# Implementation of a chat application

Jan Steeg, *924589*

Nils Jung, *924577*

Jule Martensen, *924607*

Marc Engelmann, *934089*

January 13, 2019

## Contents

All authors are members of the University of Applied Science in Kiel.

## ABSTRACT

This documentation describes the development of a Chat Application which has been done as the accompanying practical project during the module Advanced Software Programming in the winter term 2018/2019 at the University of Applied Sciences Kiel. With a focus of the use of JavaScript the application consists of both a React-Redux-based Frontend and a NodeJS backend with a MongoDB database, connected via websockets. The Chat Application is containerized with Docker.

# 1. INTRODUCTION

As part of the mandatory course Advanced Software Programming the task of a accompanying JavaScript-based project has been given with the goal to reach abilities of full stack-development. Plain JavaScript is a well-developed area with best-practices for most applications. Nevertheless the team decided to experiment with state-of-art technologies to familiarize itself with modern and well-used frameworks and tools such as React-Redux, Bootstrap and Docker. Nowadays web developers often are confronted with those broadly used technologies which gave the team additional attraction as research objectives:

- Styling and structuring frameworks
- Frontend frameworks (like Vue.js, ReactJS, AngularJS)
- Backend frameworks for NodeJS (like Express, Koa)
- Toolchain

Beside those team individual requirements of experimenting with modern frameworks given requirements were:

- Git as version control service
- User sign-up and authentication
- Database access
- Websockets

The development management was based on modern iterative approaches and used support tools and services for cooperative development such as Github and the git-flow workflow, Kanban boards with tickets (user stories, bug tickets) and code reviews.

## 1.1 Initial Phase

During the first meeting, after the selection of team members, the discussion about the concrete application began. The development of a module based chat application has been selected for multiple reasons: encapsulated development of features as modules on a basic chat, which can be extended independently during the development. After the decision the first User-Stories were described and documented to fix the vision of the application and to serve as the first project specific requirements:

- Basic chat functionality
- Translation functionality within a message
- Usage of emojis
- Markdown support
- Users can join chat rooms
- Modifiable user settings

- Online status

For the initial commit each team member was given the opportunity to familiarize himself with technologies while building a rather simple chat application. These prototypes served as candidates for the initial version of the whole team from which one got selected which already contained primitive React-Redux functionalities. From this point forward the created requirements had been imported into the GitHub ticket system and each member was able to start developing features by the Git workflow.

# 2. TECHNOLOGIES

The mainly used technologies are depicted in figure 1. The team decided to work with MongoDB combined with a Node.js server in the backend. The frontend is built with the javascript Framework reactjs and the CSS-Framework Bootstrap since these two frameworks are widely used. The real-time communication is realized with the javascript library Socket.io, which needed to be implemented on both sides.

## 2.1 Backend

### 2.1.1 Sockets

The real-time communication between frontend and backend is established via websockets provided by the javascript library Socket.io. In a chat application it is necessary to handle real-time events between clients and server (bidirectional communication). The server listens for incoming connections (sockets) and handles the following events:

- connect/disconnect
- send message
- change onlineStatus
- join/leave chatroom

The connected clients can emit events to the server and receive events, that are emitted by the server. When emitting an event, it is possible to append a JSON object to transmit data. Socket.io also provides the functionality to emit events to a specific room, that the sockets can join or leave. Events emitted to a room can only be received by clients, that have joined it.

### 2.1.2 Rest API

To store data to the database and establish a connection between server and client there are several concepts like for example REST, GraphQl or SOAP. The focus was to build a communication layer as fast as possible to enable the data transfer. Furthermore the data structure and client-side requests are not that much nested as GraphQl would come to advantage[*]. So the team developed a REST API for the endpoints `/user` and `/chatroom`[†].

[*] `https://www.robinwieruch.de/why-graphql-advantages-disadvantages-alternatives/`

[†] For a full API Reference take a look at API Documentation:`https://github.com/nilsjung/chat-application/wiki`

### 2.1.3 Mongo DB

MongoDB is a non-relational database, which stores data as self-describing documents represented by binary JSON Objects (BSON). Since many big organizations are using MongoDB and the development is supposed to be flexible and simplified, the team decided to integrate it into the application.

The modelling of the application data is done with the javascript library Mongoose. Mongoose provides the possibility to create schemas, which define the format of a document in MongoDB, as for example a user.[*]

### 2.1.4 Express

One of the most essential and popular frameworks for NodeJS-based web and mobile applications is Express (further information express) which provides a diverse amount of methods to implement the fundamentals of a web server as backend such as: simpler HTTP request handling than plain NodeJS routing One key functionality is the implementation of middlewares which processes incoming requests by a queue of functions for evaluating and executing. The main use of those middlewares is the routing from legit URL requests to the corresponding component endpoints and otherwise returning a describing error message. [†]

### 2.1.5 Unit Tests

As the authors learned about test driven development (TDD) during the course of study, they now had the possibilities to test these concept as it is often recommended to write code of high quality. As in the javascript area there are many testing libraries, it was decided to test one of them. The final decision was to use the testing framework mocha with the library chai. The team chose to try the concept of TDD, to develop the API as it is hardly possible to implement the wanted behaviour without testing. The time, that was needed to implement the testing structure and refactor the first concepts was definitive well spended as the tests helped a lot to check for the right outcome for the requests.

## 2.2 Frontend

### 2.2.1 Reactjs

Reactjs is an open source tool developed and mainly maintained by Facebook, over the last couple of years it has drawn much attention as a fast and relatively lightweight frontend framework which is the reason why the authors wanted to explore this technology. The main benefit of reactjs is the fast render times without much cost. It does so by re-rendering only those elements which have changed. React has two core components the React.Elements and the React.Components. React.Elements only exist in the virtual dom which is then rendered as a usal dom object when needed[‡]. React.Elements are written in javascript syntax extention (JSX) which look and feel similar to HTML and XML and it allows developers to write HTML and javascript within a single React.Element[§]. They are however stateless, therefore react uses components which are state full and do not exist inside the virtual dom. React.Components are converted into React.Elements and when a state change occurs the element will be inserted into the actual dom. React.Components can refer to other React.Components which allow the development of complex user interfaces by reuse of React.Components in a composite manner instead of inheritance.

[*] `https://www.mongodb.com/de,https://mongoosejs.com/`

[†] Further information about the endpoint API can be found on the Wiki `https://github.com/nilsjung/chat-application/wiki`

[‡] `https://reactkungfu.com/2015/10/the-difference-between-virtual-dom-and-dom/`

[§] `https://facebook.github.io/jsx/`

### 2.2.2 Redux

In addition to reactjs the team chose the redux approach to manage the state in a centralized way, so components dont need to pass their state around. Apart from that redux provides the following benefits:[*]

- Easy debugging
  - There are debugging tools, that allow to track the state transitions during the runtime.
- Testability
  - Redux works with pure functions altering the state, so they are easy to test.
- Consistent behaviour
  - Due to the usage of pure functions, the state is always predictable since the same actions produce the same output.

As soon as a component intercepts an event, as for example a mouse click, it creates an action, which is basically a function. The action gets dispatched to the store, which triggers the corresponding reducer function. As already mentioned, reducer functions are pure. They only take the previous state and the action and returns a new state to the store, so the state is never altered directly. As soon as the state has changed, the component is triggered to rerender and thus gets updated.[†]

### 2.2.3 Thunks

The redux dispatch function expects an action, which consists of an action-type and an object. Usually the dispatch function receives a so called action-creator, which is just a function returning the action. However when executing asynchronous code, as for example a HTTP-Request or a timeout, there is a problem with these action creators since they do not return an action anymore, but a promise. Redux Thunks is a middleware which handles such side effects in redux. Basically the action creator being dispatched, returns a function in which the asynchronous task can be executed. This function is called a thunk. The actual action object is dispatched inside the thunk after the async task has been terminated. The way the middleware works is visualised in the following code snippet. When the dispatch function is called, the middleware considers the type of the action.

```
actionOrThunk =>
        typeof actionOrThunk === 'function'
        ? actionOrThunk(dispatch, getState)
        : passAlong(actionOrThunk);
```

If the action is of the type function, the dispatch- and the getState-function are passed into it, so that the action can be dispatched later (after the termination of asynchronous code). Otherwise the action is just passed to the reducer as usually.[‡]

[*] https://blog.logrocket.com/why-use-redux-reasons-with-clear-examples-d21bffd5835

[†] Figure 6 shows a basic redux architecture.

[‡] https://github.com/reduxjs/redux-thunk,https://medium.com/fullstack-academy/thunks-in-redux-the-basics-85e538a3fe60

### 2.2.4 Bootstrap

Bootstrap was used to deal with the frontend styling and have the ability to use the responsive behaviour given by this styling framework. Doing this manually results in a higher effort. With Bootstrap, or any other styling framework, the developer gets also a predefined layout system he can use to structure the website. Furthermore there are some basic components with a default style as well as a basic behaviour. It is possible to customize the layout by override the default styling in the sass-files. To do so it is necessary to include the required bootstrap files in the custom sass-files and load the own styling definitions.

### 2.2.5 Translation API

Translating text of any language into another language is a difficult task on its own. The knowledge and computational power to do so is beyond most developers and companies since it would not be financially feasible to do. But many APIs and tools have been developed over the years and are available[*]. The Authors have looked into four Apis and tested three Apis to translate text

- Google Translate API
- Microsoft Translate API
- Yandex Translate API
- Watson Translation API (The Authors were unable to get a working API key for testing)

All three tested APIs work on the same base, a http request contains the source language (all tested API also have an auto detect function to detect the source language, either as seperate request or build in the same request), the target language and the text to be translated. The response contains a json object with the translated text. The Microsoft Translate API was able to translate text into multiple languages. The others require multiple requests. At the end the team decided on using yandex[†] since it was the most lightweight solution and the API key was easy to obtain[‡]. As a result, messages prior to sending can be translated into a set of given languages, and prior received or sent messages can be translated for view.

### 2.2.6 Emojis

To use emojis the authors used the emoji-mart library, a leightweight and easy-to-use library. The library contains an emoji-picker so that the user does not have to type the emoji or its corresponding codepoint into the text directly. Emojis picked will be appended to the already written text. The picker was wrapped in an extra component so that it can be used anywhere in the application.

## 2.3 Architecture

As an example the `Chat` component is composed of a `MessageList`, a `TextInput`, a `UserList` and a `Chatroom` component. The `TextInput` and the `Chatroom` component both use a component called `InputWithButton` which is just a custom input field with a button attached to it. Also a simple `Input` and a `Button` component was created. both are used by the `RegistrationForm` and the `LoginForm`. The usage of reusable components avoids

[*] `https://www.programmableweb.com/category/translation/api` lists over 150 of them

[†] `https://translate.yandex.com/`

[‡] It allows to send more data , microsoft and google allow 5.000 characters yandex allows 10.000

duplicated code and results in a consistent appearance of the user interface. To keep the code organization clean, the redux actions which are dispatched inside a component are kept inside files that refer to the component. For example the actions dispatched in the chatroom component are defined in a file called chatroomActions.[*]

### 2.4 Database Model

A `Chatroom` is created by a user and can be joined by multiple users. It consists of a name, which is its identifier, an array of messages and an array of users being member of the chat room. However a `UserChat` is similar to a `Chatroom` but only handles the communication between two users. It is identifiable by an ID since it doesnt need to have a name.[†] The `ChatUser` model stores the members of a `Chatroom`. It includes a name, the mail address and a role. A user can either be the admin of the chat room or just a user. The `User` model stores users of the application in general. Thus it also contains information like nickname and password.

A `Message` consists of a user, who wrote the message, the actual text and a timestamp defining the time when the message was sent.

### 2.5 Chat Communication

Figure 4 depicts the communication between frontend, REST API, database and Websocket, when the user performs chatroom actions.[‡]

When the user navigates to the route Chat on the frontend side, the client sends a request to the REST API to retrieve all chat rooms available. In the backend the chat rooms are fetched from the database and subsequently returned to the client. The chat rooms are rendered inside a list, so the user can select one. When the user opens a chat, a corresponding request is sent to the backend. The chat is identifiable by its name and can be fetched from the database. The REST API returns the chats inside the chat room, so that they can be rendered in the frontend. In order to receive messages in the selected chat room, the socket emits a joinChatroom-event, which makes the client leave the old room and join the new one. If the user sends a message inside a chat room, the websocket emits a message-event, containing information about the user, the chat room and the message. The message is stored to the related chat room inside the database, so it can be retrieved later. The websocket on the backend side also emits a broadcast message event to all other connected clients, that are members of the room. For a full workflow see figure 4.

### 2.6 Security

Security was not the primary focus on this project two features had to be implemented. The encryption of passwords and the ability to limit user actions to validated users.

#### 2.6.1 JWT

To identify that a user is logged in with valid credentials, jsonwebtoken[§] are used. They are generated on successful login and stored in the redux state and are furthermore required for any action. If a call to the backend is received the token is verified, if the validation fails a 403 will be sent. For a full workflow see figure 5

[*]In figure 3 one can see an extract of the created react components.

[†]As depicted in figure 2 the database model consists of five different schemas.

[‡]Although the routes of the REST API are protected and only accessible with a valid access token, the does not include the validation step in order to keep it clearer.

[§]`https://www.rfc-editor.org/rfc/pdfrfc/rfc7519.txt.pdf`

#### 2.6.2 Bcrypt

To secure the users password the password is encrypted with bcrypt[*]. Bcrypt is a hashing function based on the blowfish cypher[†] with an adaptive cost. If an attack is likely the calculation rounds can be increased. Bcryptjs serves as the library that implements the bcrypt algorithms, the standard round cost of 10 and salt length of 10 were used.

### 2.7 Organization and Communication

The whole communication (beside the two-weekly mandatory lab session) was based on the services Slack and GitHub. Slack, a well-known cloud-based chat service, served as channel for any non-code related information or discussion. GitHub on the other hand served as hosting service for the project and organization of tickets. Every feature progress was handled by the GitHub-integrated Kanban board with the following stages:

- Backlog
- To Do (Sprint Backlog)
- In Progress
- To Review
- In Review
- Done

The basic stages To Do, In Progress and Done got expanded to realize a stronger splitting between short- and mid-/long-term ticket (two backlogs) and an independent structure for code reviews. By convention of the git-flow workflow[‡], after a team member assigned a ticket to himself and moving it into the stage In Progress, the member created a branch with a name linked to the ticket (e.g. task-42-describing-text-about-ticket). After realization of the ticket requirements, the member creates a pull-request to open the changes for code reviews. Code reviews were an intensively used progress, as at least one additional team member had to test and discuss the changes of the ticket on code level with the developing team member. No feature/bugfix was merged into the master branch without such an additional external approval. The team matrix can be found in table 1

## 3. RESULTS

The running application is displayed on the screenshots 7 and 8. Picture 7 visualizes an example conversation between two users, sending messages, emojis and translating messages into another language. Table 2 lists the user stories the team has accomplished as well as the tasks that are planned for the future.[§]

[*]A Future-Adaptable Password Scheme,in USENIX Annual Technical Conferenc

[†]`https://www.schneier.com/academic/archives/1994/09/description_of_a_new.html`

[‡]`https://www.atlassian.com/git/tutorials/comparing-workflows/gitflow-workflow`

[§]Nine of the required user stories had been realized.

## 4. DISCUSSION

The project kicked-off with five team members in the first meeting. After the mid-term presentation, in which the attendance of each team member was mandatory, one member had left the team, reducing the team size to four members. During the development the importance of intense code reviews and cooperate thinking became even more clearer than expected and already known, due to different levels of experiences of reviews in the team. A large amount of time has been invested in refactoring and restructuring to changed requirements by new features, which made reviews an essential part in the development with the side effect of learning and understanding the programming style of other developers. Experiments and training with new (and often unknown) technologies and frameworks required a larger time investment in the first half of the development than expected, reducing the target amount of features at the end. A specific example is the usage of Docker and Docker-Compose which might have resulted in a too ambiguous overhead for a project of such a size, as multiple team members had problems running the docker-based project in the beginning. A traditional server would probably have been less problematical. Nonetheless, the development of a Docker-based multi-component server had been a valuable experience in software deployment and in regards of developing in a full-stack manner.

## 5. CONCLUSION & OUTLOOK

At lot of time has been spent to understand the used technologies as deeply as possible by all developers which resulted in multiply branches working on the same working categories at the same time resulting in a overhead when the initial merges had to be made. Once a core has been established the addition of features and general speed of development increased to an acceptable leve since every developer is now able to understand how the technology stack works and can contribute not just on a conceptional base but also in a technical discussion. However, our conceptional idea of a module based chat application where features could easily be added similar to a plugin approach was not successful. Feature implementation does however work smoothly now. Since we used react a component driven feature addition for the frontend is our goal for the future. Developers should be able to independently develop components which then only have to be referenced at the wanted location.

## APPENDIX A. FIGURES

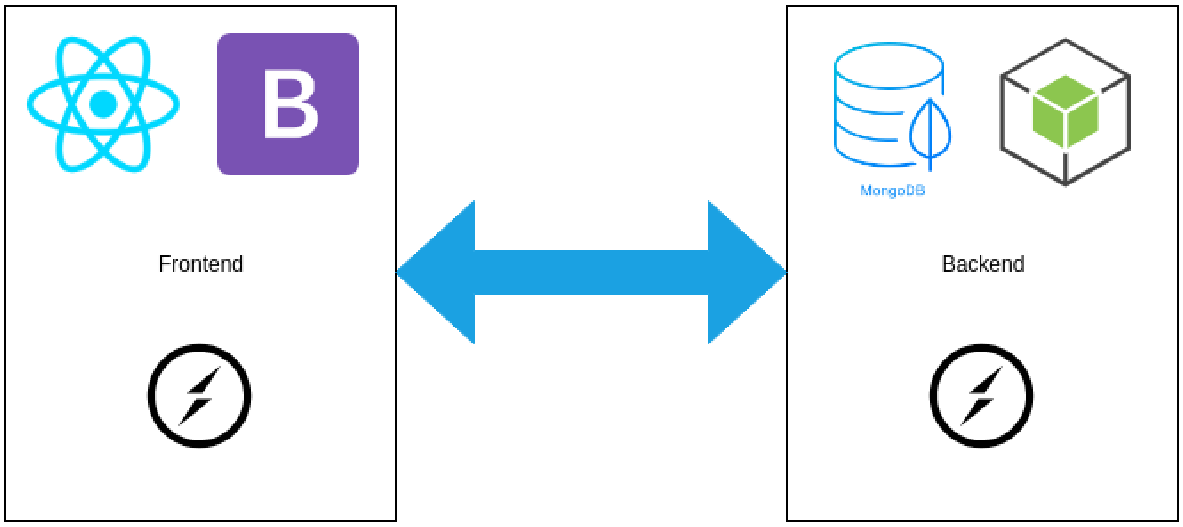


Figure 1. Basic technology stack

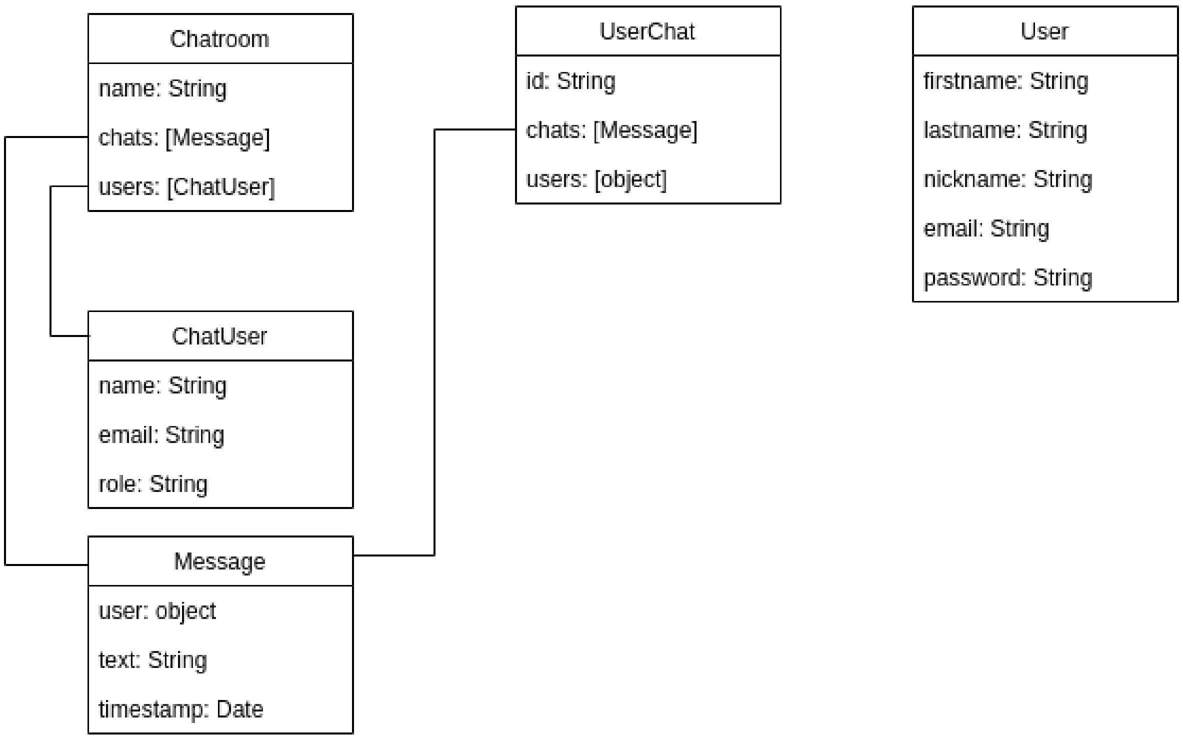


Figure 2. Data model

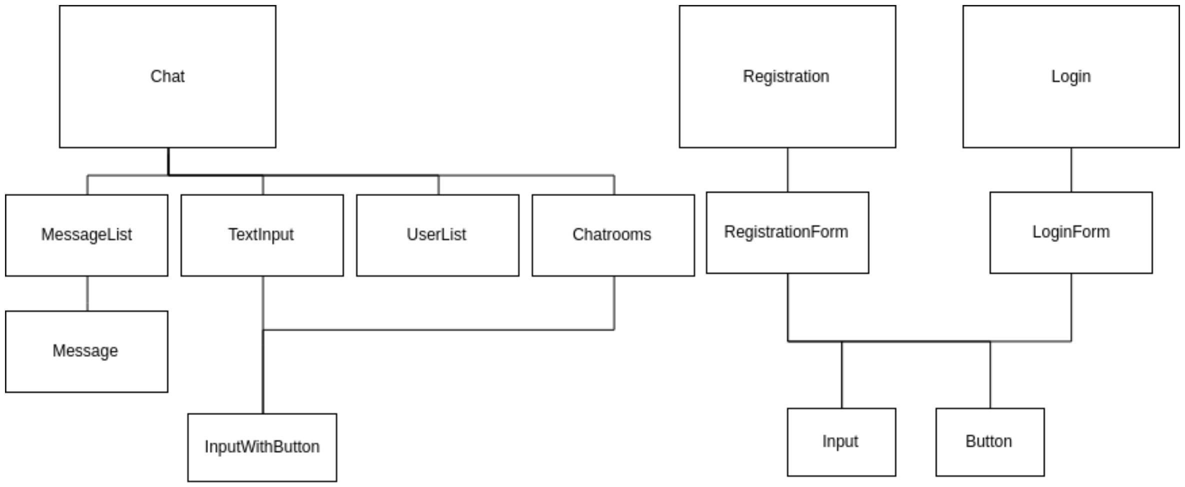


Figure 3. Components

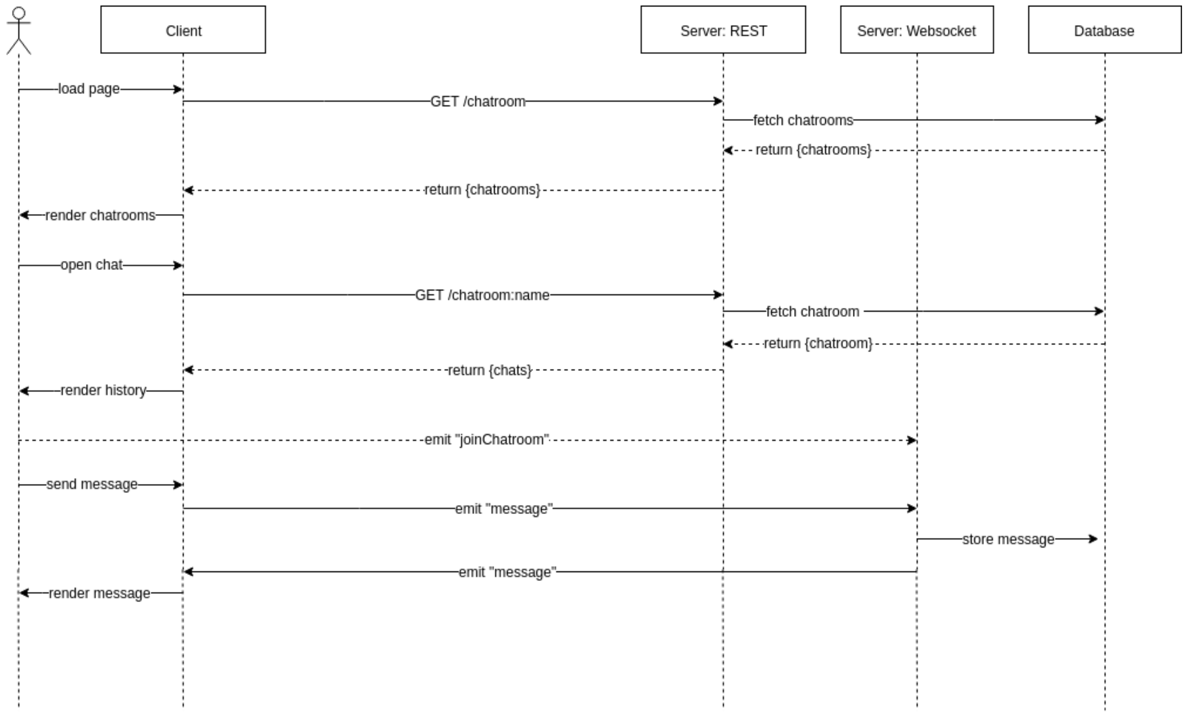


Figure 4. Chat Communication Flow

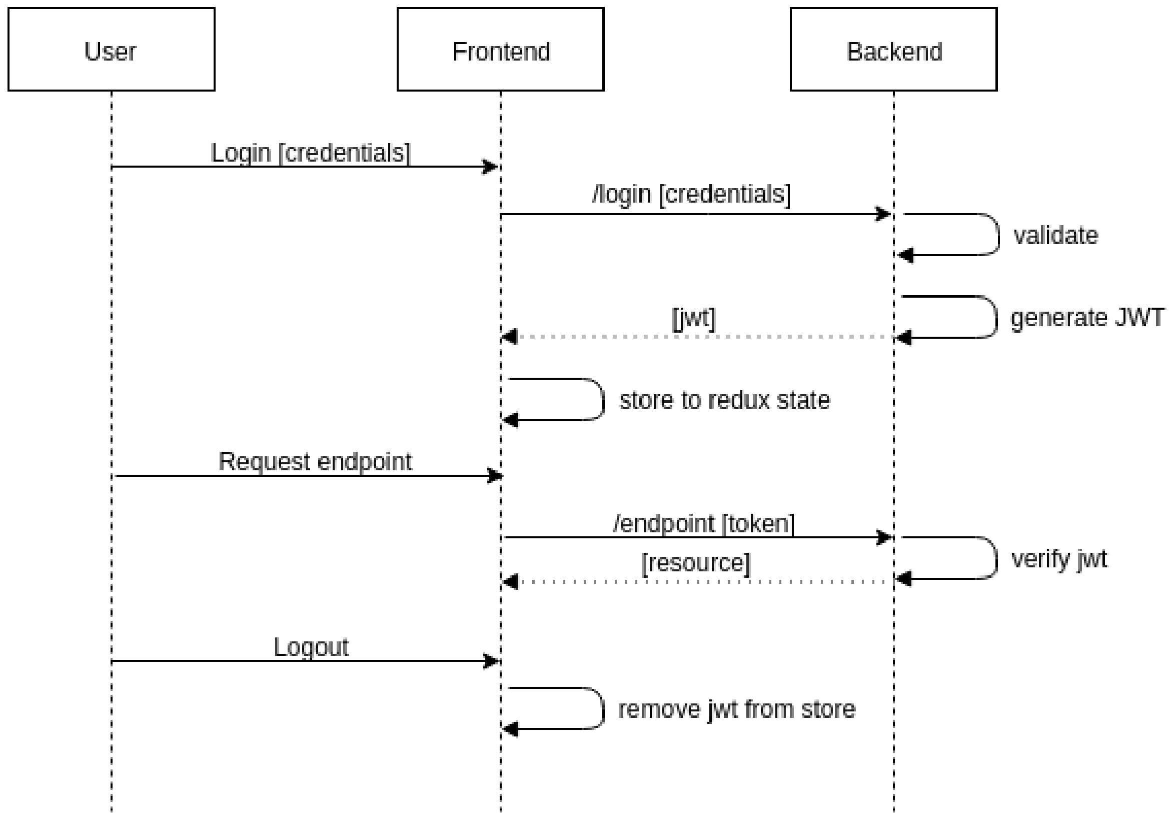


Figure 5. Authentication Flow

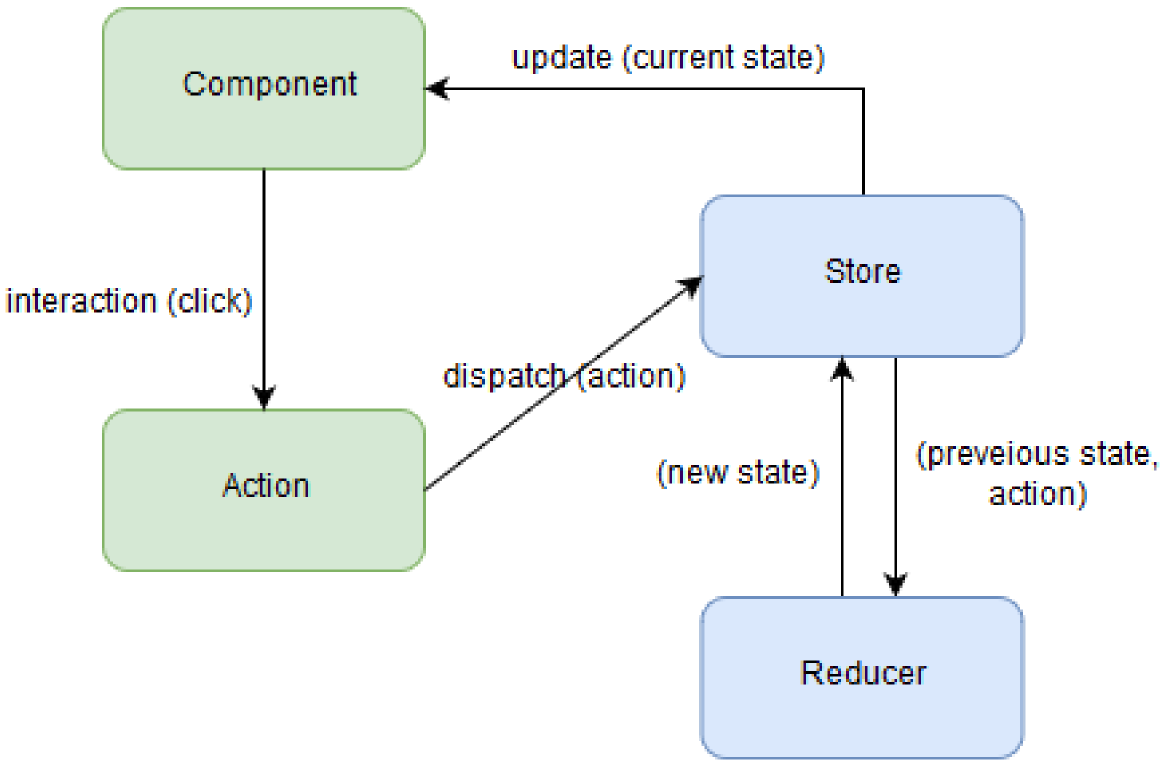
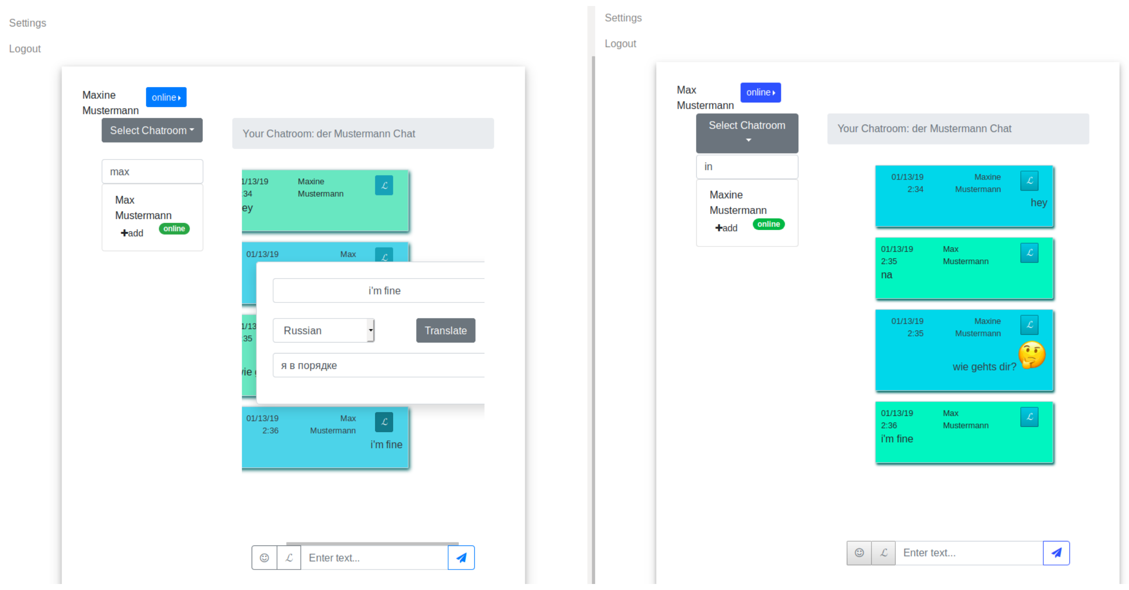


Figure 6. Redux Flow


Settings
Logout
Maxine Mustermann online
Select Chatroom
Your Chatroom: der Mustermann Chat
max
Max Mustermann +add online
1/13/19 34 Maxine Mustermann
ey
01/13/19 Max
i'm fine
Russian
Translate
я в порядке
01/13/19 2:36 Max Mustermann
i'm fine
Enter text...

Settings
Logout
Max Mustermann online
Select Chatroom
Your Chatroom: der Mustermann Chat
in
Maxine Mustermann +add online
01/13/19 2:34 Maxine Mustermann
hey
01/13/19 2:35 Max Mustermann
na
01/13/19 2:35 Maxine Mustermann
wie gehts dir?
01/13/19 2:36 Max Mustermann
i'm fine
Enter text...

Figure 7. Chat room

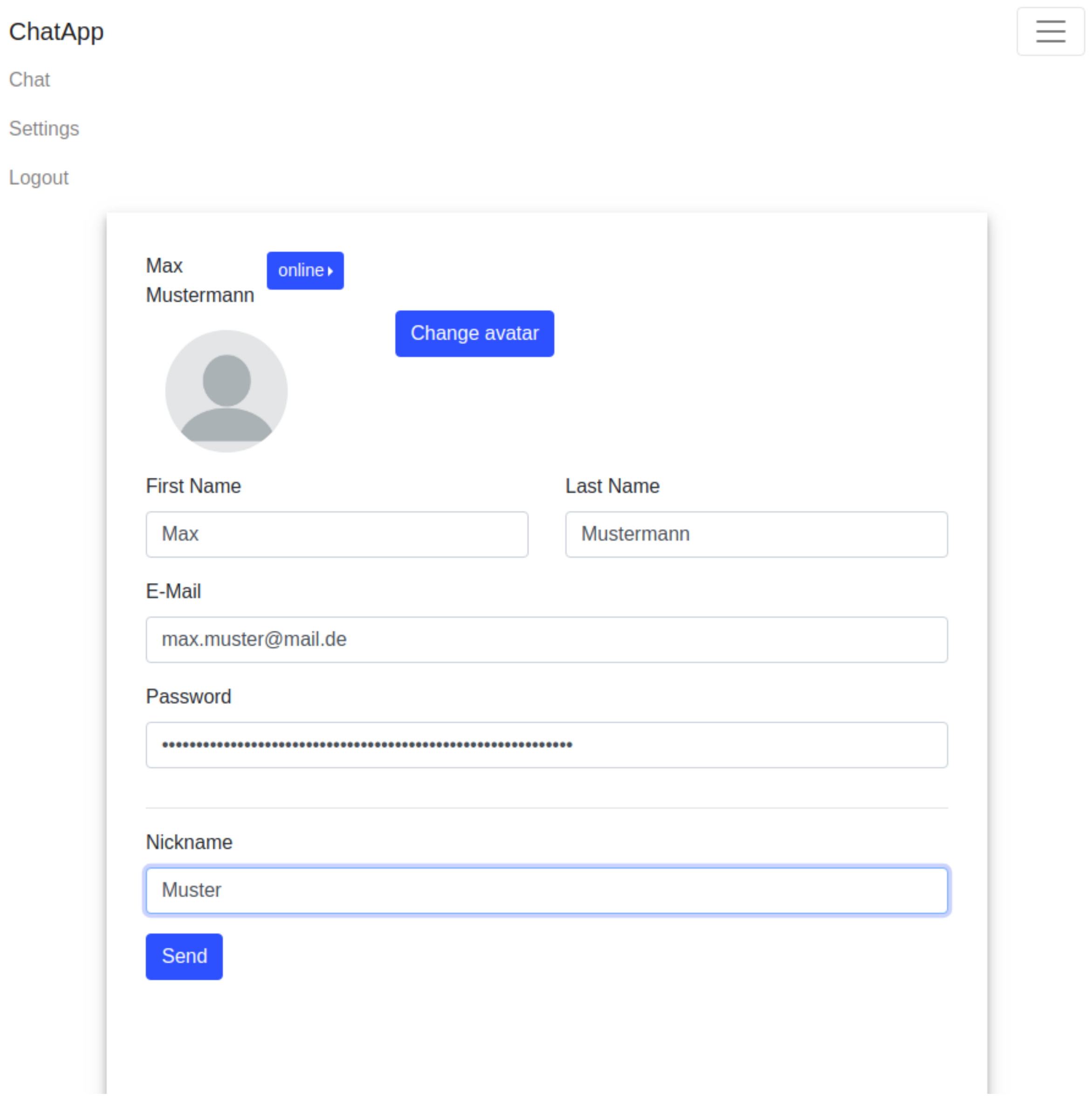


Figure 8. User settings

## APPENDIX B. TABLES

| | Jung | Martensen | Engelmann | Steeg |
|---|---|---|---|---|
| Registration | x | | | |
| Login | x | x | x | |
| Chatroom creation | | x | | |
| Change user settings | x | | | |
| Set online status | x | x | | x |
| Text translation via yandex | | | x | |
| Send/recieve messages | x | x | x | |
| Initialize/optimize development workflow (docker etc) | x | | | |
| Retrieve Chat History | | x | | |
| User list | x | x | | x |

Table 1. Task assignment for the project

| **Status** | **Feature** |
|---|---|
| Done | User can register |
| | User can see chat history |
| | User can write and send message |
| | User can create chat rooms |
| | User can change user preferences |
| | User can delete his created chat rooms |
| | User can infite others to chatroom |
| | User can translate messages |
| | User can send emojies |
| Feature Task | User can join chatroom on his own |
| | User can edit messages |
| | Guest can join chatroom with only reading access |
| | User can stay logged in with re-entering credentials |
| | User can see new messages as highlighted |
| | User can receive recommendation on channels |
| | User can receive a user rating due to his activity |
| | User can see the status and their updates of other users |
| | User can format his messages with newlines |
| | User can set his avatar |

Table 2. Feature list for the chat application