## horizontale Linie



Vormats Helpcenter

Technical Documentation

**─**

last edited: 14-01-2021

Version: 1.3

# Content

[horizontale Linie 0](#_Toc92971897)

[Content 1](#_Toc92971898)

[1. Introduction 2](#_Toc92971899)

[2. System architecture 4](#_Toc92971901)

[2.1 Deployment diagram 5](#_Toc92971902)

[2.2 Mobile application 5](#_Toc92971903)

[Libraries: 6](#_Toc92971904)

[Frameworks 7](#_Toc92971905)

[2.3 Admin panel 7](#_Toc92971906)

[2.3.1 MobX 8](#_Toc92971907)

[2.3.2 Environment variables 9](#_Toc92971908)

[2.4 Server 9](#_Toc92971909)

[3. Deployment 11](#_Toc92971910)

[3.1 How to deploy to Heroku: 11](#_Toc92971911)

[3.2 Deploy platform & Domain name 11](#_Toc92971912)

[3.3 Environment Variables 12](#_Toc92971913)

[4. Database model 13](#_Toc92971914)

[5. Interfaces to other systems 14](#_Toc92971915)

[6. Known issues for the delivered product 14](#_Toc92971916)

[7. Recommendations for future improvements of the product 14](#_Toc92971917)

# Introduction

Hello and welcome to the Vormats Help Center! In this project we built a part of an existing application: the help center. If a user has some questions, we can help them right away!

For this project we built a mobile application that helps users with their questions. We’ve built four different options that the user can choose from:

* Articles (written by employees)
* Chatbot
* Chatting with an employee
* Contact by E-Mail

These four are all available in the android mobile project. There is also a website available for employees where they can for instance change the responses of chatbots and add, modify or delete articles. It even has chat functionality! In this website the employee can chat with their users. Of course, logging in is required.

Now that is a lot to handle, right? Don’t worry! This documentation will be your guideline. This document describes what we have made, how different applications are connected with each other, what dependencies are used, how you as a developer can change things if needed and much more!

# 

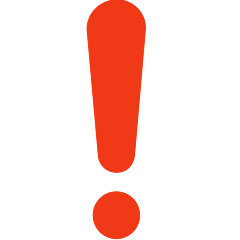
# 

# System architecture

For this application, there are four components used: Android, NodeJS, React and MongoDB. The heart of the system is the NodeJS server. The two applications are connected with this server. The server has a connection with the database: MongoDB. The android application and the React application make requests to the NodeJS server. Depending on the request, the server does crud-operations to MongoDB and sends a response back to the user in the form of JSON.

**How to change to another database?**

As for the database, there are many options you can choose from. Right now, the server uses MongoDB. However, it is very simple to change the database to another. Just change the file DB to the desired database. Don’t forget to change models to the required database syntax.

note: delete the library ‘mongoose’ if switching to another database with:

npm uninstall mongoose **or** yarn remove mongoose

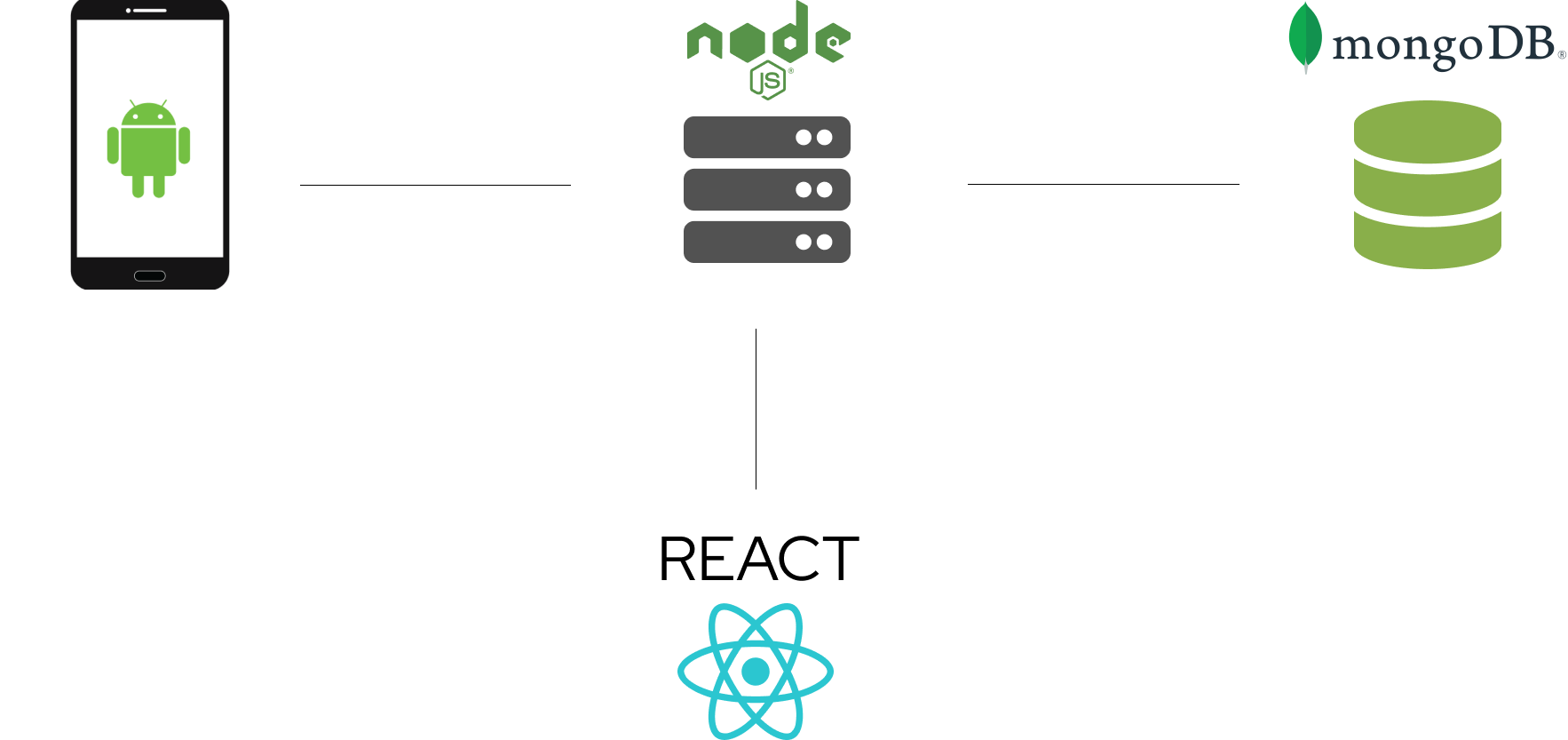


Figure 1: System architecture

## 2.1 Deployment diagram

In figure 2 there is a deployment diagram made. Here you can see how the applications are connected with each other and what dependencies are used in this project.

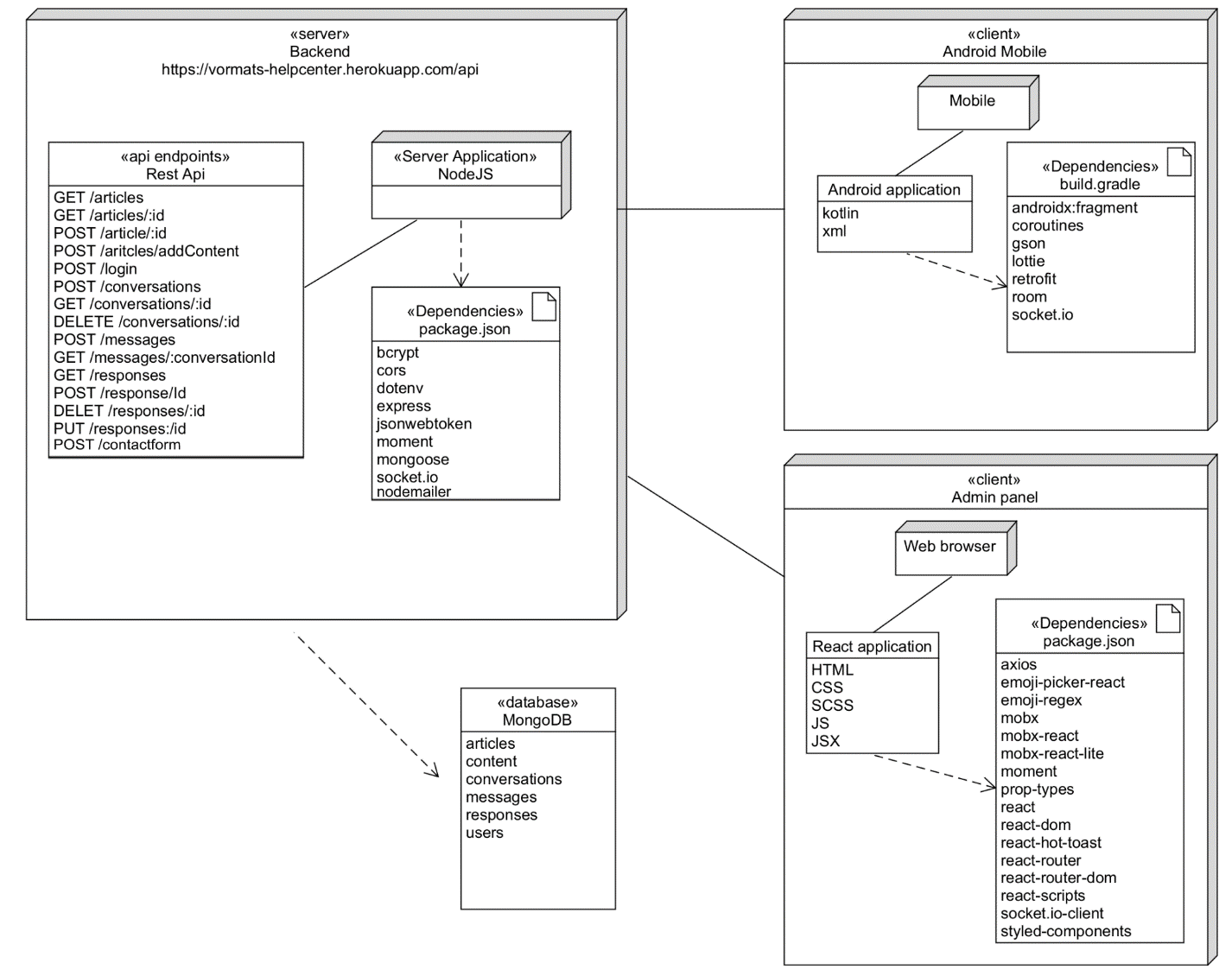


Figure 2: Deployment diagram including the android application, NodeJS server and the react application

## 2.2 Mobile application

The mobile application is built for android using Android Studio. The code is written in Kotlin for functionality and xml for visuals.

This mobile application is a package for an existing application, meaning that you would need to attach the module to your application to access it from there. Let’s not forget that it’s just a part of an application.

The architecture used in the project is Model - View - ViewModel (MVVM). We make use of viewmodels, because of the fact that viewmodels respect the lifecycle hooks of android. We don’t want any data leaks happening in the application.

In figure 2 you can see how the models are connected with each other.

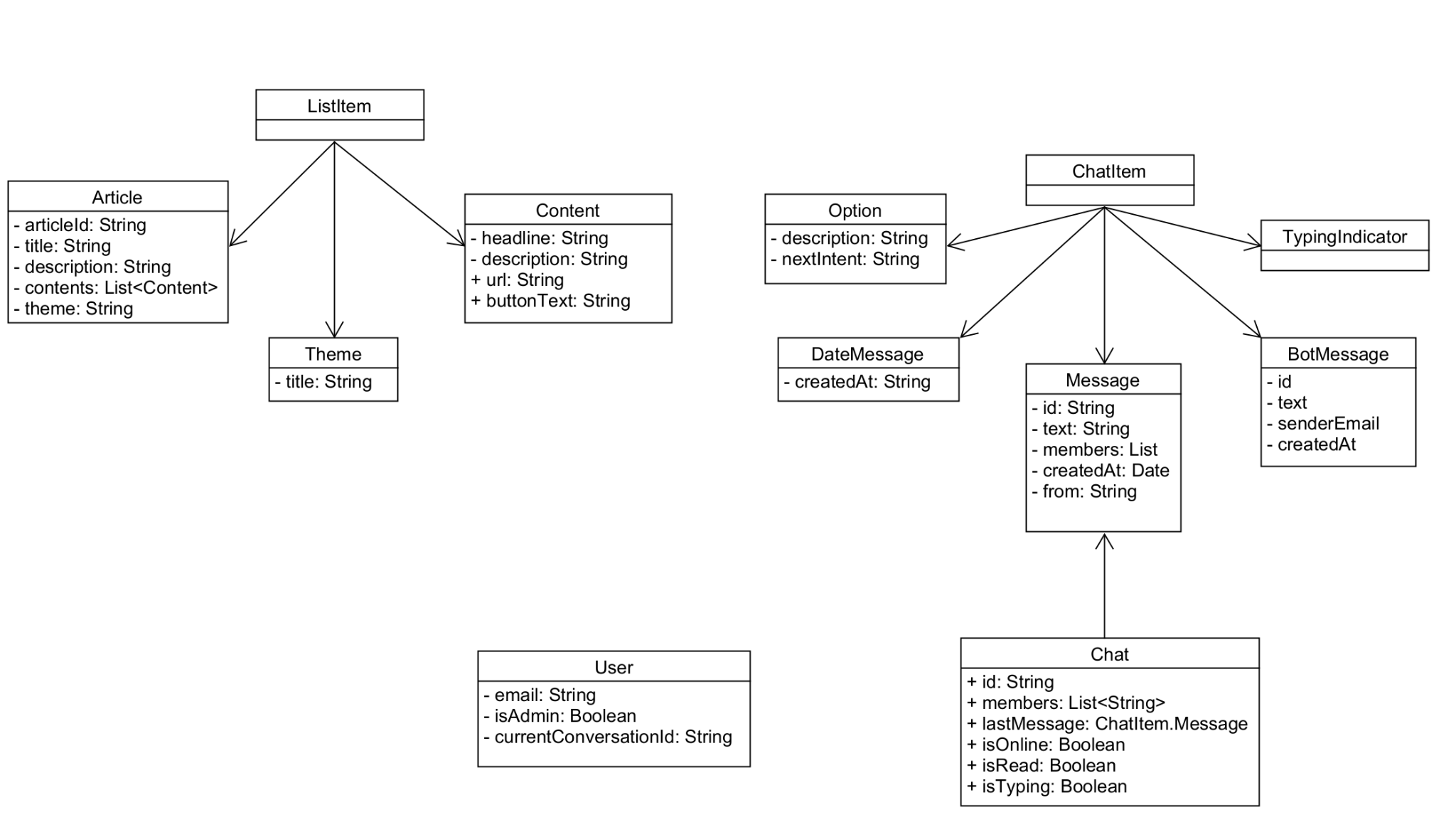


Figure 3: UML diagram

### Libraries:

As seen in Figure 3, there is an API endpoint available made in NodeJS. To connect with the backend and to do some http requests to the server, the application uses Retrofit. For development purposes there is also Retrofits HttpLoggingInterceptor implemented. With the help of this, the application can intercept every request and log it, which can be helpful when debugging.

Now the problem is that the android application doesn’t understand the fetched request. The response is in the form of Json while Kotlin expects objects. So, to convert json to Kotlin object and vice versa, the application uses the Gson library.

The requests are all asynchronous code. To handle async code, Kotlin can make use of coroutines. That’s what is used when handling responses in Kotlin.

To chat between employees and users uses sockets. That’s why the socket.io library is needed. For animations like ‘user typing..’, the Lottie library is being used.

Last but not least, the room library. This one is used for chatting with the chatbot. The messages are saved in a local room database.

### Frameworks

This project does not use android frameworks, as it hasn’t been deemed necessary.

## 2.3 Admin panel

For this project, there is also an admin panel available. React is being used as the frontend framework. It is more of a library, but people like to call it a framework. React uses JSX as its language, which is almost identical to plain JavaScript.

As for the CSS-framework, Material UI is being used for the components. You can compare this with Bootstrap. However, the thing about Material UI is that it is better compatible with React. Moreover, it has some rich features. Like several built-in components you can use in your website.

The admin-panel has several pages. A page for the chatting, a page for the chatbot and a separate page for the articles. To navigate between those pages, the react-router-dom library is used.

For the styling the library ‘styled-components’ is being used. If you’ve ever worked with React, you know that the JSX syntax can be quite messy (see figure 4). You can see that with plain html classes your whole file could be filled up with things that you don’t need to see. Styled-components library is the rescue.

Figure 4: Styled components comparison with plain html

Furthermore, the application uses necessary utils like MomentJS for date formatting, socket.io for web sockets and emoji-picker-react to send emojis.

### 2.3.1 MobX

The thing about React is, you must keep track of the state. We don’t want to do that manually. So for this, Mobx is used. Mobx is a library that makes state management simple and scalable. MobX uses things called stores. You can compare this with a data class of Kotlin.

Open the ‘stores’ file and as you can see every file is a class-based file. All classes extend from the BaseStore. This file contains the logic that all other stores should have. Another important store is the RootStore file. This file gathers all the other stores you want to use in the application. So when you want to make a new store and use the store in react components, you have to make an instance of the store in the RootStore.

To observe the data changes in the desired component, import the observer from mobx-react-lite and surround this with the exported component (see figure 5).



Figure 5: example of how to use obervers in React.

With this, MobX then knows which components are subscribed and wants to observe data changes. In figure 6 you can see how the stores are connected with each other.

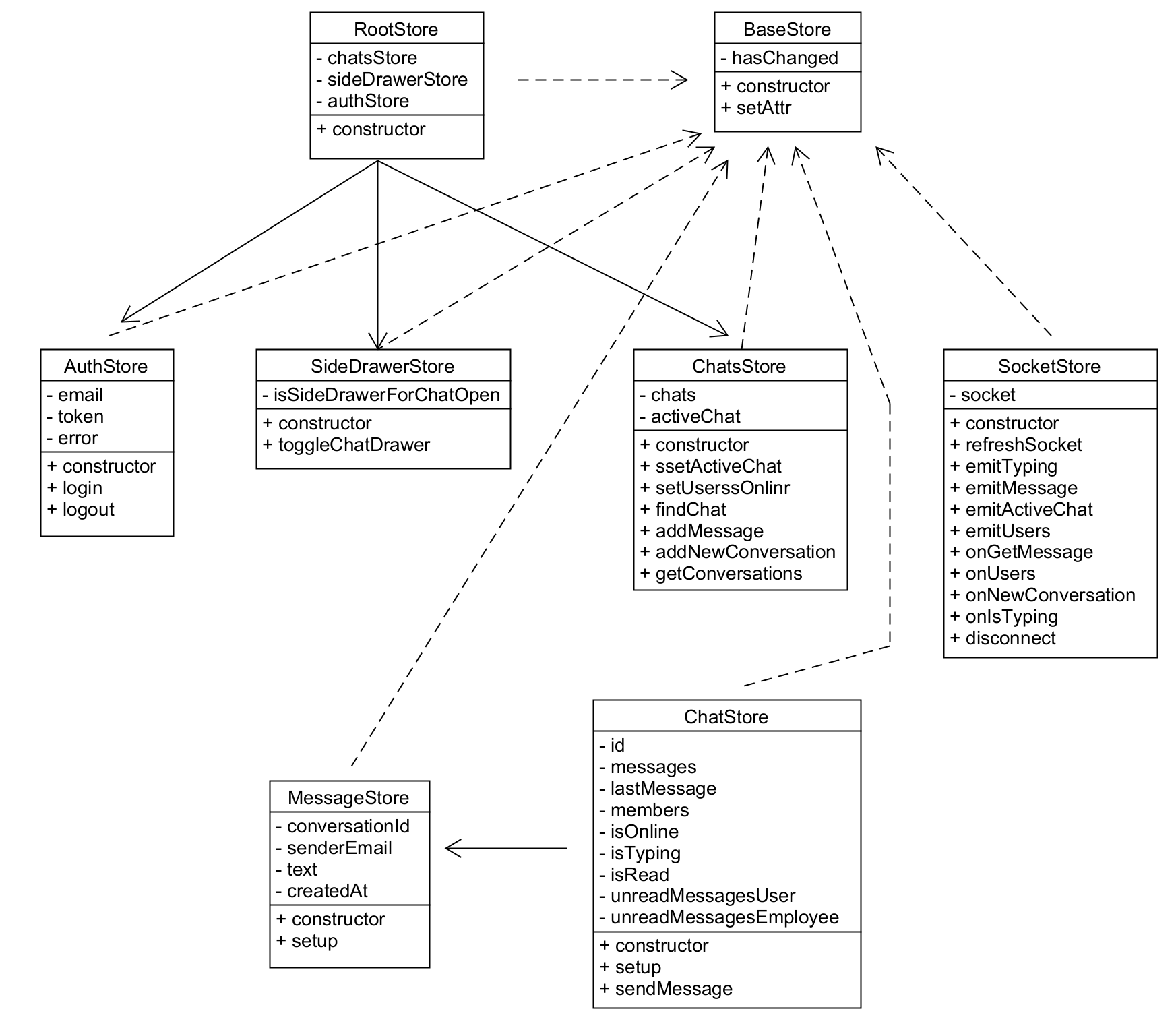


Figure 6: UML of stores in React

### 2.3.2 Environment variables

to use this project locally, add a .env file in the root file. In the .env file create a variable ‘REACT\_APP\_API\_ROOT’ and assign the value to the endpoint. This can be your Heroku endpoint or localhost. At the end, it should look something like this:

REACT\_APP\_API\_ROOT=http://localhost:8000

## 2.4 Server

There is an endpoint available where the mobile application as well as the react application can send their requests to: the backend server. The server is a simple express server written in NodeJS. The endpoints can be seen in Figure 2. These routes are dedicated to do CRUD-operations for conversations, messages, articles, users and chat responses, as well as sending mails through the contact form.

If a developer would start the NodeJS server locally and make some requests to the same server, everything would be fine with no errors. However, if you’d make a request from another port, you’d get an error as shown in figure 7. The well-known CORS error. To fix the problem we used a library called CORS. With the help of this, we allow requests from every other port as well.

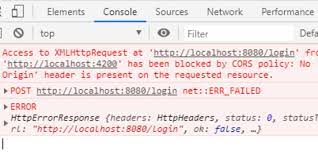
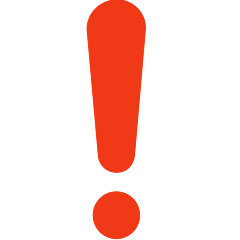


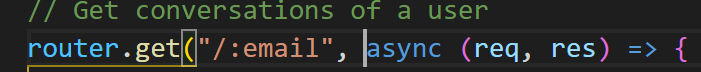
Figure 7: CORS error

note: the nodeJS project uses node version 14.17.0

The server is connected with a MongoDB server. That is where the library mongoose comes in. This makes it easier for us to handle requests to the database. More on that in ‘Database model’.

As for the users, the react application requires you to login to make use of our system. The users of the React system are at the moment separated with the users in the database of Vormats.

Bcrypt is used for hashing the password of the users. Json Web Tokens are used to encode and decode the content of the requests. Although it is already implemented, the application itself doesn’t use this at the moment. That’s because the database of the users is separated. Once there is a connection between the application and the database of Vormats, you could simply activate JWT by adding ‘verifyToken’ in the router function (see figure 8).



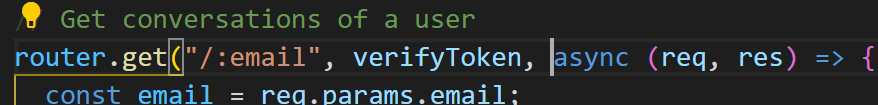


Figure 8: how to activate encoding of content

With our server, you have the ability to make use of the chat-system. Because of the fact that chatting happens with sockets, the chat-system uses the socket.io library. With this library it is possible to chat with each other.

The server also handles the messages coming in through the contact form. To do this the server uses the module Nodemailer to forward the messages to a pre-existing SMTP server which will then send emails with the content of the message to an employees’ mailbox.

Last but not least, the application uses some libraries that are just handy to use. For instance, moment is used for date formatting.

**Environment variables**

to use this project locally, add a .env file in the root file. We use the dotenv library to gather variables of the environment. For this project we need two variables: ‘MONGO\_URI’ and ‘SECRET\_TOKEN’. The mongo URL is for connection with the MongoDB. The secret token is for JWT. This can be anything you want. At the end, it should look something like this:

MONGO\_URI=mongodb+srv://<username>:<password>@cluster0.fxsjm.mongodb.net/myFirstDatabase?retryWrites=true&w=majority

SECRET\_TOKEN = vO2rMf2ATsSec3rEfdtKeYVfID4EOmaDeEasy30fffTEST

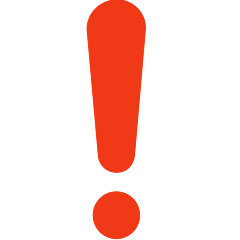
In the .env file create a variable ‘REACT\_APP\_API\_ROOT’ and assign the value to the endpoint. This can be your Heroku endpoint or localhost. At the end, it should look something like this:

REACT\_APP\_API\_ROOT=http://localhost:8000

# Deployment

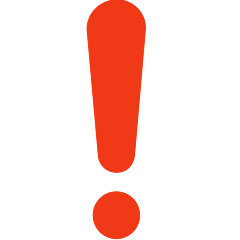
At the moment, only the NodeJS server is deployed. The React application only works locally. The server is deployed on Heroku. Due to the PAAS structure of Heroku it is easy and safe to deploy your application. Heroku provides features like hosting, DDOS protection, auto restart and a whole lot of other useful features to keep your service up and running.

The link to the backend is: <https://vormats-helpcenter.herokuapp.com/>

note: requests are done at uri: /api

## 3.1 How to deploy to Heroku:

1. set your remote Heroku git repository: git remote add Heroku <Git-URL>
2. git add .
3. git commit -m “<message>”
4. git push –force Heroku main

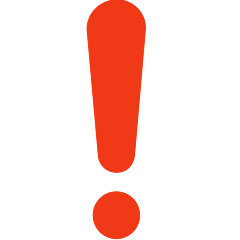
note: At the moment there is no CI/CD implemented.

## 3.2 Deploy platform & Domain name

However, if you want, you can redeploy it to another place and remove it from Heroku. Just follow a guide to deploy to the desired program.

Once you deploy it to anywhere else, you should change the endpoint url in all the android and the react applications.

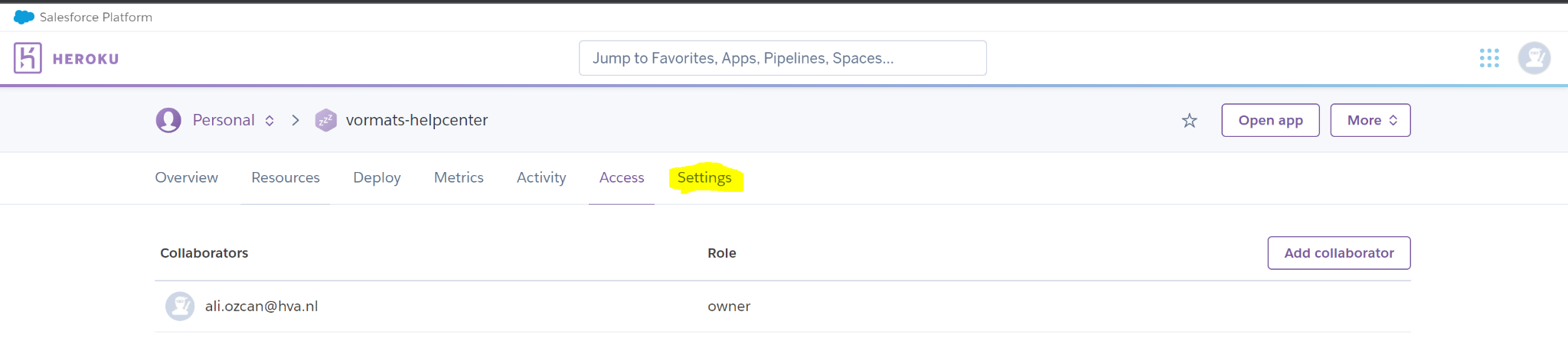
For the react application, you have to use a .env file. In this file change the ‘**REACT\_APP\_API\_ROOT**’ with your baseurl.

note: You also have to do this as well when you want to use a domain name!

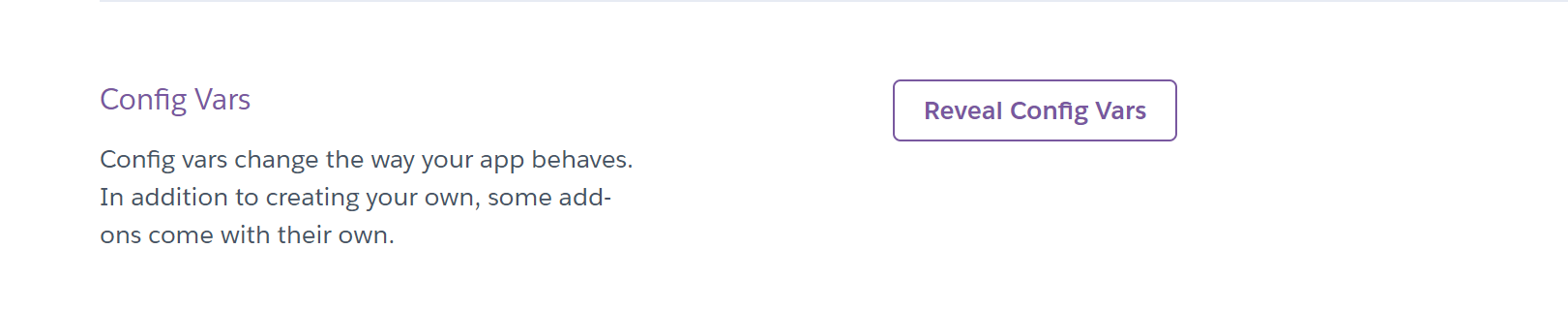
## 3.3 Environment Variables

While deploying, you don’t want to deploy your .env file. That’s why it’s ignored by git in the .gitignore file. Luckily for us, deployment platforms like Heroku can provide them for us. This tutorial makes use of Heroku; however every platform should provide environment variables.

When using Heroku, go to your application settings.



Scroll down and click on ‘reveal config vars’.



After that, copy the environment variables in your .env file and paste it in Heroku. Congratulations! You are done with the deployment .

## 3.4 Configuration of email settings

In order for the server to be able to send emails, an external SMTP server is used. The NodeJS server must be configured to connect to your SMTP server of choice in order to send emails with the correct address.

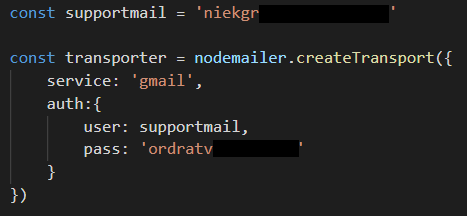
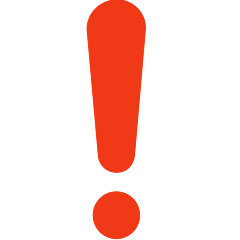


Figure 9: Settings for a connection to a SMTP server.

As shown in the figure above, the transporter has been configured to connect to gmail’s SMTP server for the sending of emails using the email of one of the team members. The password that’s used here is an authentication key generated by gmail in order to verify the connection. This constant can be adjusted to connect to your mail provider and their SMTP server.

note: Different providers may use different configurations in order to be able to connect. So please check with your email provider what parameters and settings are required.

# 

# 4. Database model

The models in the database are the following:

* Article
* Content
* Conversation
* Message
* Response
* Theme

The article, conversation and response are connected with each other. Response model as well as the Conversation model uses the Message model for their content.

This is also the same for Article, Content and the Theme model. Those are also connected with each other. For the full model see figure 10.

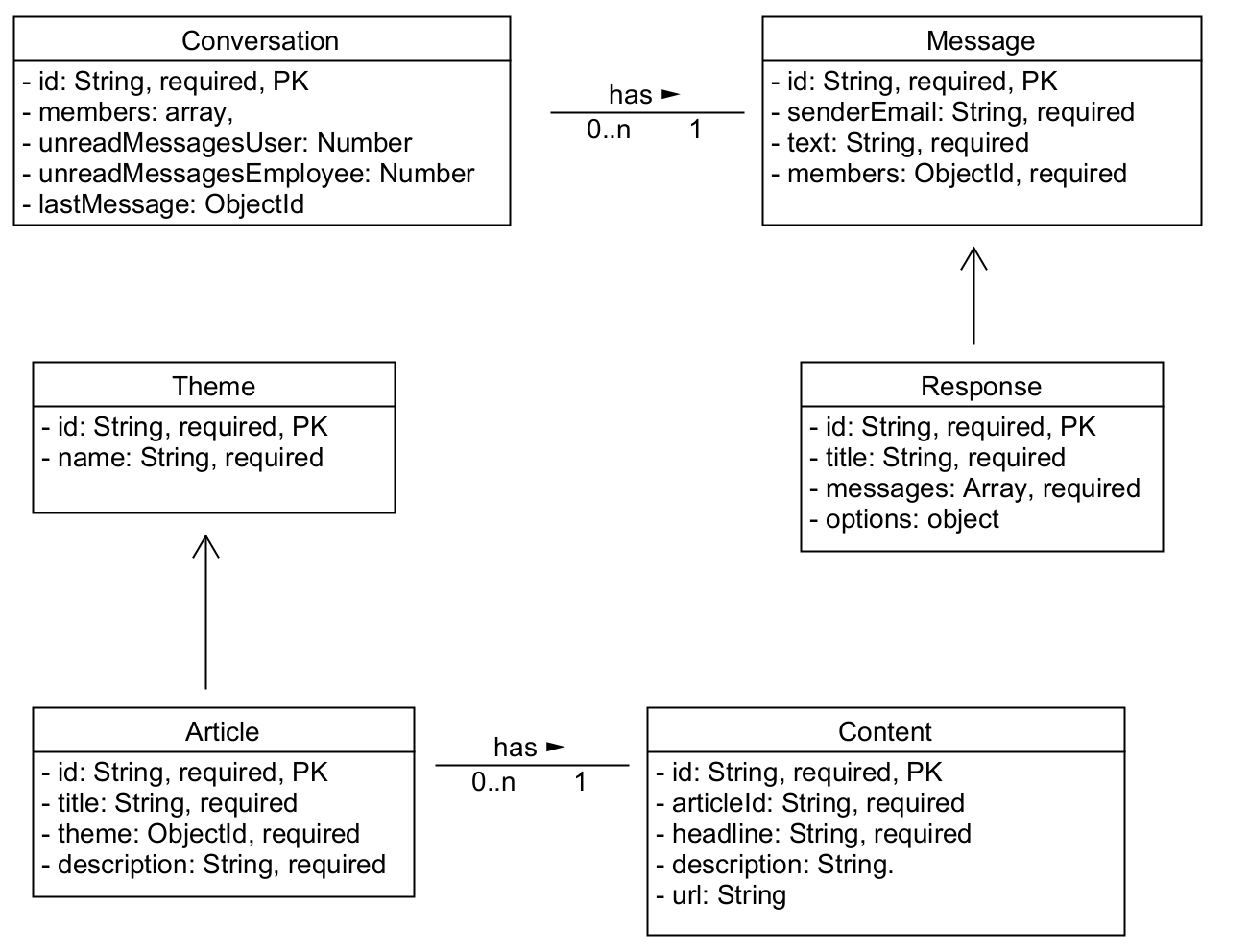


Figure 10: mongodb database UML

# Interfaces to other systems

In the mobile application the users can send an email to an employee. This with the help of an external SMTP server. The server uses an external SMTP server to send emails containing the messages sent through the contact form to a support employees’ email address. The external SMTP server can always be changed when desired. You are not dependent on one of the SMTP servers. Via the NodeJS server the emails are sent to a hard-coded email address. This can be changed anytime.

# Known issues for the delivered product

At the moment there are no issues found.

# Recommendations for future improvements of the product

There are always some improvements that can be done with any software product. This also applies to this project.

## 7.1 Notifications

In the near feature, the notification system should be implemented. At the moment, the employees don’t get a notification when a user has sent a message. If the employee isn’t logged in, the employee cannot know if and when a user sends a message. At the moment, he must login to the admin panel. It would be handy if both sides can get notifications.

## 7.2 Message type

At the moment, users can only send messages. Pure text and emoji. No photos nor videos. It is handy if an employee can send a video explaining stuff or vice versa. This can simply be done by adding a ‘type’ attribute in the models and make this an Enum. In the frontend handle the how to display the message by using the type attribute given from the backend.

If (type == ‘message’) show this

Elseif (type === ‘ímage’) show it like that

Else show it this way

## 7.3 Video’s in articles

In the articles, we can show the content needed to let the users know how to handle their problems. We can do this by text and images. However, we cannot show them any videos yet.

To handle this. Also add a ‘type’ attribute OR add another field that is called ‘video URL’ or a name like that. If you add the type, then you can only use one video OR image per content. If you add video AND image URL, you can show both in a content.