel with Vue – Reactive components help to make the best event-driven apps.

Vue has the capability to create a full-scale event-driven web app that can handle all front-end activities. It also makes use of composable components that can be used as per the developers' liking. Since Laravel couples well with Vue, developers only need to make a few trips to request data from the Laravel application and make changes to the UI by switching components without reloading the page. With Vue, we can prompt UI changes on the front-end. This provides an immersive user experience for users. We can make a content piece on our page editable or swap out an entire component to load a video as requested by a user without reloading the entire page.

Laravel with Vue – ideal for creating complex front-end pages.

4.6.4 Conclusion

While we can take Angular and React as the front-end development tools to use with Laravel, we will face some of the other challenges since they aren't built to be perfectly compatible with Laravel. On the other hand, Vue is easier to learn and has some promising features that complement the requirements of Laravel as well. Packed together, Laravel and Vue have

immense potential to create good web applications and website solutions. Therefore, the best choice for our project would be to choose Vue.js.

5 IMPLEMENTATION

In this section, we will look at the finished site prototype and its code. But before that, implementation steps need to be considered too. To do this, we will examine all the main steps that were taken to create the prototype: how the environment was set up, the project configuration, discuss the structure of the Laravel project, and note important points of development. After that, we can review the implementation of the prototype, to determine how successful the project was and point out all the shortcomings.

5.1 Setting up the environment and first configuration

Before the code implementation, we need to prepare the environment of the system. Download and install the required software, IDE, project files, database, set up the project's configuration file, etc. To simplify this process, we will discuss it step by step:

- The first step involves installing the programming language of the environment. The PHP version 7.4.22 was downloaded and installed from the official site. This is one of the latest and most stable versions at the moment.
- Next, we installed the PostgreSQL database package and an additional tool - pgAdmin4, for settings and subsequent work in the database.
- In the pgAdmin, we can immediately create a database for our project, create a user with a username and password to connect to the database.
- On the Laravel site, various ways of installing the project are provided. Setting up via composer is one of the easiest. To do this, we need to install the composer itself. After the installation, use the command: "*composer create-project laravel/laravel chat-Application*". The composer creates a project folder in the current directory, where it will download all the project files.
- For further configuration, we can open the project in our IDE – PHPStorm. We can download it from the website [4].
- Then we can immediately bind our database to the project. This is done in the configuration file .env which is located at the root of the project.
- In order to install vue.js, we need to run the following command: *"composer require laravel/ui"* to get the package and install it: *"php artisan ui vue"*.
- For the vue.js to work correctly, we need to install also npm: *"npm install && npm run dev"*.
- For the convenience of development, we will use a version control system. Therefore, we create a project on Gitlab [5] and make an initial commit of our project. This is necessary so that in the event of unexpected errors we can always roll back to the previous version of the application.

After the steps taken, we have a working environment for developing a website. Now we can create a development branch in the git in order to back up the project in case of unexpected errors during development. But, before we start writing the code, we need to understand the

structure of the Laravel project. Since it is very complex, it has many different directories and it is not entirely clear how to work in them.

5.2 Laravel structure

The application structure in Laravel is basically the structure of folders, sub-folders, and files included in a project. Since we created a project, we got the following structure:

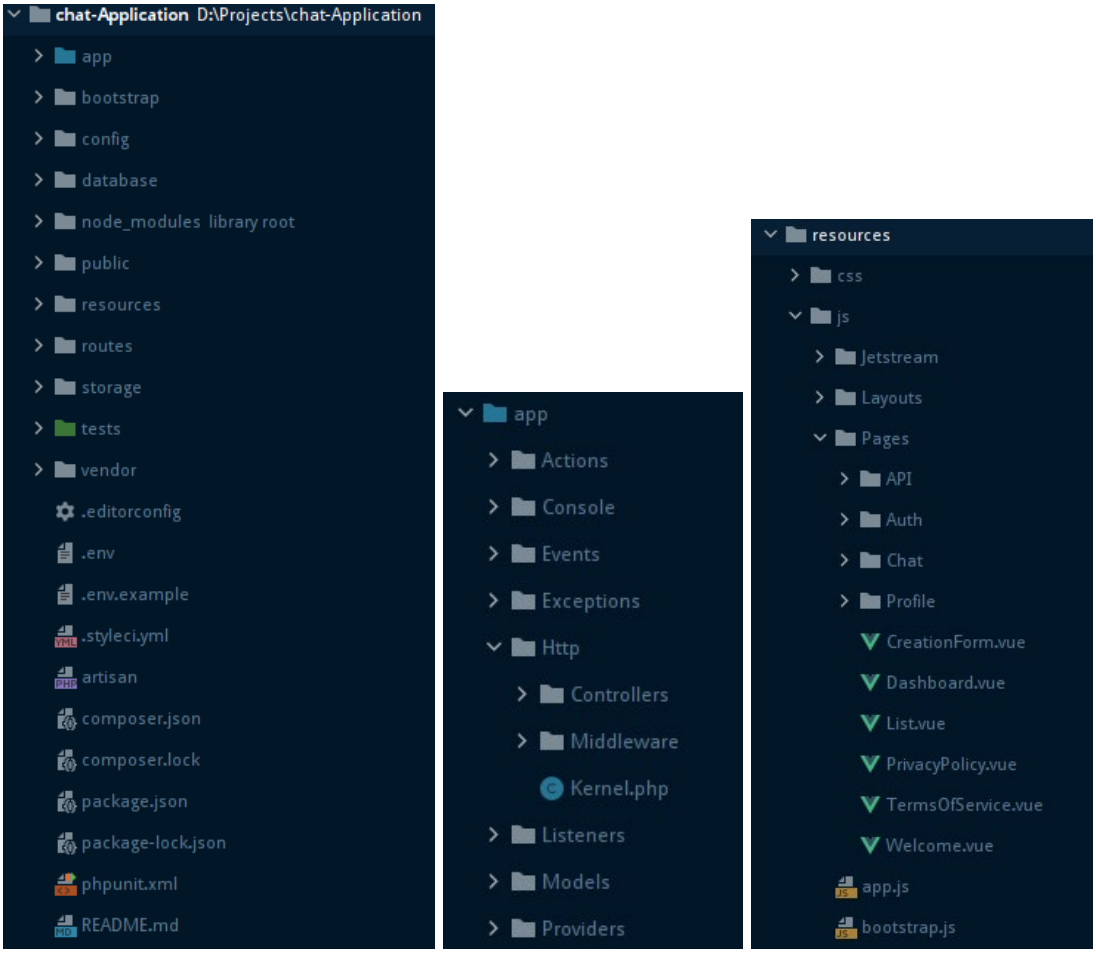

Figure 9: Laravel structure

The images that are shown here refer to the root folder of the project. To implement a project, we need to have a good understanding of its structure. Using the documentation [6], it is necessary to parse the directories and determine the most important things that we need for implementation.

App

It is the application folder and includes the entire source code of the project. It contains events, exceptions, and middleware declarations. The Http folder has sub-folders for controllers, middleware, and application requests. As Laravel follows the MVC design

pattern, this folder includes model, controllers, and views defined for the specific directories. The Middleware sub-folder includes a middleware mechanism, consisting of the filter mechanism and communication between response and request. Events directory is used to trigger activities, raise errors or necessary validations, and provide greater flexibility. Here, we will work with controllers, define our models and create some events.

Databases

This directory includes various parameters for database functionalities. It includes three sub-directories:

- Seeds - contains the classes used for the unit testing database.
- Migrations - helps in queries for migrating the database used in the web application.
- Factories - is used to generate a large number of data records.

In this directory we will create and edit our migrations for the database.

Public

It is the root folder that helps in initializing the Laravel application. It includes the index.php file, configurations, and assets of the framework.

Resources

The resources directory contains our views as well as raw, un-compiled assets such as CSS or JavaScript. This directory also houses all of the language files. Here, we will work with vue.js components.

Routes

The routes directory contains all of the route definitions for the application. By default, several route files are included with Laravel: web.php, api.php, console.php, and channels.php. The web.php file contains routes that the RouteServiceProvider places in the web middleware group, which provides session state, CSRF (Cross-site request forgery) protection, and cookie encryption. If the application does not offer a stateless, RESTful API then it is likely that all of the routes will most likely be defined in the web.php file. The channels.php file is where we may register the entire event broadcasting channels that the application supports.

Storage

The storage directory contains logs, compiled Blade templates, file-based sessions, file caches, and other files generated by the framework. This directory is segregated into app, framework, and logs directories. The app directory may be used to store any files generated by the application.

Tests

The tests directory contains automated tests. Example PHPUnit unit tests and feature tests are provided out of the box.

Vendor

Laravel is completely based on Composer dependencies, for example, to install Laravel setup or to include third-party libraries, etc. The Vendor folder includes all the composer dependencies.

Artisan.php

Artisan is the command-line interface included with Laravel. Artisan exists at the root of the application as the artisan script and provides a number of helpful commands that can assist us while we build the project. To view a list of all available Artisan commands, we may use the command: *"php artisan list"* in the terminal.

Despite the fact that the structure of the Laravel project is quite large, all the elements of the structure are arranged logically. This allows to quickly adapt and also make fewer mistakes. Knowing in which files to write the corresponding code, we can start the implementation of the website.

5.3 Implementation of the code

After installing and configuring the project, we can start it by writing the command: *"php artisan serve"*. Then Laravel starts a local server on port 8000, and we can open our project in the browser. At this stage, there is only a start page with a greeting to Laravel and nothing else.

5.3.1 Laravel Jetstream

To speed up the development of the interface, we connect the Laravel Jetstream. It is a beautifully designed application starter kit for Laravel that provides the perfect starting point for Laravel application. Jetstream provides the implementation for application's login, registration, email verification, two-factor authentication, session management, API via Laravel Sanctum, and optional team management features.

The Inertia stack provided by Jetstream uses Vue.js as its templating language. Building an Inertia application is a lot like building a typical Vue application. However, we will use Laravel's router instead of Vue router. Inertia is a small library that allows rendering single-file Vue components from Laravel back-end by providing the name of the component and the data that should be hydrated into that component's "props".

In other words, this stack gives the full power of Vue.js without the complexity of client-side routing. You get to use the standard Laravel routing and view data hydration approaches that you are used to.

Using the documentation [7], we need to install the Jetstream. We add it to the project by command: *"composer require laravel/jetstream"* and install by artisan: *"php artisan jetstream:install inertia"*.

After installation, Jetstream automatically loads authentication and registration migrations into the project. It also creates additional tables that are required for sessions, tokens, etc. This is very convenient since we don't have to create it, just supplement them. In order to migrate to the database, we need to write the command: *"php artisan migrate"*.

```
Migrating: 2014_10_12_000000_create_users_table
Migrated:  2014_10_12_000000_create_users_table (64.64ms)
Migrating: 2014_10_12_100000_create_password_resets_table
Migrated:  2014_10_12_100000_create_password_resets_table (37.99ms)
Migrating: 2014_10_12_200000_add_two_factor_columns_to_users_table
Migrated:  2014_10_12_200000_add_two_factor_columns_to_users_table (82.15ms)
Migrating: 2019_08_19_000000_create_failed_jobs_table
Migrated:  2019_08_19_000000_create_failed_jobs_table (54.97ms)
Migrating: 2019_12_14_000001_create_personal_access_tokens_table
Migrated:  2019_12_14_000001_create_personal_access_tokens_table (58.10ms)
Migrating: 2020_11_27_141435_create_sessions_table
Migrated:  2020_11_27_141435_create_sessions_table (125.95ms)
```

Figure 10: Migrations

After the performed operations, Jetstream creates a ready-made interface for our prototype, provides its own login and registration forms, dashboard, user profile.

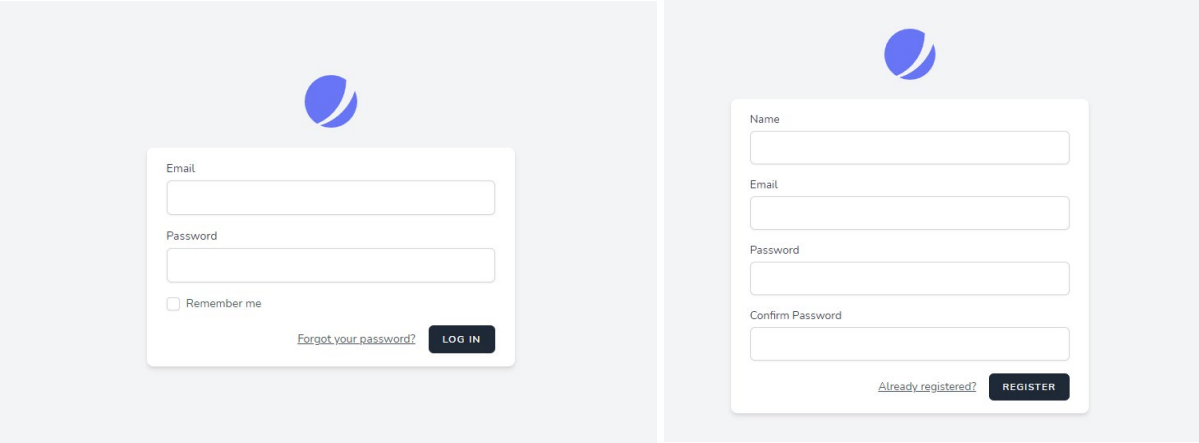

Figure 11: Login page

We do not need to create user authentication. It is already implemented in the project. All we need to register on the site. After registration, the user enters the database and our dashboard opens. So, without coding anything, authorization already works for us.

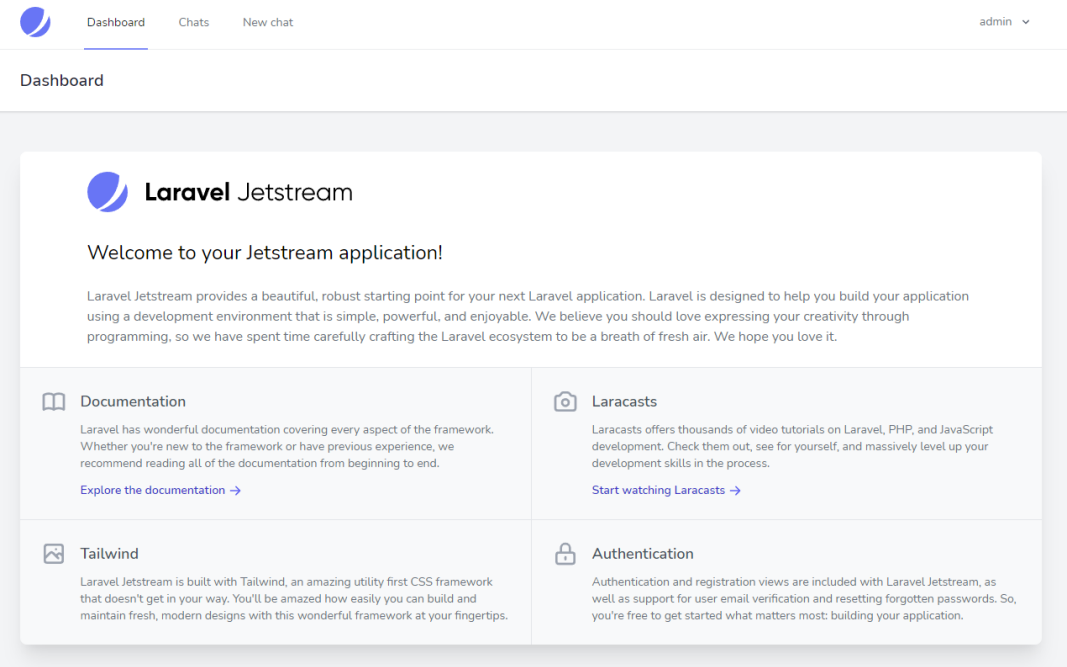

Figure 12: Dashboard

At this step, we can say that working with the framework greatly speeds up the development of a prototype. The sessions and tokens of the users are implemented, migrations are created automatically, the basic interface is also built automatically, the registration and authorization system are implemented. Jetstream also allows using its own CSS styles from the library. In just a few commands, we prepared the interface for our prototype.

5.3.2 Back-end

To create the chats system, we must prepare tables for it in the database, where rooms, user messages, their id should be stored. To do this, we create migrations using the commands: *"php artisan make:model ChatRoom"* and *"php artisan make:model ChatMessage"* and write attributes in them. In order to bind our foreign keys to the tables, we also need to create the model files and write their relations in them. Each message can have only one user who wrote it and can only be written in one room. Therefore, they have a relation *hasOne.* While a room can have a lot of messages, so the relation is *hasMany.*

Since we need to correctly display information on the site, and we want the entered user data to be saved correctly, we need to create controllers. First of all, we want to process our chat, so we create a chat controller using the command: *"php artisan make:controller ChatController".* We denote 3 functions in this file: rooms, messages, newMessage.

- Room function should return all information about the rooms.
- Messages function should return all messages for a specific room. Also, it has to be sorted by date, so that the user does not have confusion in the messages displayed on the site.
- NewMessage function should create a new record in the database with the user message. In order to determine the user id, we use the Auth function, since only authorized users can leave messages.

```php
public function rooms(Request $request) {
    return ChatRoom::all();
}

public function messages(Request $request, $roomId) {
    return ChatMessage::where('chat_room_id', $roomId)
        ->with('user')
        ->orderBy('created_at', 'DESC')
        ->get();
}

public function newMessage(Request $request, $roomId) {
    $newMessage = new ChatMessage;
    $newMessage -> user_id = Auth::id();
    $newMessage -> chat_room_id = $roomId;
    $newMessage-> message = $request -> message;
    $newMessage -> save();
    broadcast(new NewChatMessage($newMessage))->toOthers();
    return $newMessage;
}
```

Figure 13: Code example 1

All these functions are directly related to the chat and are handled by the chat controller. However, on the site, the user will also be able to create rooms and this is directly working with the table of rooms. Therefore we create a room controller: *"php artisan make:controller RoomController"*. In which we write a function for creating a new room. This controller will also be used if we want to add the function of deleting or editing rooms. However, this is not required for the basic prototype functionality.

In order for the site to open the necessary pages, we must register routes for it. To do this, we need to edit the web.php file. Here we indicate which requests the site can receive and how it should process them. The requests can be of two types: get and post. They can be processed using the controllers or redirected to other pages.

```php
Route::middleware(['auth:sanctum', 'verified'])->get( uri: '/dashboard', function () {
    return Inertia::render( component: 'Dashboard');
})->name( name: 'dashboard');

Route::middleware( middleware: 'auth:sanctum')->get( uri: 'chat/rooms', [ChatController::class, 'rooms']);
Route::middleware( middleware: 'auth:sanctum')->get( uri: 'chat/room/{roomId}/messages',
    [ChatController::class, 'messages']);
Route::middleware( middleware: 'auth:sanctum')->post( uri: 'chat/room/{roomId}/message',
    [ChatController::class, 'newMessage']);

Route::middleware(['auth:sanctum', 'verified'])->get( uri: '/chats', function () {
    return Inertia::render( component: 'List');
})->name( name: 'list');

Route::middleware(['auth:sanctum', 'verified'])->get( uri: '/chats/{roomId}', function () {
    return Inertia::render( component: 'Chat/container');})->name( name: 'chats');

Route::middleware(['auth:sanctum', 'verified'])->get( uri: '/create', function () {
    return Inertia::render( component: 'CreationForm');
})->name( name: 'creationForm');

Route::middleware( middleware: 'auth:sanctum')->post( uri: 'chat/create/room',
    [RoomController::class, 'newRoom']);
```

Figure 14: Code example 2

We have indicated all possible pages on the site. For each route, we have to check the user's authorization and redirect to the login page otherwise. For this purpose, we use the ready-made function *auth*. In order to not create a separate path for each room, we will refer to each of them through their id in the get request. This approach will permit to process requests more efficiently and not heavily load the site.

5.3.3 Front-end

At this step, we can already create a UI for new pages, create a chat form, make a list of chat rooms, create a form for adding chat rooms, etc. For this, we will work with Vue components using the documentation [8].

To work with the components, we go to the *resources/js/* directory. There already exists a tree of Vue components with Jetstream directory, which stores its reusable components. For development, we will need some of them in order to adhere to the styles of the application and not write them again.

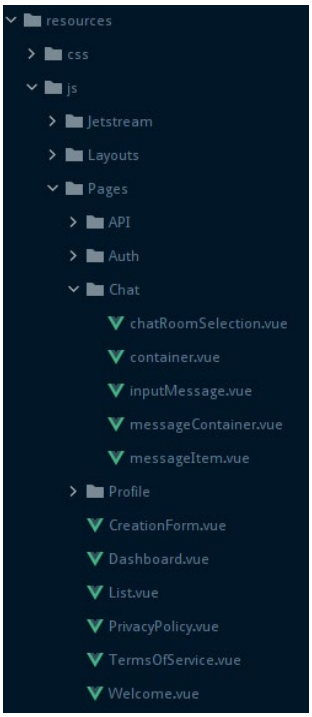

Figure 15: Vue structure

Here we create the components: List.vue and CreationForm.vue. The List component will display a page with rooms that the user can open and CreationForm is a component, which displays the room creation page with the required form.

To work with chat, we create a separate Chat directory. This directory contains the parent component - container and its children components: chatRoomSelection, inputMessage, messageContainer, messageItem. We could create a chat in one component, however, for development on a Vue it is always desirable to do the decomposition, since many components can be reused in the project, and the separation of the code helps to simplify development.

Next, we can create links to new site pages (Chats and New chat) in the header of the project, where we indicate which route and component we use.

After that, we are ready to work with List.vue component. As we already know, here we implement a list of chat rooms with links to each of them. However, before adding any content, we need to import the header, as it should be on every page of the site. It is implemented in the Applayout component that we are importing. To get the rooms we use *axios* function which uses *"/chat/rooms"* route. In the web.php file, this route is already registered and the controller is prepared for processing, so we get our data without any problems. All that remains is to bind our data in html and give it the desired style using css. Each link of the list is a route that redirects to a container with a unique room id in the argument. This id will be displayed in the room URL address and is needed to define the room.

We also implement a search function on the page. To do this, we use the java-script function - filter. The filter is constantly applied to the names of the rooms, comparing them with the value in the search bar. We use the function *"toLowerCase()"* so that the comparison takes place without a caps. Thanks to this, we have implemented a simple search by name, which is quite enough to find the right room. Since we are designing a prototype, we will not add features like filters to the page.

After implementation of the code, the rooms page looks like this:

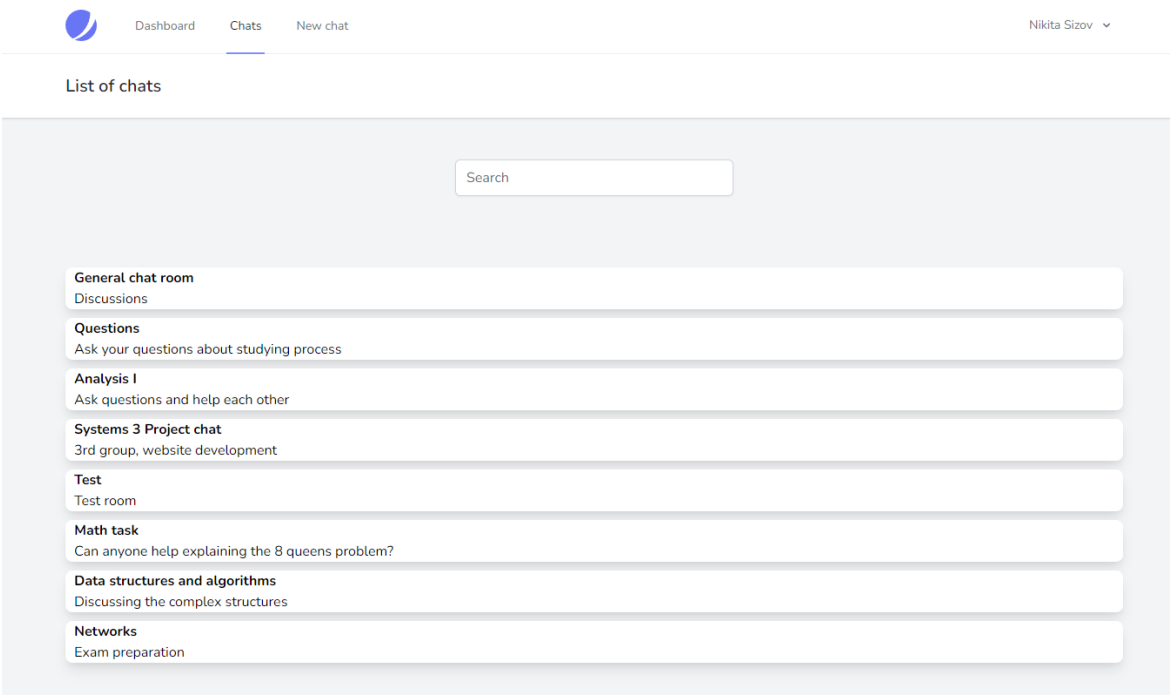

Figure 16: Rooms page

The CreationForm component contains one form that accepts data for the room (name, description) and adds it to the database. For these purposes the *axios.post* function is used with the *chat/create/room* route.

Next, we need to determine how the chat page will look. We need to develop the container component first, as this file includes all the other components. Therefore, we import the header and all our components from the chat folder. Then, we need several functions for processing the information.

```
    getRooms() {
        axios.get('/chat/rooms')
        .then(response => {
            this.chatRooms = response.data;
            this.setRoom(response.data[this.id]);
        })
        .catch( error => {
            console.log(error);
        })
    },
    setRoom(room) {
        this.currentRoom = room;
        this.getMessages();
    },
    getMessages() {
        axios.get('/chat/room/' + this.currentRoom.id + '/messages')
        .then(response => {
            this.messages = response.data;
        })
        .catch(error => {
            console.log(error);
        })
    }
},
created() {
    this.id = window.location.href.split('/').pop();
    this.getRooms();
}
```

```
<template>
    <app-layout title="Dashboard">
        <template #header>
            <h2 class="font-semibold text-xl text-gray-800 leading-tight">
                <chat-room-selection
                    v-if="currentRoom.id"
                    :rooms="chatRooms"
                    :currentRoom="currentRoom"
                    v-on:roomchanged="setRoom($event)"
                />
            </h2>
        </template>

        <div class="py-12">
            <div class="max-w-7xl mx-auto sm:px-6 lg:px-8">
                <div class="bg-white overflow-hidden shadow-xl sm:rounded-lg">
                    <message-container :messages="messages" />
                    <input-message
                        :room="currentRoom"
                        v-on:messagesent="getMessages()"/>
                </div>
            </div>
        </div>
    </app-layout>
</template>
```

Figure 17: Code example 3

In this component, we also have a list for selecting rooms, for the comfortable switching between them. So, we need the *getRooms* function again. We can copy it from the previous component. In order to receive messages that will be displayed in the chat, we need the *getMessage* function which uses */chat/room/id/message* route. Depending on the id, we will receive data from different rooms. In the html template, we build a chat frame, add components and put event listeners in order to receive values from child components.

In the messageContainer, we define how messages will be located. We can send data to the child using props. In this case, we send an array of messages, which we parse using a for loop. Since we want messages to be displayed with the username, we need to define the format of data in the messageItem.

In the inputMessage component, we create a text input field and a button. To process the message, we use the *sendMessage* function. Here we check the message and send it by the *axios.post* function with the route */chat/room/this.room.id/message.* It is very important to check for empty messages so that they don't take up space in the database.

The chatRoomSelection component implements a list of rooms. After choosing a room, its object enters the container and uses the *setRoom* function. Since we simply replace the data on the page, our address line remains the same (the URL contains the id of the original room). Therefore, the component also contains a function for changing id in the URL.

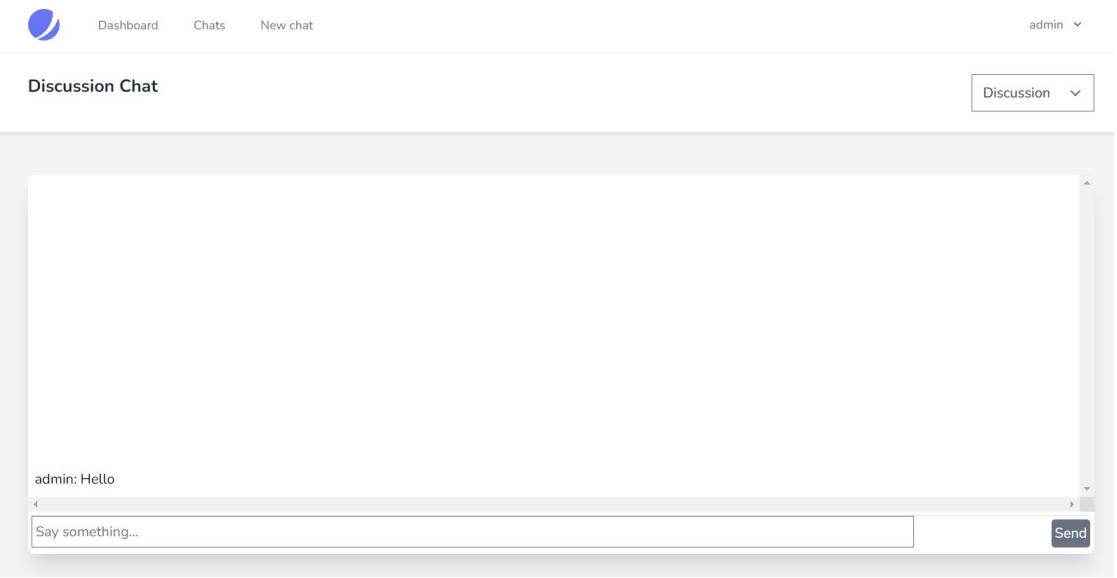

Figure 18: Chat page

In the end, we have a simple and user-friendly interface. Since the back-end is already configured, the data is transferred correctly. The user can write to the chat and see messages from other users. If he needs to go to another room, he can quickly do it through the selection button. However, while the chat application looks ready to use, there is one major problem – real-time synchronization.

5.3.4 Pusher

The problem with the project at this stage is that the user in the chat doesn't see the real-time modifications. If someone writes to the chat, then other users will be able to see his message only by refreshing the page of this chat, nothing else. To solve this kind of problem, developers use **WebSocket** protocol. The WebSocket is an independent protocol based on the TCP protocol. It enables closer interaction between the browser and the website, facilitating the distribution of interactive content and the creation of real-time applications. However, to implement the protocol, we need to build, maintain and support WebSocket infrastructure. This is not an easy task for a developer, so there is a need to consider an alternative option. One of the most popular alternatives today is a **pusher**.

Pusher is a hosted WebSocket solution that provides a simple, scalable WebSocket solution for developers. Its essence lies in the fact that the connection with the pusher channels is opened on the webserver. After the performed operation, the user entering the chat will subscribe to the channel of this chat room, and if he leaves the room, then unsubscribe from the channel. When the user wants to send a message, an event is created that sends his message to the channel using the API pusher. All users who subscribe to the channel will receive it. Therefore, without building a WebSocket infrastructure, we can configure the application for real-time work. All we need to do is connect the pusher to the project and configure it.

To connect a pusher, we need to create a pusher channels project on the official website [9]. There we can get the app keys for connecting our project. We can write them in the *env* config file. Laravel already has a built-in configuration for a pusher. To install all requirements, we need the commands: *"composer require pusher/pusher php server"* and *"npm install –save Laravel-echo pusher-js"*. Next, we just need to configure the channels. To do this, we create an event file – NewChatMessage.php, where we get the channels, the event should broadcast on. Also, in the ChatController, in the *newMessage()* function, we need to write how we do the broadcast. The internal function of Laravel *toOthers()* is suitable. Next, we need to register the event broadcasting channel in the *routes/channels* file. We define channels as *"chat.{roomId}"*.

We have determined how the message will be distributed through the channel and we only need to determine how the user will subscribe and unsubscribe to it. To do this, we need to register listeners in our chat container. We will define a connect and disconnect function with *"chat. + this.currentRoom.id"* parameters. Accordingly, if a user opens a chat, he subscribes to the channel, if he changes the chat room, then he unsubscribes from the old one and subscribes to a new one.

After the configurations have been made, we can test and debug the implementation of the pusher using the developer tools on the pusher's website. Debug console constantly listens to all channels and shows the actions carried out in the channel.

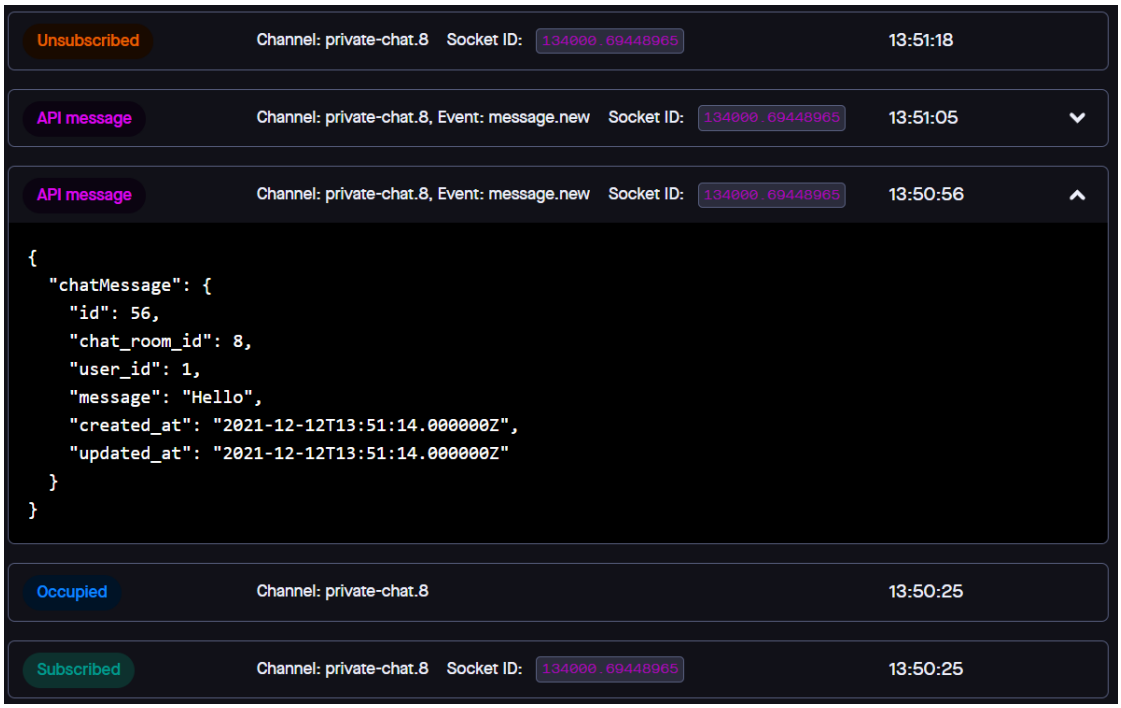

Figure 19: Pusher debug console [9]

As we can see, the user subscribed to the channel, wrote some messages, and left. So, all the events worked correctly. And now, if we subscribe from several users to one channel, they will be able to write and see messages in real-time. Therefore, we have achieved the main goal, the full-fledged work of the chat.

6 TESTING AND ANALYSIS

6.1 Testing

Now one of the main questions to ask - is whether the project is showing good performance? Using the method of observation, tests were carried out on the speed of the website. We can say that the rate of loading data on the page is within the specified requirements. The tables of delays are attached below.

Table 5: Time tests

Action	Time
Opening a chat page	Up to 5 seconds
Sending a message	Up to 3 seconds
Opening a list page	Up to 2 seconds

The table [5] shows the delay before the response from the server, without user interaction. Based on this data, we can say that the list page is loading absolutely normally. It never crossed the 2-second threshold, which is a good result. The operation of sending a message is already showing a worse result. Since users will be constantly dealing with the chat, it is necessary for it to work quickly and process sending up to 1 second. This delay is not constant and it fluctuates from the norm, up to 3 seconds. The longest delay in opening a chat page. It consistently spends about 5 seconds, which is quite long, but not very critical. It can be associated with a pusher since it needs to subscribe to the channel, and a loading data from the database. This can take a lot of time.

It is also worth considering whether it is advisable to use the pusher in the project further. The fact is that its use is well suited for a prototype and visual demonstration. But if we launch the project itself together with the pusher, then we may need a lot more resources. Pusher provides only limited resources to work for free, it is processing 200k messages per day and 100 parallel connections with the server. Although, these data meet our non-functional requirements, if we need more resources, then we have to pay for it. Therefore, it is necessary to decide on these parameters and conduct tests.

The important requirements were tested:

- When user opens any page of the site, the login page opens. If user enter incorrect data or empty fields, the system shows error message. So, we can say that the authorization works correctly.
- When user creates a chat room, the system notifies about the successful creation. If one of the required fields has not been filled in, the room is not created and user didn’t get any message.
- Each chat room has a unique id, which is needed for a get request. All chat rooms can be opened by user, and messages of other users successfully load from the database.
- To send message, the user can use the send button or use enter key. If text field is empty, then message can’t be sent.

6.2 Analysis

In order to sum up the results, about how successful and promising the project is, we need to analyze its implementation. At this step, we can say that most of the functional and non-functional requirements were implemented. Users can log in to the website, create chat rooms, make a search of the room, and have a chat in real time. Requests are processed by the server quite quickly, and the system supports parallel connections. The user interface is made friendly. All text is readable, and all elements on the site are arranged according to the standard layout that users are used to. Private rooms have not been implemented, but the prototype can be easily demonstrated without this feature. The project takes up about 100 MB of space in the storage. That's quite a lot when talking about a chat application. However, it should also be considered that we didn’t optimize the project in any way and didn’t remove unused components. So, it can take up less space.

Thanks to a variety of tools, we were able to build an application that performs all the main functionality of the chat application. The most important thing is that we didn’t have to write the entire structure from scratch and we spent much less time on development and made an effective solution.

A demonstration of a working prototype can be viewed on YouTube platform *https://youtu.be/UaLOBUwQ8O8* and the code of the project can be viewed on GitLab repository *https://gitlab.com/Logain/chat-application*.

7 CONCLUSION

At the beginning of the project, we identified the goal of the diploma thesis as creating a prototype of a chat site where students could communicate and exchange information. In the present work, we identified the functional and non-functional requirements of the system, visualized the interactions users with the system using UML Use Case diagram, made the design of the database and the system, identified the structure of the site, described the processes of the system with UML Deployment diagram, compared various development frameworks, installed IDE, Composer, Laravel, Vue.JS, Pusher, Jetstream, and other auxiliary tools, wrote the project code, used a GIT system for development and backup project, conducted a tests of performance and analyzed the prototype functions. Eventually, we created a prototype with a simple but intuitive interface and the most basic functions. Its performance and some problems were demonstrated. Also, we have analyzed the development process, outlined some frameworks and libraries, and showed how they made development easier. Some development issues have been shown and after analyzing the results, we found that the prototype works well.

The built prototype can actually fulfill the tasks assigned to the project. Authorized users can enter thematic rooms, communicate and exchange information, and if the required room is not there, it can be created. However, despite the project's workability, it is also worth noting that this is only a prototype of the system. For a working system, it is important to have a number of other equally important functions, such as smart search by rooms, sorting list, deleting rooms that the user creates, private rooms, sending files in chat, etc. The project didn’t have a task to implement these functions, but they are important for such application. And as mentioned above, thanks to frameworks, these functions can be implemented much easier.

Also, it was demonstrated how such tools as frameworks and libraries accelerate development, make it more efficient and safer. All these factors lead to a reduction in the cost of the project, which is a very significant point in the decision to create such an application.

Summarizing all of the above, we can say that all of our goals were achieved. Using modern development methods, we have created a working prototype, that can become the basis for the project.

8 DALJŠI POVZETEK V SLOVENSKEM JEZIKU

Na začetku projekta smo za cilj diplomske naloge opredelili izdelavo prototipa klepetalnice, kjer bi študenti lahko komunicirali in izmenjevali informacije. V pričujočem delu smo identificirali funkcionalne in nefunkcionalne zahteve sistema, vizualizirali interakcije uporabnikov s sistemom z uporabo UML Use Case diagrama, izdelali načrt podatkovne baze in sistema, identificirali strukturo spletne strani, opisali procese sistema z UML Deployment diagramom, primerjali različna razvojna ogrodja, namestili IDE, Composer, Laravel, Vue.JS, Pusher, Jetstream in druga pomožna orodja, napisali kodo projekta, uporabili GIT sistem za razvoj in varnostno kopiranje projekta, izvedli preizkusi zmogljivosti in analizirali prototipne funkcije. Na koncu smo ustvarili prototip s preprostim, a intuitivnim uporabniškim vmesnikom in najosnovnejšimi funkcijami. Pokazali smo njegovo zmogljivost in nekatere težave. Prav tako smo analizirali razvojni proces, orisali nekatera ogrodja in knjižnice ter pokazali, kako so olajšali razvoj. Prikazane so bile nekatere težave pri razvoju in po analizi rezultatov smo ugotovili, da prototip deluje dobro.

Zgrajen prototip lahko dejansko izpolni naloge, ki so bile dodeljene projektu. Pooblaščeni uporabniki lahko vstopajo v tematske prostore, komunicirajo in izmenjujejo informacije, če pa zahtevane sobe ni, jo je mogoče ustvariti. Vendar pa, kljub delujočemu projektu, velja tudi omeniti, da je to le prototip sistema. Za delujoč sistem je pomembno imeti številne druge enako pomembne funkcije, kot so pametno iskanje po sobah, razvrščanje po seznamu, brisanje prostorov, ki jih ustvari uporabnik, zasebne sobe, pošiljanje datotek v klepet, itd. Projekt ni imel za nalogo implementirati te funkcije, vendar so za takšno aplikacijo pomembne. In kot je omenjeno zgoraj, je s pomočjo razvojnih ogrodij te funkcije mogoče veliko lažje implementirati.

Prav tako je bilo prikazano, kako orodja, kot so ogrodja in knjižnice, pospešujejo razvoj, ga naredijo učinkovitejšega in varnejšega. Vsi ti dejavniki vodijo k znižanju stroškov projekta, kar je zelo pomembna točka pri odločitvi za izdelavo takšne aplikacije.

Če povzamemo vse našteto, lahko rečemo, da so bili vsi naši cilji doseženi. Z uporabo sodobnih razvojnih metod smo izdelali delujoč prototip, ki lahko postane osnova za projekt.

9 REFERENCES

[1] "MVC Laravel," TrevorThompson, 2019. [Online]. Available: https://trevor-thompson.com/posts/an-introduction-to-laravel-and-its-mvc-architecture.

[2] "Statistics and data: Most Popular Backend Frameworks," Statistics and data, 2021. [Online]. Available: https://statisticsanddata.org/data/most-popular-backend-frameworks-2012-2021/.

[3] "Statistics and data: Most Popular JavaScrypt Frameworks," Statistics and data, 2021. [Online]. Available: https://statisticsanddata.org/data/the-most-popular-javascript-frameworks-2011-2021/.

[4] "Jetbrains: phpstorm," Jetbrains, [Online]. Available: https://www.jetbrains.com/phpstorm/.

[5] S. Nikita, "Gitlab," 2022. [Online]. Available: https://gitlab.com/Logain/chat-application.

[6] "Documentation of Laravel," Laravel, [Online]. Available: https://laravel.com/docs/8.x.

[7] "Documentation of JetStream," Laravel, [Online]. Available: https://jetstream.laravel.com/2.x/introduction.html.

[8] "Documentation of VueJS," Vue.js, [Online]. Available: https://vuejs.org/guide/introduction.html.

[9] "Documentation of Pusher," Pusher Limited, [Online]. Available: https://pusher.com/docs/channels/.

[10] C. Macrae, "Vue.js: Up and Running: Building Accessible and Performant Web Apps," O'Reilly Media; 1st edition, 2018.

[11] M. Stauffer, "Laravel: A Framework for Building Modern PHP Apps," O'Reilly Media; 2nd edition, 2019.

[12] J. Lockhart, "Modern PHP: New Features and Good Practices," O'Reilly Media; 1st edition, 2015.

[13] C. Churcher, "Beginning Database Design: From Novice to Professional 2nd Edition Book," 2012.

[14] S. Byanjankar, "LDAP Authentication in Laravel," May 2020. [Online]. Available: https://medium.com/@byanjankarsujan/ldap-authentication-in-laravel-516edbdedfd8.

[15] S. Nikita, "Prototype demonstration," 2022. [Online]. Available: https://youtu.be/UaLOBUwQ8O8.

[16] Scrypster, "Real-Time Chat with Laravel, Vue.js & Pusher," 2020. [Online]. Available: https://youtu.be/UaLOBUwQ8O8.