University College of Nordjylland
Sofiendalsvej 60, 9100 Aalborg

PBA Web Development
Spring 2020, pwe0919

Web Programming II - Semester 2

Messaging web app

Application info
App URL: https://stormy-tundra-13713.herokuapp.com/
Github repository: https://github.com/JAMT-UCN/react-chat-app
Test user:

- Email: test@jamt.com
- Password: qwe123

Quick start - run the app locally
(more info: https://github.com/JAMT-UCN/react-chat-app#readme):

- Install dependencies on the server side: *npm install*
- Install dependencies on the client side: *npm client-install*
- Run the client & server with concurrently: *npm run dev*
- Run the Express server only: *npm run server*
- Run the React client only: *npm run client*

Pages: 10.56
Characters (with spaces): 25.367
Characters (without spaces): 21.142

Participants:
Tomas Sedurskas
Maria Aldis Garðarsdóttir
Alessia Trianti
Jose Carlos Esparza Martí

Maria Aldis Alessia Trianti

March 18th, 2020
Aalborg, Denmark

Table Of Contents

1. Introduction

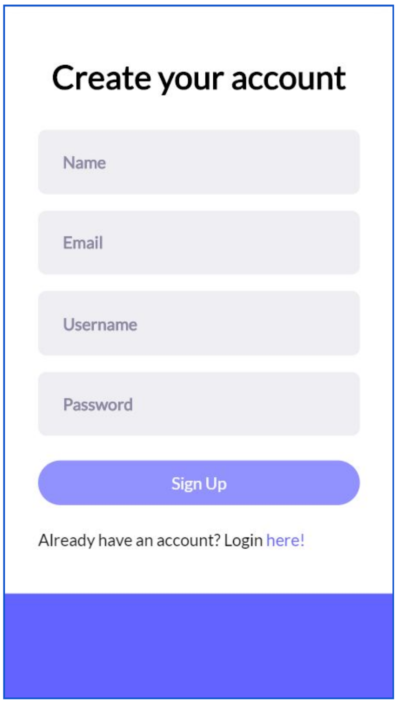
The aim of this project is to build a web-application with a chat function that allows users to build contacts and connect with their friends. This web-application should allow users to have an ability to add contacts, chat with friends and create group chats.

In order to achieve this goal, the MERN stack will be used to build the application. The MERN stack includes the database MongoDB, Express (a web framework for Node.js), the front-end framework React, as well as Node.js (a JavaScript runtime environment). Socket.io, a JavaScript library for realtime web applications, will also be used in order to enable real time communication between web clients and the server.

In this report, you will find an overview of the process that went into developing the web application. First, a description of the project is presented. Thereafter, discoveries about relevant technologies and reasons for the choice of technologies will be discussed. In addition, relationships between client- and server-side, as well as challenges faced during the process will be covered.

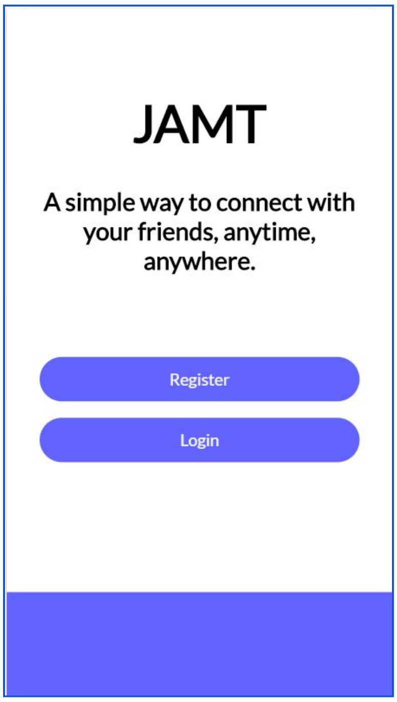
2. Project Description

When a user starts using the application for the first time, they must register their account. User-friendly error messages are given to the user if they enter incorrect data, such as invalid email or password, in order to start the experience well. Once the user has registered an account they can log in and start using the application.



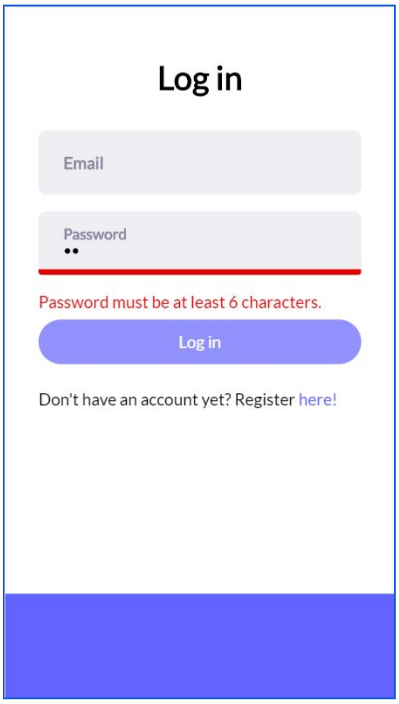
Create your account

Already have an account? Login here!



JAMT

A simple way to connect with your friends, anytime, anywhere.



Log in

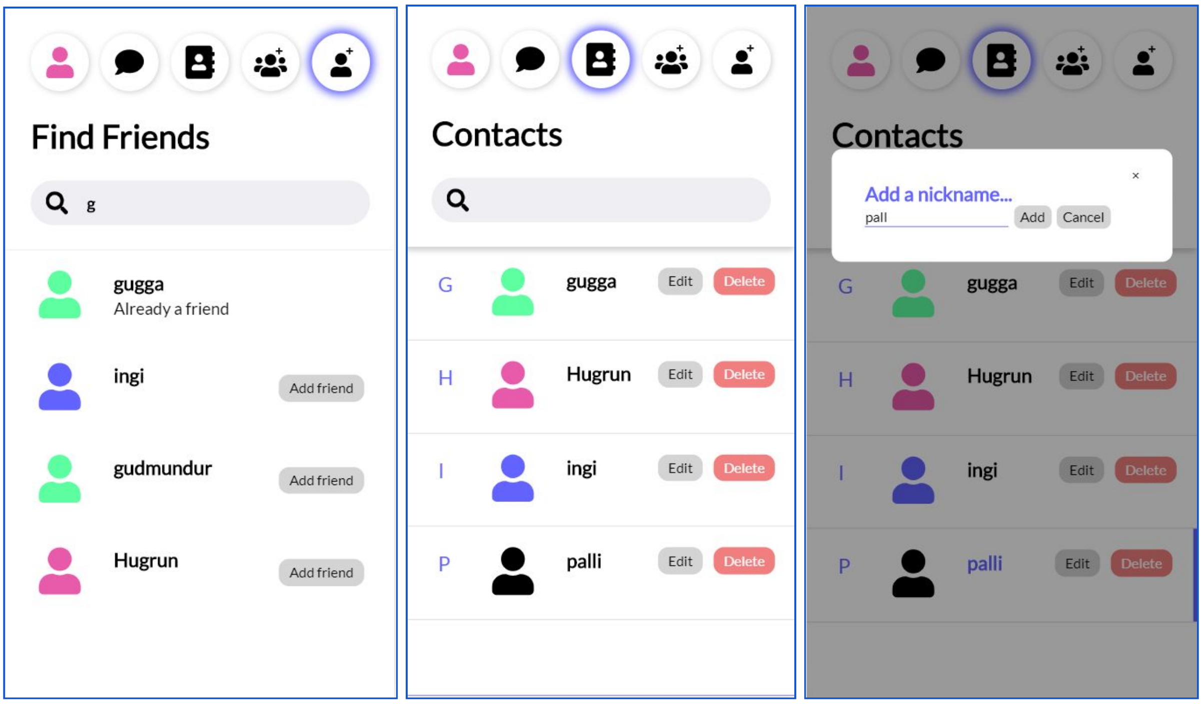
Password must be at least 6 characters.

Don't have an account yet? Register here!

Images nr. 1, 2, 3 : Screenshots of "Front", "Register", and "Log-in" pages

In order to use the app as intended, the user must add some contacts. On a page called "Find Friends", the user can search for some contacts by name (the user can enter characters and existing users containing those characters will appear in the search results).

The user can then navigate to a "Contacts" page, where they can see all their current contacts, delete a contact or give their contact a nickname. When a user clicks a contact, a new chat is created with the contact they have selected, and the user redirected to the newly created chat. The user can then start chatting with their contact.



Find Friends

g

gugga
Already a friend

ingi

gudmundur

Hugrun

Contacts

G gugga

H Hugrun

I ingi

P palli

Contacts

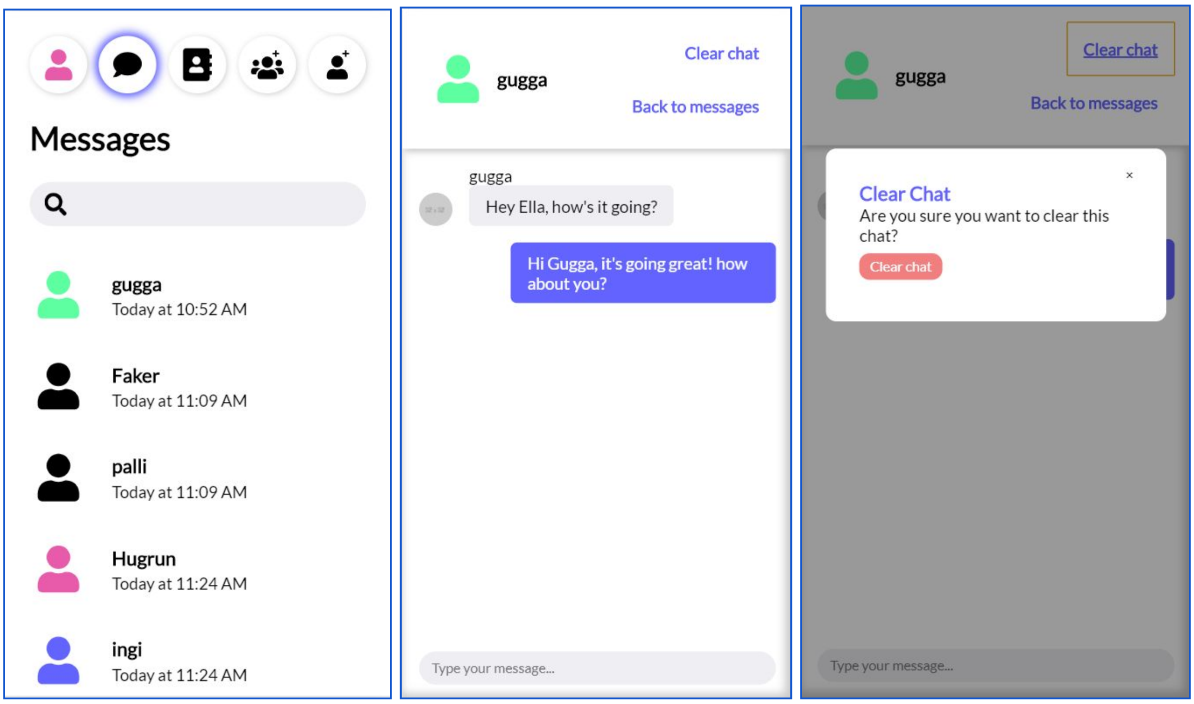
Add a nickname...

pall

Images nr. 4, 5, 6 :Screenshots of "Find Friends", "Contacts" pages

On the "Messages" page, the user can view all existing chats it has with different contacts. When the user selects a chat, they are taken to that specific chat.

If the user would like to clear a chat, it can be done by pressing a "Clear chat" button on the chat page. A pop-up message will then be displayed, asking the user if they are sure they would like to clear the chat.



Messages

gugga
Today at 10:52 AM

Faker
Today at 11:09 AM

palli
Today at 11:09 AM

Hugrun
Today at 11:24 AM

ingi
Today at 11:24 AM

gugga

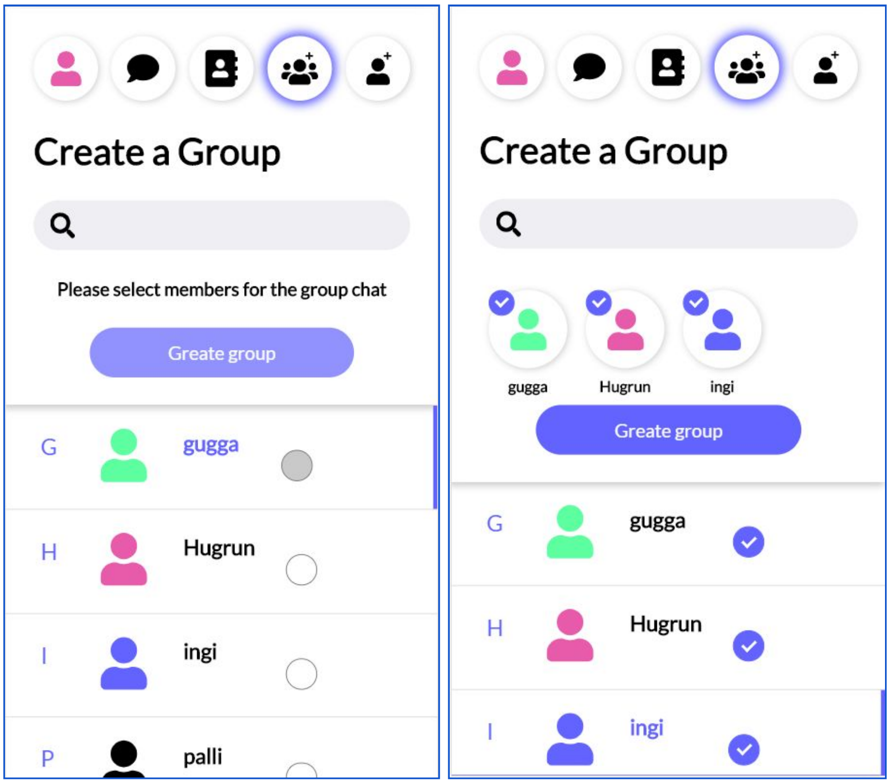
gugga: Hey Ella, how's it going?

Hi Gugga, it's going great! how about you?

Clear Chat
Are you sure you want to clear this chat?

Images nr. 7, 8, 9: Screenshots of "Messages" page, "Chat", and "Clear Chat" popup.

The user has an ability to create group chats as well. On the page "Create a Group", the user can view their contacts and select which contacts they would like to start a group chat with. After the user has chosen which contacts to have in a group chat, a "Create group" button is pressed, and the group chat is created. The user is taken to the group chat and can start chatting with the group.

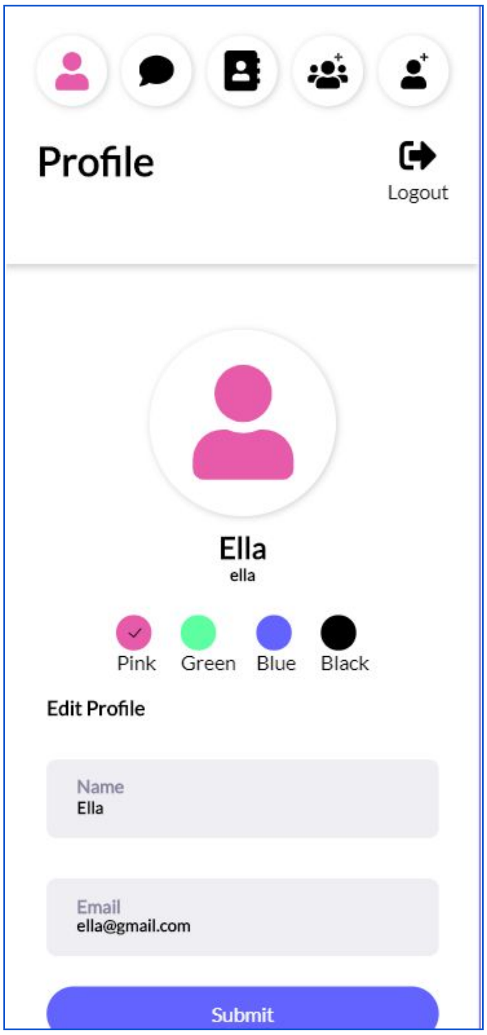


Create a Group

Please select members for the group chat

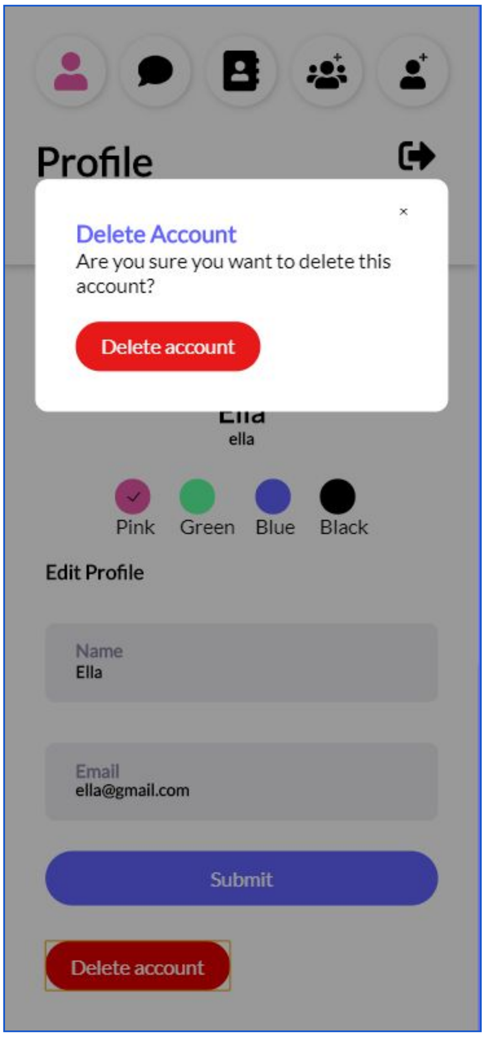
Images nr. 10, 11, 12: Screenshots of the "Create a Group" page.

On the “Profile" page, the user can edit their profile information, such as avatar colour, name, email and address, as well as delete their account.



Name
Ella

Email
ella@gmail.com



Delete Account
Are you sure you want to delete this account?

Name
Ella

Email
ella@gmail.com

Images nr. 13, 14, 15 : Screenshot of “Profile” page and “Delete Account” popup.

3. Introduction of technologies

3. 1. General Findings:

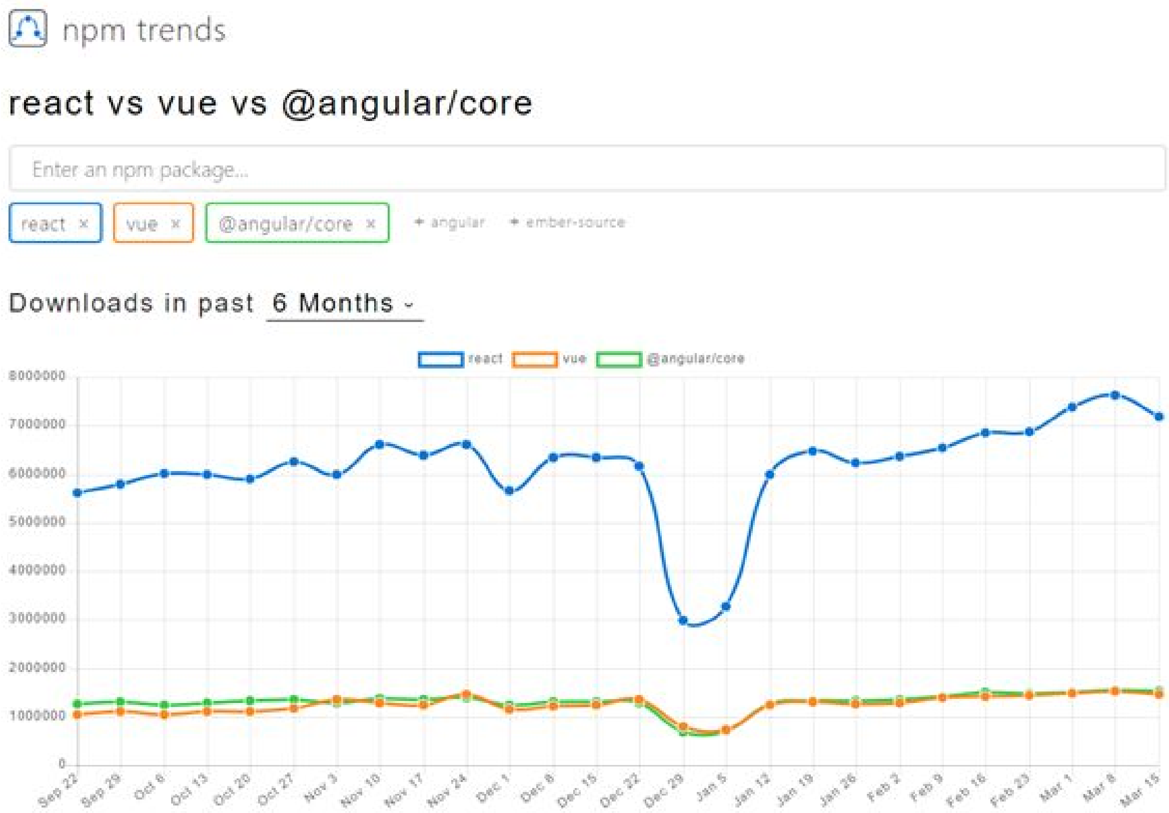
When choosing a front-end framework there are a lot of choices. Each of them has advantages and disadvantages in different areas. There is no one single perfect framework for everything, but there are three frameworks that stand out of the crowd as the go-to choices in the year 2020. Those frameworks are Angular, React and Vue. These are currently the most popular and widely used front-end frameworks with the biggest community support. This is how the selection of the front-end frameworks was narrowed down to just three options.

We based our selection on a number of criteria such as modularity, reusability, fit with our project, project size, poor code patterns, popularity and future job market relevancy. The project could have been developed and coded with the use of any of these frameworks, but we looked at what would fit our needs best based on the pros and cons of each framework.

3. 2. Modularity, reusability and size

To start with, the complexity and size of the messaging application is not that big. This means that Vue and React would do a great job in this case due to their smaller bundle sizes and lighter approach compared with what Angular has to offer. Since Angular has a lot of features pre-bundled out of the gate. This results in a bigger overall file size and in turn this would load slower compared to Vue or React (Daityari, 2020).

Moreover, React and Vue are similar in this case, but overall React has the biggest advantage in the case of it's modularity and reusability. Due to React being around for longer than Vue, React having a significantly bigger active user base of NPM's and being supported not only by a big company like Facebook, but also a big community of developers. React developers can easily download and implement modular components or features with the use of NPM. As the graph below shows, the amount of downloads of different NPM's for React is visibly higher by a significant margin. While Vue and Angular have a similar amount of downloads each month, way below the number of React related downloads.



react vs vue vs @angular/core

Downloads in past 6 Months

Image nr. 16 : Line *graph of most most NPM downloads in 2020 based on front-end frameworks (Potter, Npm trends, 2020)*

This could end up being a double-edged sword. On the up-side it could provide developers with all the standard and niche features and components they need with smaller final file sizes, but on the down-side the different features were not pre-bundled which means that during development some problems might arise. Problems with efficiency, performance or just poor code being written to make separate features or components connect and work together.

In our case, Angular seemed like the least favorable. There were a few reasons. Firstly, React and Vue could do all the things that we needed with smaller bundles. Secondly, Angular had a steeper learning curve and it introduces an additional technology – TypeScript. Lastly, Angular is mainly used for large scale application creation which is not something that this messaging app is. That is why Angular will no longer be mentioned in the further considerations of framework choice. (Daityari, 2020)

3. 3. Poor code patterns

React and Vue are very different in the way that their components are coded. React relies on JSX which means that both JavaScript, HTML and CSS can be in one single file. Usually all React components follow a similar pattern especially that with the release of React 16.0 in the second half of 2017 hooks were introduced into the framework. Developers were encouraged to switch from JavaScript class-based components to Functional components. Functional components are just a simpler, stripped down version of the class component, but as of 2020 functional components have all the main features that were in class components. Such features props, state, context and component life cycles. As a result the components became easier to code, easier to read and understand what the code does which is great for development in teams. (Facebook inc, 2020)

While Vue relies on the MVC (Model–view–controller) software design pattern. This pattern allows for more customizability and choice due to its two way binding. Technically, full Vue components can be coded in a single file containing the JavaScript, HTML and CSS, but usually that is not the common way of creating those components. Usually there are multiple files that make up a full component. FIles for HTML templates, CSS styling and JavaScript would all be separate.

Customizability and freedom of choice can be seen as advantages, especially when working alone on a project, but when working as a group of developers this freedom can become a disadvantage. The main concern is that different developers can have different coding styles and they could prefer different coding patterns. This could end up producing hard to read and understand code and ultimately end up slowing or preventing progress from being done. This is especially an issue for new teams that have no experience with each other's code. (Daityari, 2020) (Evan You, 2020)

All in all, Vue is a great framework for developing small scale applications due to it's lightweight and easy learning curve, but it could end up being a tough choice when working in a group, particularly when it's a new development team. React is also a great choice for the messaging app, and exactly because of its standardized code pattern it appears to be the more favorable choice for the current project with a new team of developers.

3. 4. Job Market and Future Popularity

Lastly, on a different note when choosing the framework, we also looked at the future perspective of getting employed based on the framework that we learn. Based on data from 2018.

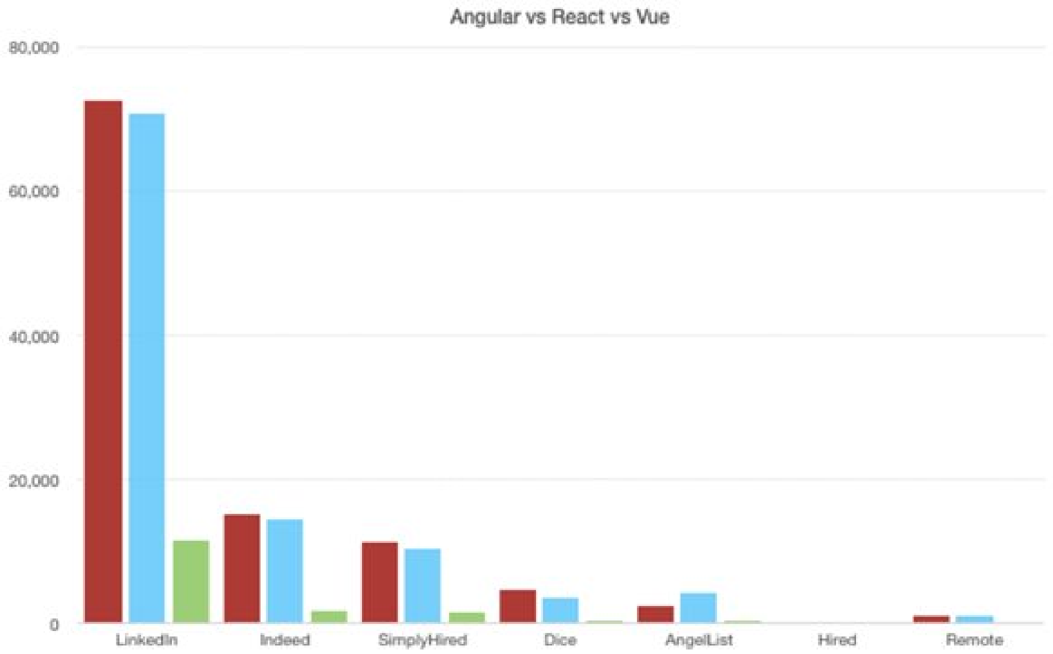


Image nr. 17 : Bar graph of job offers in 2018 based on front-end frameworks on most popular job search websites. (Neagoie, 2018)

Angular and React developers are in very high demand compared to Vue. Of course, Vue in 2018 was not as popular as it is right now, and it definitely has potential to grow in the future. But as of now React looks like a safer and more reliable choice. In addition to this, in 2020 React seems to be the most loved and used framework due to constant performance improvements, almost daily updates to their documentation and very big support from the NPM community.

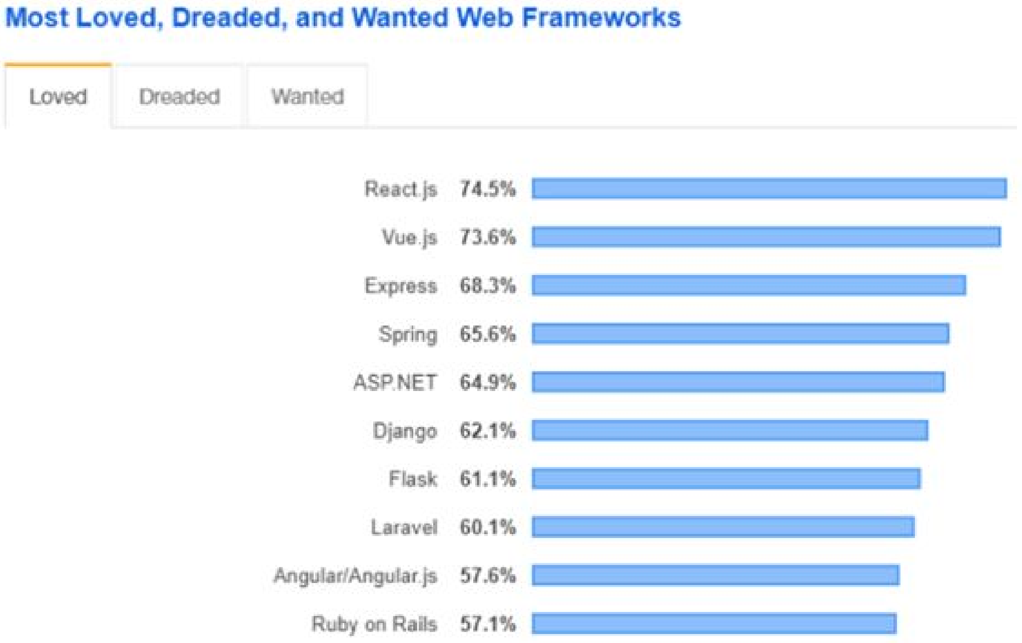
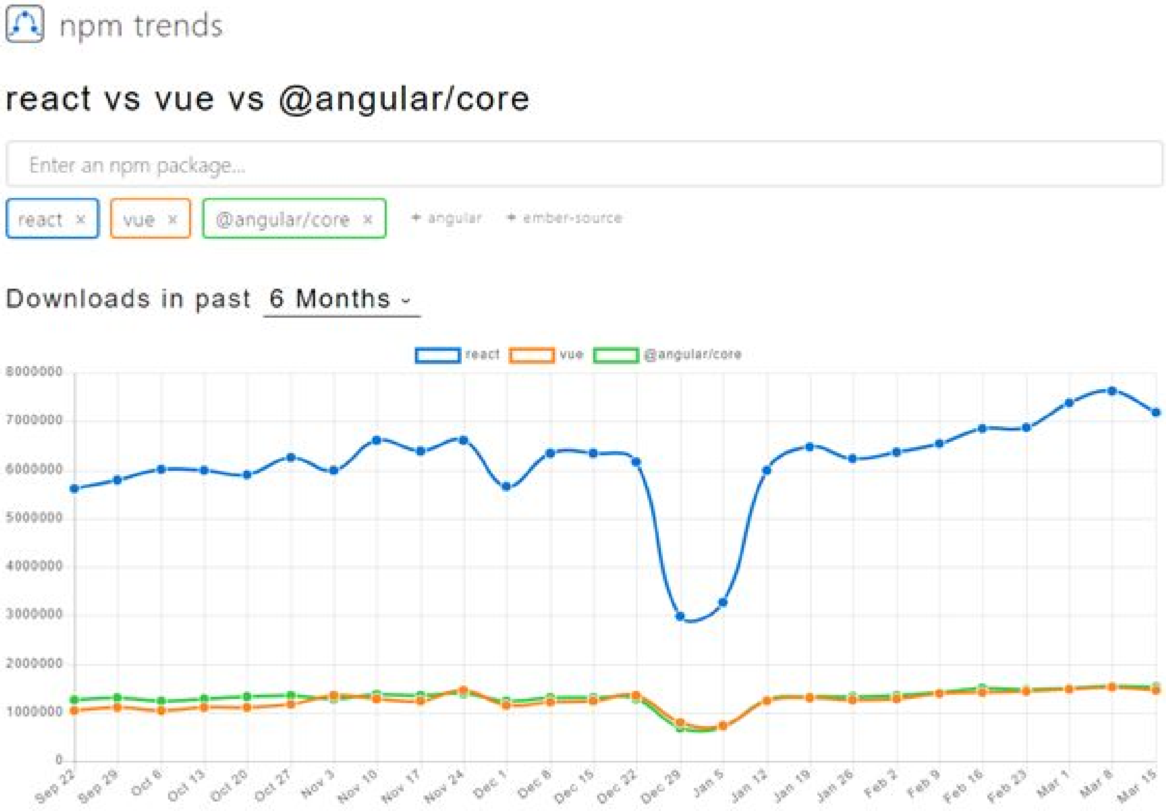


Image nr . 18 : Bar graph of most loved web frameworks in 2018 based on front-end frameworks (Neagoie, 2018)



npm trends

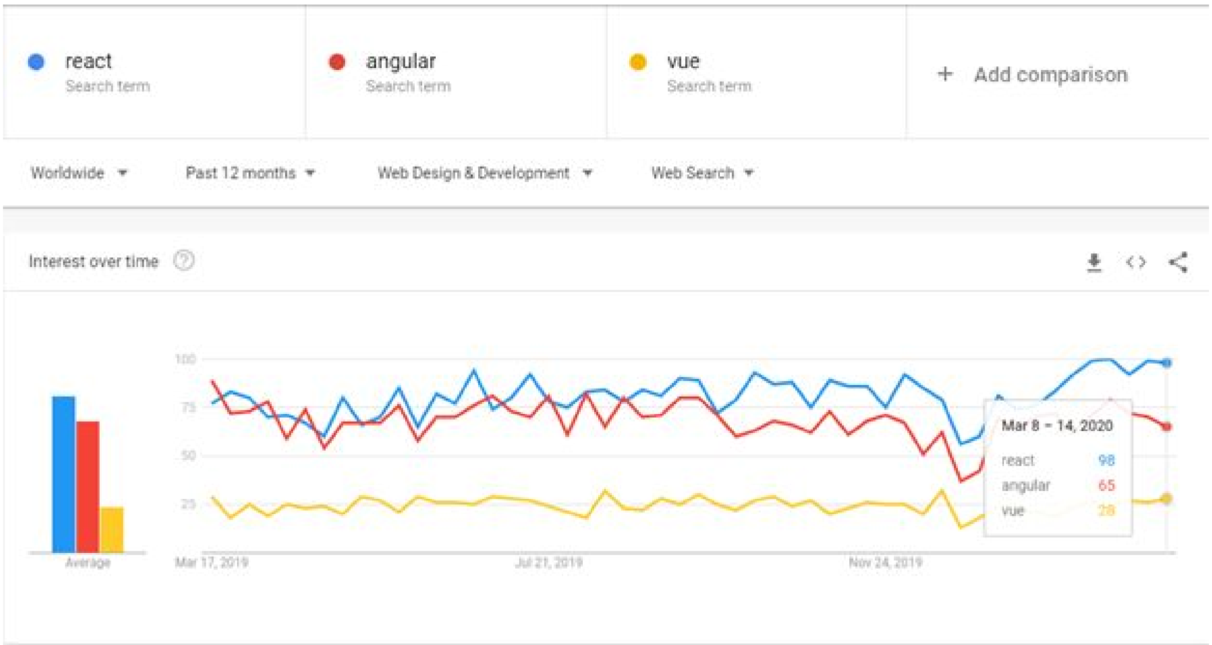
react vs vue vs @angular/core

Enter an npm package...

react × vue × @angular/core × + angular + ember-source

Downloads in past 6 Months ⌄

Image nr. 19 : Line *graph of most most NPM downloads in 2020 based on front-end frameworks (npm trends, 2020)*



● react Search term ● angular Search term ● vue Search term + Add comparison

Worldwide ▾ Past 12 months ▾ Web Design & Development ▾ Web Search ▾

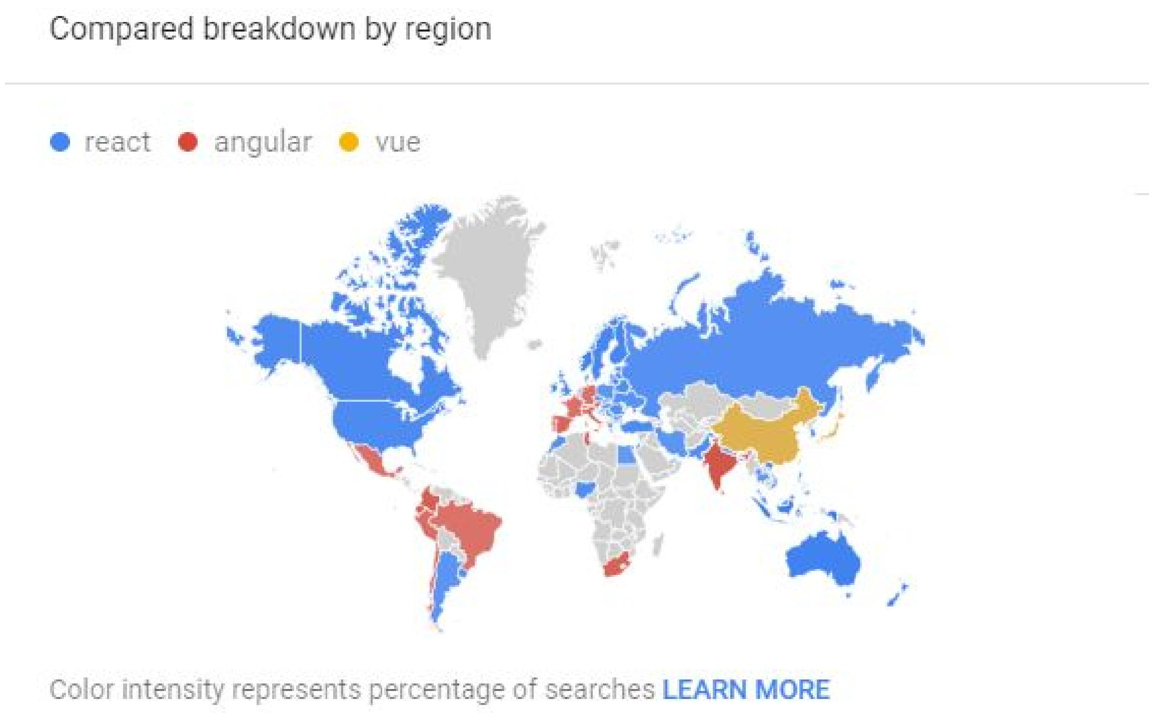
Interest over time

Mar 8 – 14, 2020

react	98
angular	65
vue	28

Image nr . 20 : Line *graph of most google searched front-end frameworks in 2020 (Google Trends, 2020)*

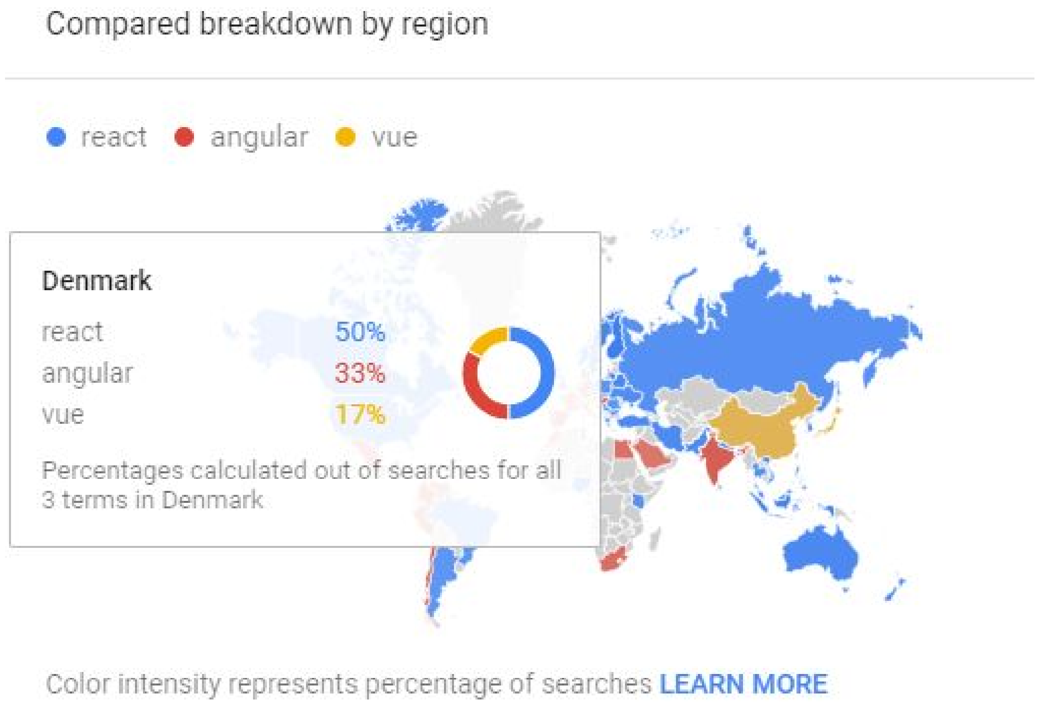
Moreover, preferences for frameworks also seem to be regional. Vue is the most popular in China due to its creator being from China who put in a lot of effort into having great Chinese documentation for the framework. But for Europe it looks to be a split between React and Angular. Angular is searched for more in the south-western side of Europe and React being more searched in the north-eastern part of Europe, including Denmark.



Compared breakdown by region

Color intensity represents percentage of searches LEARN MORE

Image nr . 21 : Regional *map graph of most google searched front-end frameworks in 2020 (Google Trends, 2020)*



Compared breakdown by region

Denmark

react	50%
angular	33%
vue	17%

Percentages calculated out of searches for all 3 terms in Denmark

Color intensity represents percentage of searches LEARN MORE

Image nr . 22 : Regional *map graph of most google searched front-end frameworks in 2020 with focus on Denmark. (Google Trends, 2020)*

To sum up, the chosen front-end framework was React. Because it had the most advantages and least number of disadvantages for this specific project. In addition, React is also a potentially very useful tool to master in the future for the current European job market.

4. Implementation

After deciding that React would be used for the client side of this application, a question arose: "How are the backend logic and data storage going to be handled?". In Web Development, developers are given tons of freedom when it comes to choosing and combining technologies. In this project, the choice made depended on the kind of solution to build, as well as personal preferences.

First of all, the greatest, overall motivation of the group seemed to be the will to learn and create something that would reflect the members' competences. This resulted in selecting a whole stack that allowed the team to work within frontend, backend and database management while using one common language: JavaScript.

The chosen stack ended up being the MERN stack, which stands for the following technologies:

- MongoDB: a document-based, NoSQL database.
- Express: a web framework for Node.js.

- React: a front-end JavaScript framework.
- Node.js: a JavaScript runtime environment.

These technologies did not only fulfill the needs of the team, but they also facilitated the workflow and collaboration between all the members. This is mostly thanks to the fact that only one language was used (as mentioned above) and that each of these technologies has a vast online community and is rather simple to set up (Ganguly, 2019).

However, these are not the only motives behind the team's choice. Out of all the technologies that could have been combined with React, Node.js seemed to be the most logical choice for this particular project.

The reasoning behind this statement is rather simple: combining React and Node.js is extremely convenient. Node.js offers a lightweight server-side for React apps, and the possibility to create a JSON-based API, which increases reusability (Kasundra P., 2020). The workflow becomes even easier with the addition of Express, which is also quite lightweight and adds, at the same time, necessary features in order to make working with Node.js more simple (Keinänen, 2018).

The choice of database, when working with Node.js, is almost obvious. Despite the fact that MySQL could also be a good alternative, MongoDB seems to be an extremely popular option in this particular case. For instance, one of the multiple reasons why this makes sense, is the combination of a JSON-based API (mentioned above) and JSON-based documents.

In relation to this project, more reasons for choosing MongoDB can be found. Considering that the solution had to be a "contacts-app with a chat function", speed and scalability were some of the factors that were taken into consideration, as well as flexibility and reusability, since the future of this product is quite unsure. This is also why a rigid data model was not considered crucial: not knowing for sure if more features are going to be implemented or modified, makes the need for polymorphic schemas more important than having fixed structures. This is exactly why MongoDB seemed to be the best database for this project.

4. 1 Structure and relationship between client- and server-side

The MERN stack gives much more freedom when it comes to structuring an application, compared to other stacks. In this particular case, the app was organized as such (visual representation: image nr. 23):

- The `client` folder contains the React app, where all the functional components, CSS files and the main HTML page can be found: in short, this is where the whole frontend resides. A particular aspect of this section is the presence of the `serviceWorker.js` and `manifest.json` files, which are the basis for PWAs (Progressive Web Apps) development.

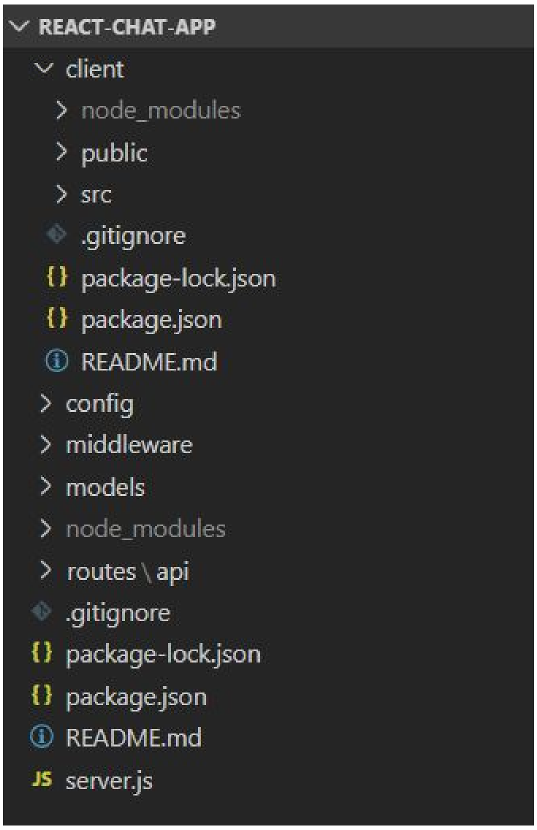
- \`config\` includes the server port, the local strategy used for authentication, a string used for sessions and two database strings.

- \`middleware\` contains the \`isLoggedIn\` middleware, used for authorization.

- \`models\` includes three files with one mongoose schema each. These schemas represent the skeleton of the objects which will be contained in the database collection in consideration.

- \`routes\` has a sub-folder called \`api\`: this is where all the API routing is handled.

- Lastly: \`.gitignore\`, \`package.json\` and \`package-lock.json\`, \`README.md\` and \`server.js\` (the main file of the application, which handles the connection to the database, the creation of the backend server, and more).



```
∨ REACT-CHAT-APP
  ∨ client
    > node_modules
    > public
    > src
      .gitignore
      package-lock.json
      package.json
      README.md
  > config
  > middleware
  > models
  > node_modules
  > routes \ api
    .gitignore
    package-lock.json
    package.json
    README.md
    server.js
```

Image nr.23: Folder structure

The overall architecture can be described as MVC-like (Model-View-Controller): the Model corresponds to the use of MongoDB and Mongoose (to model data); the View is represented by React on the client side; lastly, the Controller coincides with part of the server side (Wilson, Koroliova, 2018).

The concept behind the structure of this application, in fact, was to connect the React app to the MongoDB database through a RESTful API. For example, a POST request is handled in the following way.
On the client side, a dependency called "axios" is used for HTTP requests: a method has to be defined, as well as an API URL, an object with the data that has to be sent, and a callback. With this request, the data is sent to the API method with the corresponding URL. This method also takes a callback with the request and response as parameters: this is where the data is retrieved and modelled with the use of the "Mongoose", a MongoDB object modeling tool.

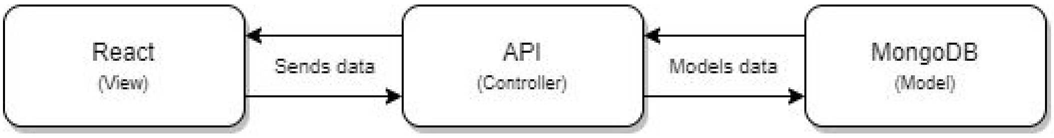


Image nr.24: MVC-like behaviour

4. 2. Database design

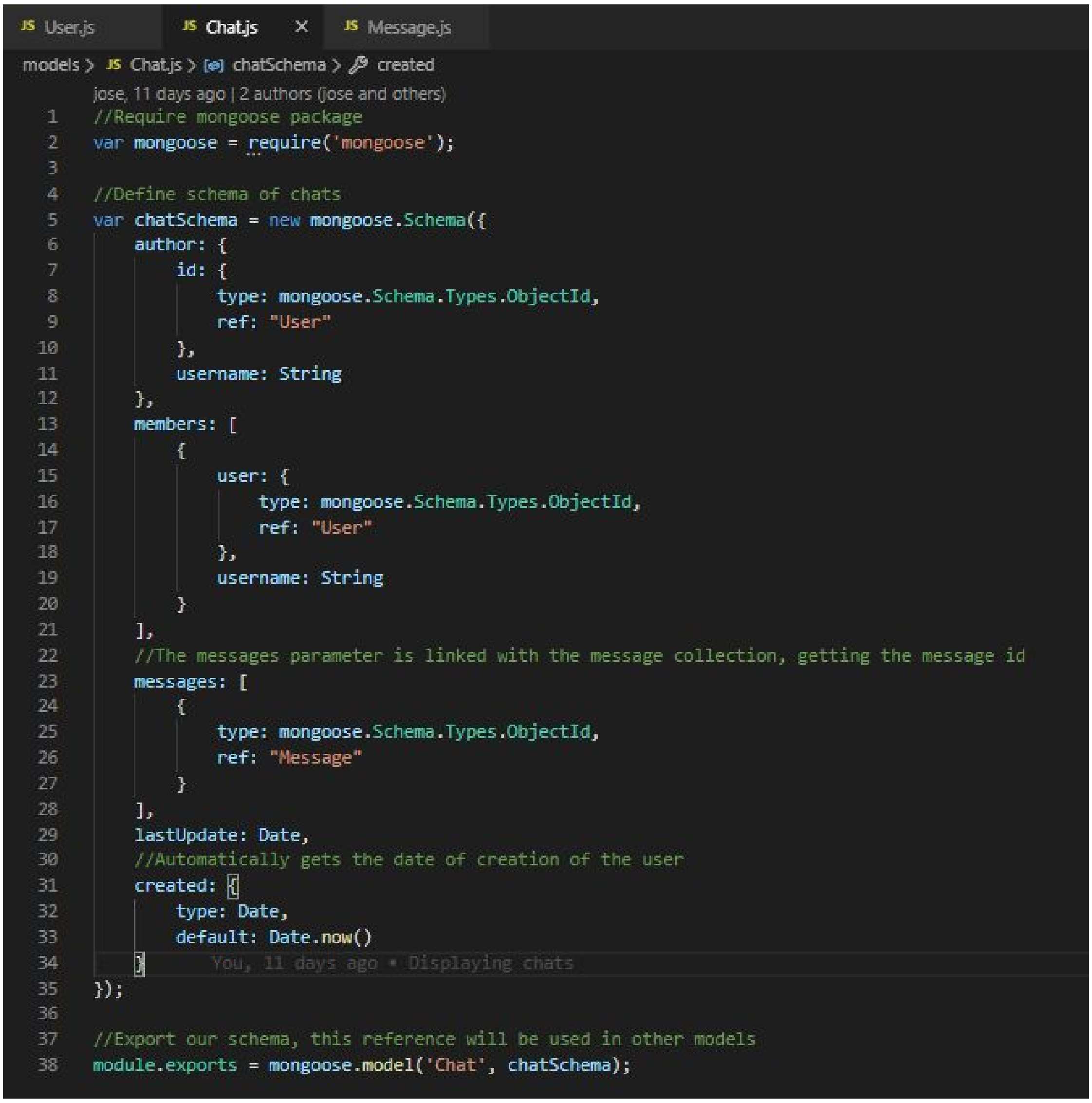
As stated in the previous section, the database used for this contacts-app was MongoDB. This allowed the group to design database schemas without worrying too much about the specific attributes that needed to be added to each model. The way that those schemas were created, was by having in mind the general skeleton of the model in consideration.

Three schemas were designed: User, Chat and Message. The relationship between these is done through embedding, which is possible thanks to the absence of many-to-many relationships: if the structure of this application had been more complex, referencing would have been a preferable method. The current models are structured as follows.

Users require a username, name, password, email and address, and they also contain a “avatarColor” attribute, a creation date and an array of “contacts”: objects which refer to existing users in the app. Each object is made up of a user id, a username and a nickname, which is empty at first.

Chats are, perhaps, the most complex out of all three schemas. Each chat consists of: an author (a user object with an id and username); an array of “members” (user objects); an array of “messages” (this is the only example of pure referencing in this application); and the “lastUpdate” and creation date attributes.

Lastly, messages only contain a string named “content”, an author (same as in chats) and a creation date.


User.js
Chat.js
Message.js
models > Chat.js > chatSchema > created
jose, 11 days ago | 2 authors (jose and others)
You, 11 days ago • Displaying chats

```javascript
//Require mongoose package
var mongoose = require('mongoose');

//Define schema of chats
var chatSchema = new mongoose.Schema({
    author: {
        id: {
            type: mongoose.Schema.Types.ObjectId,
            ref: "User"
        },
        username: String
    },
    members: [
        {
            user: {
                type: mongoose.Schema.Types.ObjectId,
                ref: "User"
            },
            username: String
        }
    ],
    //The messages parameter is linked with the message collection, getting the message id
    messages: [
        {
            type: mongoose.Schema.Types.ObjectId,
            ref: "Message"
        }
    ],
    lastUpdate: Date,
    //Automatically gets the date of creation of the user
    created: {
        type: Date,
        default: Date.now()
    }
});

//Export our schema, this reference will be used in other models
module.exports = mongoose.model('Chat', chatSchema);
```

Image nr.25: Example of mongoose schema - Chat

As far as the hosting of the database is concerned, two different databases are used: one for production and one for development. The database used for production is in a MongoDB Atlas cluster, whereas the one used for development was entirely up to the developer: some used another Atlas cluster, and others MongoDB Compass.

4. 3. The challenge of creating a full stack web app

As mentioned above, the MERN Stack (MongoDB, Express, React & NodeJS) has been chosen to create this Progressive Web Application.

The main challenge encountered by the group was having to deal with a full-stack environment with different technologies.

4. 4. Organization of the team

The team was divided in two main groups, frontend and backend, but they were not rigid. At the beginning the frontend team was mainly creating the basic views for the app and the backend team was setting up the database, the backend server and the authentication.

After this stage the roles of both teams were merged and the common goal became the connection between the server and client sides.

In this part the problems were the state management with React Hooks and getting the information from the forms in the frontend and sending it to the backend to do the request to the database afterwards.

4. 5. Authentication

The use of MongoDB was a challenge when it came to the authentication of the users.

Before MongoDB was selected as the database for this application, the use of Firebase was considered. This is because Firebase automatically manages the authentication system with "Firebase Authentication SDK" (Google - Firebase, 2020). Despite this advantage, MongoDB ended up being chosen for security and high performance within high traffic apps.

This decision made the creation of the authentication system more challenging, and it was done using sessions with the npm modules "bcrypt" and "passport".

When a new user signs up, the password is encrypted using salt/hash: the salt is created by bcrypt, and the hash is generated by mixing the password and the salt.

When logging in, instead, bcrypt was used in the opposite way: for decrypting the password. This is how the login system works: first of all, when the submit button of the login form is pressed, a post request to /api/users/login is made with the user's credentials. Then bcrypt is used to compare the user's password in the database and the password inserted by the user in the login form, in other words, it decrypts the password in the database and compares it with the password sent by the user. If the passwords correspond, *passport.authenticate()* is called to verify the user, and then returns the serialized user to the success page, if everything goes as planned.

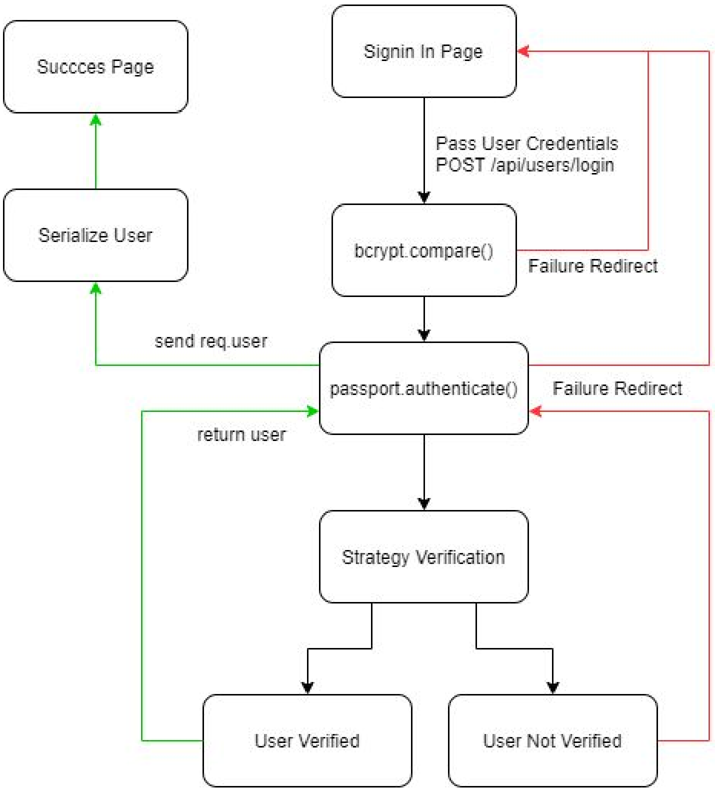


Image nr.26: Authentication System Diagram

4. 6. Deployment

Heroku has been chosen for the deployment of the chat app. This decision has been taken because Heroku is a free PaaS (Platform as a Service) and it supports NodeJS. But the most important characteristic is the continuous deployment via GitHub repositories, in this way the app can be updated with the last changes.

The app can be tested using the link on the front page (see “App URL”) and using the users on the chart.

Email	Password
test@jamt.com	qwe123
alessia@jamt.com	qwe123
jose@jamt.com	qwe123
maria@jamt.com	qwe123
tomas@jamt.com	qwe123

5. Key learnings

The team has learned a lot from the start of the project until the end of it. Some of the key learnings the students have learnt over this course of time are the following: team organization and the balance between challenges and delivering a fully functioning product.

5. 1. Team organization

Managing team organization with four students was a challenge, especially at the start of the project. But some key factors that the team learned along the way and helped a lot when working as a team include assigning roles and breaking down tasks and having an overview of the project.

Assigning roles helped the team members understand what their strengths and weaknesses were and assign tasks based on that. Understanding each other's strength and weaknesses was also very beneficial if a team member was having trouble with a certain task: then they knew who might be able to help them out based on the assigned roles.

In the start of the project, the team did not have an overview of all the tasks that needed to be completed. Not having a plan makes it hard to understand what exactly can be accomplished within a specific time frame. The team ended up over-estimating what could be accomplished in the given time and had to change the original project idea into a simpler project in order to finish on time. As plans changed, the team decided to write out all the tasks that needed to be done for the project to be completed. Having the tasks written out gave the team a good overview of what had to be done and made the project more manageable. It made it easier to assign tasks to each member of the group and understand what each member was currently working on.

5. 2. Balance between challenges and functioning product

As mentioned earlier in this report, one of the greatest motivations for the team was the will to learn. The team could have decided to go with Firebase, which goal is to help developers worry less about the server implementations by abstracting it and taking care of the app's back-end (Umeh, 2018).

But the team wanted to learn how to work with and gain experience with a MERN stack application, which was therefore the chosen stack for this project. The team's goal was to learn how to work with different technologies in a single application as the students had never created an application combining these technologies before. It was a challenge, but each member of the team gained new experiences working with the MERN stack and learned something new, and the team is therefore happy with the decision to work with the stack.

The team overestimated the capabilities of what could be completed in the time that was dedicated for the project, as stated earlier. This was partly because of the team's desires for challenges and learning new things. The team had planned to add quite a lot of extra features to the application in the beginning of the project, but later on realized that it was not a sustainable choice to add so many features in the time frame we had to work on the project. Since none of the students had worked with the features that were being considered to add to the project before, everyone would have to learn about them from scratch, which would take a lot of time. The team therefore learned that there needs to be a balance between learning something new and delivering a fully functioning product.

Instead of sticking to the original project idea, the team decided to choose fewer features and focus on making them well, instead of trying to do too much at once, which can lead to poorly designed and developed features. Even if the team did not implement all the features it had originally planned to, there is always a possibility to expand the application later on.

6. Conclusion

As stated earlier, the aim of this project was to build a web-application with a chat function which allows users to build contacts and connect with their friends. The results that the team obtained is a messaging application called JAMT where users can easily connect with friends, add contacts and create group chats. The team has learned a lot throughout the process of creating the application and considers the project to be successful, although there is still plenty of room for improvements and future expansions of the application.

7. Literature/bibliography

- Wilsom E., Koroliova I., 2018, *MERN Quick Start Guide*. Packt Publishing. Available through: https://books.google.dk/books?id=HnxeDwAAQBAJ&pg=PA1&lpg=PA1&dq=mern+mvc&source=bl&ots=0g5Uef6b9t&sig=ACfU3U0BgT9EadEBAtH2xF6ePVsaskXN5g&hl=da&sa=X&ved=2ahUKEwjjgP_o2p_oAhUMsaQKHd6rAsMQ6AEwCHoECAoQAQ#v=onepage&q=mern%20mvc&f=false

- Ganguly D.S., 2019, *Most Popular Technology Stack To Choose From Full Stack Vs. MEAN Stack Vs. MERN Stack In 2020*. [online] Available at: https://medium.com/datadriveninvestor/most-popular-technology-stack-to-choose-from-full-stack-vs-mean-stack-vs-mern-stack-in-2019-d12c0a17439a

- Keinänen M., 2018. *Creation of a web service using the MERN stack*. Bachelor of Engineering. Metropolia University of Applied Sciences. Available at: https://www.theseus.fi/bitstream/handle/10024/153461/Markus_Keinanen.pdf

- Kasundra P., 2020. *5 Reasons to use Nodejs with React for Web Development*. Simform. [blog]. Available at: https://www.simform.com/use-nodejs-with-react/

- Neagoie A., 2018. *Tech Trends Showdown: React vs Angular vs Vue.* [online] Available at: https://zerotomastery.io/blog/tech-trends-showdown-react-vs-angular-vs-vue/?utm_source=medium&utm_medium=react-angular-vue

- Shaumik Daityari. 2020. Angular vs React vs Vue: Which Framework to Choose in 2020 [online] Available at: https://www.codeinwp.com/blog/angular-vs-vue-vs-react/

- Facebook Inc., 2020 . React Documentation. [online] Available at: https://reactjs.org/docs/

- Evan You, 2020 . Vue Documentation. [online] Available at: https://vuejs.org/

- John Potter, 2020. Npm Trends. [online] Available at: https://www.npmtrends.com/

- Google Trends, 2020. Google Trends. [online] Available at: https://trends.google.com/

- MongoDB Inc., 2020. MongoDB Documentation. [online] Available at: https://docs.mongodb.com/manual/introduction/

- OpenJS Foundation, 2020. Node.js Documentation. [online] Available at: https://nodejs.org/en/docs/

- Axios, 2020. Axios (0.19.2). [npm package] Available at: https://www.npmjs.com/package/axios

- dcodelO, 2020. Bcryptjs (2.4.3). [npm package] Available at: https://www.npmjs.com/package/bcryptjs

- Holowaychuk TK, Wilson D.C., 2020. Express (4.17.1). [online] Available at: https://www.npmjs.com/package/express

- Shtylman R., Wilson D.C., 2020. Express-session (1.17.0). [online] Available at: https://www.npmjs.com/package/express-session

- LearnBoost, 2020. Mongoose (5.9.5). [online] Available at: https://www.npmjs.com/package/mongoose

- Hanson J., 2020. Passport (0.4.1). [online] Available at: https://www.npmjs.com/package/passport

- Hanson J., 2014. Passport-local (1.0.0). [online] Available at: https://www.npmjs.com/package/passport-local

- Walcher C., 2020. Passport-local-mongoose (6.0.1). [online] Available at: https://www.npmjs.com/package/passport-local-mongoose

- Umeh E., 2018. *Build a Serverless full stack app using firebase cloud functions.* [online] Available at: https://blog.usejournal.com/build-a-serverless-full-stack-app-using-firebase-cloud-functions-81afe34a64fc

- Google Firebase, 2020. Firebase Authentication. [online] Available at: https://firebase.google.com/docs/auth