TALLINN UNIVERSITY OF TECHNOLOGY
School of Information Technologies

Martin Kalvik 164664IAPB

User Interface for Questions and Answers Platform

Bachelor's thesis

Supervisor: Ago Luberg
MSc

Tallinn 2021

TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Martin Kalvik 164664IAPB

Kasutajaliides küsimuste ja vastuste platvormile

Bakalaureusetöö

Juhendaja: Ago Luberg
MSc

Tallinn 2021

Author’s declaration of originality

I hereby certify that I am the sole author of this thesis. All the used materials, references to the literature and the work of others have been referred to. This thesis has not been presented for examination anywhere else.

Author: Martin Kalvik

24.05.2021

Abstract

The aim of this thesis is to create a user interface for a knowledge sharing platform. The application is in the format of questions and answers, with users being able to find and ask new questions with other users being able to answer them. The questions can be categorised into shards to allow for better organisation of content.

In the analysis section, some research has been made into what makes a good user experience and which principles one should follow while developing a user interface. Before heading into development, a prototype of what the UI could look like is made. Technological choices made before and during the development are mentioned along with how some of those technologies were used during development.

The result of the thesis is a user interface which allows users to manage shards, ask and answer questions with the ability to leave comments.

This thesis is written in English and is 40 pages long, including 7 chapters and 18 figures.

Annotatsioon

Kasutajaliides küsimuste ja vastuste platvormile

Antud lõputöö eesmärgiks on luua kasutajaliides teadmiste jagamise platvormi jaoks. Rakendus on küsimuste ja vastuste formaadis, kus kasutajatel on võimalus leida ja küsida uusi küsimusi, millele teistel kasutajatel on võimalik vastata. Küsimusi saab kategoriseerida, et võimaldada rakenduse sisu paremini organiseerida.

Analüüsi osas on uuritud kasutaja kogemuse ning heade tavade kohta, mida peaks järgima et luua kasutajale hea kasutajaliides. Enne arendusega alustamist valmistati prototüüp, et saada ettekujutus, milline võiks kasutajaliides välja näha. Mainitud on ka tehnoloogilised valikud, mis tehti enne arendust ning arenduse käigus ning kuidas neid tehnoloogiaid on kasutatud.

Lõputöö tulemuseks on kasutajaliides, mis võimaldab kasutajatel hallata sisu kategooriaid, küsida küsimusi, nendele vastata ning vajadusel ka kommenteerida.

Lõputöö on kirjutatud inglise keeles ning sisaldab teksti 40 leheküljel, 7 peatükki, 18 joonist.

List of abbreviations and terms

DRY	Do not Repeat Yourself
GUI	Graphical User Interface
JSON	JavaScript Object Notation
MVP	Minimum Viable Product
NPM	Node Package Manager
Shard	Grouping of content
UI	User Interface
URL	Uniform Resource Locator
UX	User Experience

Table of contents

Author's declaration of originality 3
Abstract 4
Annotatsioon Kasutajaliides küsimuste ja vastuste platvormile 5
List of abbreviations and terms 6
Table of contents 7
List of figures 9
1 Introduction 10
2 User experience 11
- 2.1 Graphical User interface 11
- 2.2 Principles 11
  - 2.2.1 Predictability 11
  - 2.2.2 Forgiveness 12
  - 2.2.3 Feedback 12
  - 2.2.4 Cognitive load 12

3 Technological choices 13
- 3.1 Vue.js 13
- 3.2 Vuetify 13
- 3.3 Vuex 14
- 3.4 Vue CLI 15
- 3.5 Vue Router 15
- 3.6 Eslint 15
- 3.7 Axios 15

4 Development 16
- 4.1 Requirements 16
- 4.2 Preparation 17
- 4.3 Web application 18
  - 4.3.1 Setup 18
  - 4.3.2 Routing 19
- 4.4 Views 21

4.4.1 Login ... 21
4.4.2 Questions ... 22
4.4.3 Question ... 23
4.5 Components ... 24
4.5.1 Sidebar ... 24
4.5.2 Modal dialogues ... 26
4.5.3 Comments ... 28
4.5.4 Question and Answer ... 29
4.5.5 Votes ... 30
4.6 Authentication ... 31
4.7 Reactive rendering ... 32
4.8 Theme ... 33
4.9 Deployment ... 33
5 Future plans ... 35
6 Summary ... 36
7 References ... 37
Appendix 1 – Non-exclusive licence for reproduction and publication of a graduation thesis ... 38
Appendix 2 – List of the used back-end API endpoints ... 39

List of figures

Figure 1. Vuex state management process pattern [6]... 14
Figure 2. The first version of the project UI, created in Figma..................................... 17
Figure 3. Figma view of the page containing questions based off the template............ 18
Figure 4. Currently mapped routes.. 19
Figure 5. The configuration of the nginx server located in /etc/nginx/nginx.conf in the server. 20
Figure 6. Login view. 22
Figure 7. Entirety of the questions view.. 22
Figure 8. Entirety of the question view. 24
Figure 9. Vuex action to parse the API response to the correct format for treeview component. 25
Figure 10. Modal dialogue window for creating a new question.................................. 26
Figure 11. The method that runs when clicking the save button................................... 27
Figure 12. A Vuex action that executes an axios get request to get the question data... 28
Figure 13. The card component of a posted answer with the new comment form active. 29
Figure 14. Object returned by making a POST request upon voting. 30
Figure 15. Method that runs when clicking the downvote button in the Voting component. 31
Figure 16. The subscriber function that listens for SET_TOKEN mutation calls.......... 32
Figure 17. Middleware function that redirects the user if not authenticated................. 32
Figure 18. Content of the .gitlab-ci.yml file.. 34

1 Introduction

One of the cornerstones to success in a collaborative effort is for the individuals involved to share their knowledge and experiences with each other or look for answers from someone better equipped to answer them. This could of course be done by posing said questions to people directly, however that would not make the organisation's best problem-solving experiences reusable.

One solution to that would be to implement a knowledge sharing platform so that asking questions would be as simple as possible and already existing information would be easily available. This would lead to better and faster decision making when facing a particular problem, more efficient training of personnel and spreading information widely across an organisation.

The aim of this thesis is to create a user interface for such a platform, where questions can be asked, grouped and accessed by people in an organisational environment while trying to provide a good user experience.

2 User experience

User Experience refers to the feeling the user has while browsing through an application, product or webpage, considering how easy the application is to navigate, whether the content displayed is relevant to the user and how effortless the product is to use overall. [1]

Designing a UX primarily focuses on one thing: the user. Meaning that a good UX tries to make the journey through an application as smooth as possible and the user feel like they are accomplishing their tasks efficiently. This is accomplished by doing research into the needs of the target audience of a product, creating a design strategy after those needs are understood and then creating prototypes to convey their ideas to the people that are responsible for executing said ideas [2].

2.1 Graphical User interface

The GUI is the graphical layer of an application. It consists of all the things the user can see, for example buttons, navigation bars, text fields and all the other things that are visible when visiting a site [2]. While UX design takes care of how the user should move through an application, the UI design is responsible for making those components that the user interacts with and to make them feel reliable and appealing for the eye.

2.2 Principles

Regardless how an applications UI may look like, it should still follow some basic principles of a well-designed UI.

2.2.1 Predictability

A good UI should allow users to draw from their past experiences from interacting with different websites, thus making it easy to predict what a certain button or action does. It also helps the user reach their goals on the webpage quicker.

A good way to implement this is to give visual ques for every action, whether it be in a short text form explaining what a certain action will do or using familiar icons that are generally used across multiple platforms. In addition, the UI should be consistent in how certain components are used so that the user can get used to them and with repeated usage will already know the outcome of certain operations [3].

2.2.2 Forgiveness

The user should never be afraid to explore the possibilities of a website. This can be achieved by making every possible user action reversible, instilling a sense of safety in users that whatever their next move might be, they will always be able to return to the point they started from [3].

2.2.3 Feedback

Providing feedback to the user about every action also gives a sense of reliability of the product. Whether it is something simple like the cursor changing to a pointer when an element on a page is clickable or something a little bit more complicated like showing a loading bar with the time remaining to indicate that the system has not gotten stuck.

The same principle applies when the user does manage to find themselves doing something that is not intended or allowed by the system. Its then important to let the user know what went wrong or what is the correct way to do achieve their goal. For example, the system should indicate to the user that their input is invalid in a way that at the same time tells them what the correct format would be [3].

2.2.4 Cognitive load

The information relayed to the user should be relevant to accomplishing their task. Often an average user, overwhelmed by irrelevant data, would rather turn away from the application instead of staying and trying to work their way around the unnecessary clutter.

A way to counter that would be to keep in mind the principles of content organisation. Separating similar content from each other into easily processable chunks, numbering items and using headings and prompt text when needed just to name a few [3].

3 Technological choices

This section gives an overview of the technological choices made before and during the development process.

3.1 Vue.js

The choice of which JavaScript framework to use was primarily between either Vue.js, React.js or Angular.js. While react is still by far the most popular framework used in front-end development and all three offering very similar features, the decision ultimately landed on Vue [4].

The choice for Vue.js was made due to it being a lightweight open-source framework suitable for smaller projects, simpler learning curve, steadily growing popularity and the easier to understand syntax [5].

3.2 Vuetify

In order to make development easier it is generally a good idea to use a UI component library that provides some pre-built components that are easy to modify to the specific requirements of a project.

There are plenty of options to choose from and ultimately the choice comes down to personal preference. However initially only BootstrapVue and Vuetify were considered. The reason being that the author has had a little bit of experience with Bootstrap before but Vuetify having the built-in material design.

The decision was made in favour of Vuetify due its design and the similarity of the syntax to Vue. The documentation for Vuetify components API is also extensive with examples of each available component provided, with some variations based on the properties available to the component.

3.3 Vuex

Vuex is a state management pattern and library for Vue.js applications. It is used to manage states across all the components used in an application with which can only be mutated in a predictable fashion [6].

Initially the need for Vuex came up in order to improve the user authentication process, however it was later discovered that it could also help with the reactive rendering of components.

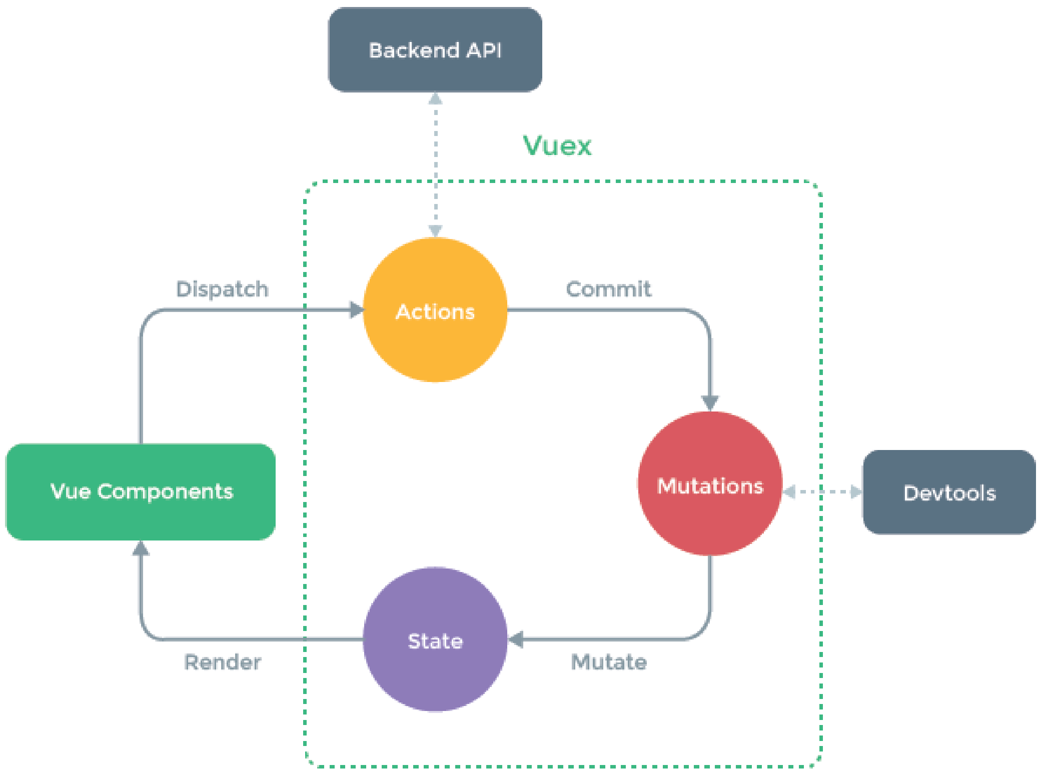


Figure 1. Vuex state management process pattern [6].

As described in Figure 1, the Vue component dispatches a Vuex action, for example a request to the back-end API. The response from the request is then committed to a mutation which essentially makes a change to a state. That state can then be referenced by multiple components across the project, which allows reactive rendering.

3.4 Vue CLI

Vue CLI is a standard tooling system for Vue projects. It ensures that the build tools that make up a Vue project run together smoothly so that it would reduce the time it takes to setup a project configuration manually [7].

The choice to use Vue CLI was made due to it being a standard when creating a new project and the fact that it builds a new project structure automatically with the necessary files. It also enables the use of the *vue serve* in the command line which allows to see the GUI on the localhost domain.

3.5 Vue Router

Vue router is the official router for Vue.js [8]. It simplifies the routing between different Vue components by mapping them to routes and letting the router know at what point it needs to render the components. The choice for the router was also made due to it being a standard practise for making routing more scalable and understandable.

3.6 Eslint

ESLint is used for detecting problems in the JavaScript code. This helps the developer to keep the style of the code consistent throughout the project. It can also be set up with custom rules if needed. The choice to use ESLint was made to detect errors and to keep the look of the code constant and it is something that is Vue CLI offers by default when creating a new project.

3.7 Axios

Axios is an HTTP client that provides a single API that allows making XMLHttpRequests from the browser to the server. The differences between Axios and the Fetch API are very minute and everything that is already built-in in Axios can be reproduced with fetch with a few additional steps.

The decision to use Axios was up to the authors subjective opinion on which was more understandable and it already having a set of features out of the box.

4 Development

This chapter goes over the development process. The back-end side requirements of the application, the preparation process prior to starting development and then the development itself.

4.1 Requirements

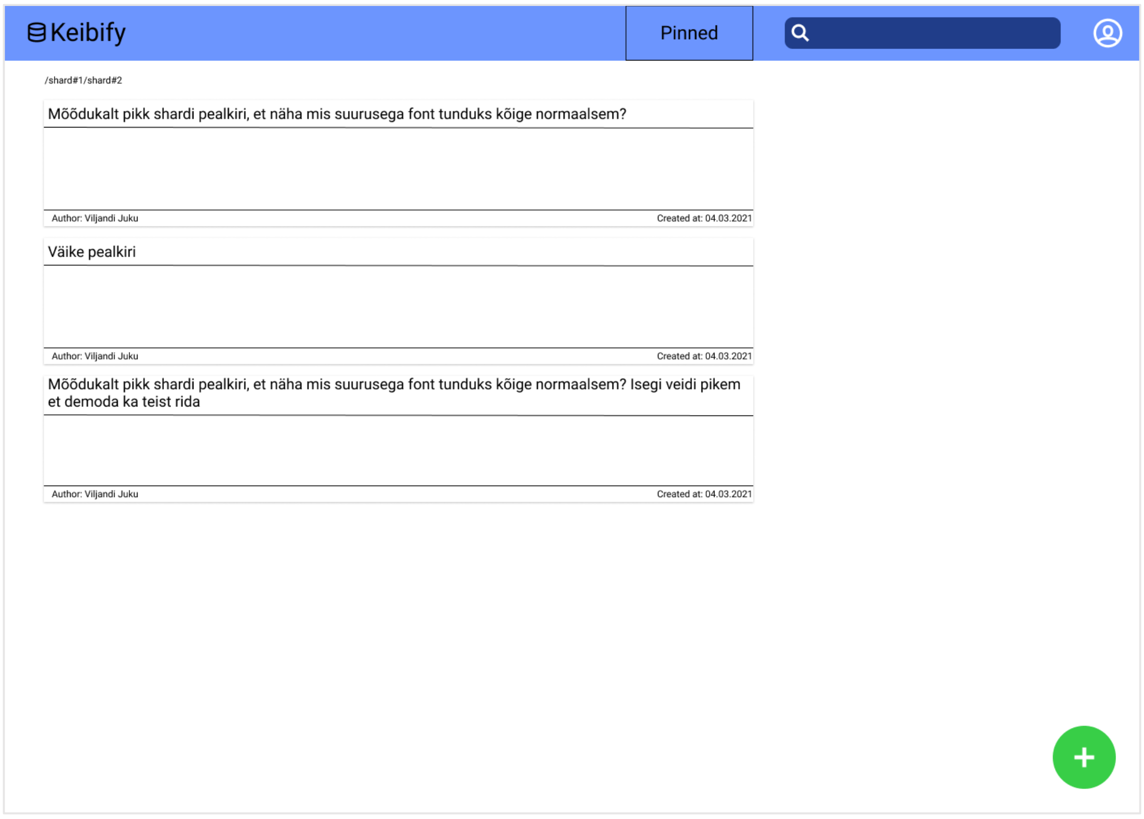
The requirements that the server-side application provided have not all been implemented in this project. However, the requirements in the following list are the ones that have been applied to make an MVP prototype.

- User can log in
- User can log out
- User is logged out if token is invalid
- Create, edit and delete a shard that groups questions
- Create a new question
- Add an answer to a question
- Add a comment to either a question, answer, or an existing comment
- Vote on either a question, answer or a comment.
- Add or remove an accepted answer for a question. Only one can be added per question
- Follow a piece of content to get notifications when updates are made. At the time of writing this document, the notifications feature is not yet implemented.
- User can search for questions

4.2 Preparation

In order to prepare for development, a mock-up UI solution was made using a website called Figma. The application was made for creating a design in a real-time collaborative environment.

The first draft of the prototype displayed in Figure 2 was quickly scrapped, since it looked clunky from the start and did not make use of some of the features that the website offers.



/shard#1/shard#2

Mõõdukalt pikk shardi pealkiri, et näha mis suurusega font tunduks kõige normaalsem?

Author: Viljandi Juku — Created at: 04.03.2021

Väike pealkiri

Author: Viljandi Juku — Created at: 04.03.2021

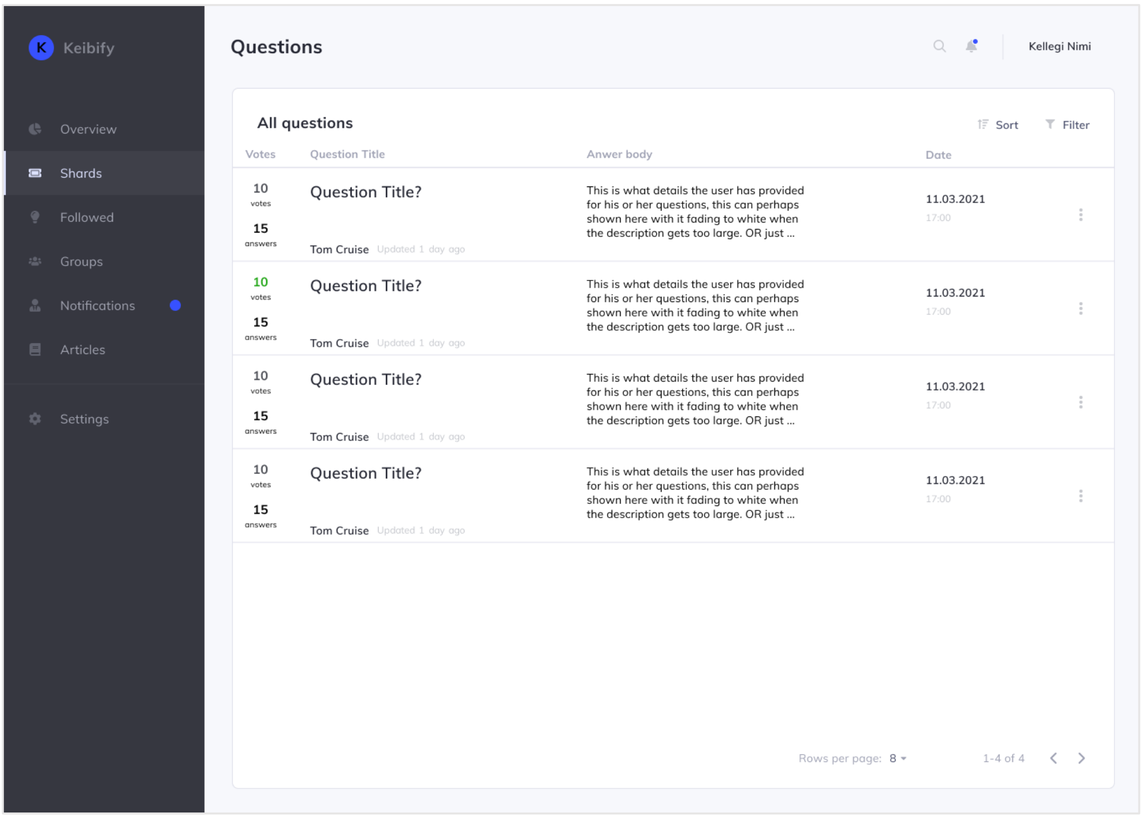
Mõõdukalt pikk shardi pealkiri, et näha mis suurusega font tunduks kõige normaalsem? Isegi veidi pikem et demoda ka teist rida

Author: Viljandi Juku — Created at: 04.03.2021

Figure 2. The first version of the project UI, created in Figma.

The initial mock-up was rushed and did not consider any of the principles of UX design mention in section 2.

Also, rather than creating a design from the ground up, a template was found among the various free resources that designers have made using Figma [9]. The template gave a rough idea of how the UI could look, and it was then modified to make a prototype which is displayed on Figure 3, that would fit the requirements of the project.



Questions

All questions

Votes	Question Title	Anwer body	Date
10 votes 15 answers	Question Title? Tom Cruise Updated 1 day ago	This is what details the user has provided for his or her questions, this can perhaps shown here with it fading to white when the description gets too large. OR just ...	11.03.2021 17:00
10 votes 15 answers	Question Title? Tom Cruise Updated 1 day ago	This is what details the user has provided for his or her questions, this can perhaps shown here with it fading to white when the description gets too large. OR just ...	11.03.2021 17:00
10 votes 15 answers	Question Title? Tom Cruise Updated 1 day ago	This is what details the user has provided for his or her questions, this can perhaps shown here with it fading to white when the description gets too large. OR just ...	11.03.2021 17:00
10 votes 15 answers	Question Title? Tom Cruise Updated 1 day ago	This is what details the user has provided for his or her questions, this can perhaps shown here with it fading to white when the description gets too large. OR just ...	11.03.2021 17:00

Rows per page: 8 ▾ 1-4 of 4

Figure 3. Figma view of the page containing questions based off the template.

Currently the UI solution of this project does not have the same kind of feel as is conveyed by the prototype, however it does share some similarities. The aim of making the prototype was not to then identically replicate it in the final product but rather getting a general direction on what kind of components to use.

4.3 Web application

This section covers how some of the technologies were used and gives a short overview of some of the components in the application.

4.3.1 Setup

To make the creation of a new Vue project as simple as possible, Vue CLI was used. This allows to scaffold a new project with just one single *vue create appname* command while also providing the possibility to install some additional dependencies right at the start. The dependencies chosen when creating the project were the Vue router and ESLint with the standard configuration.

4.3.2 Routing

In order to navigate throughout the application, the view components need to be mapped to the Vue router to let the application know, which view component it needs to render based on the route. To do that, if the project was created with Vue CLI, a router directory already exists containing an *index.js* file, where the mapping will take place. To map a view to a route some parameters need to be specified as seen in Figure 4.

```
const routes = [
  {
    path: '/',
    beforeEnter: authenticated
  },
  {
    path: '/shard/:id/edit',
    name: 'shardEdit',
    component: ShardEdit,
    props: true,
    beforeEnter: authenticated
  },
  {
    path: '/questions/:id',
    name: 'questions',
    component: Questions,
    beforeEnter: authenticated
  },
  {
    path: '/question/:id',
    name: 'question',
    component: Question,
    beforeEnter: authenticated
  },
  {
    path: '/searchResults',
    name: 'searchResults',
    component: SearchResults,
    beforeEnter: authenticated
  },
  {
    path: '/login',
    name: 'login',
    component: Login
  }
]
```

Figure 4. Currently mapped routes.

The path parameter specifies the URL where the component will be rendered. In order to achieve dynamic routing some parameters need to be added to the URL of the route. For example, if the user clicks on a shard, the questions for that specific shard should be rendered so the shard id is added to the questions route.

The name parameter defines the name of the route which can be later used when declaring the destination of a route and the component parameter defines which view component will be used. There is an additional *beforeEnter* navigation guard displayed in Figure 4 which will guard the navigation by redirecting or cancelling it, based on whether the conditions provided are met [10]. The *authenticated* condition will check if the user is currently authenticated and can access the route, if not then they will be redirected to the login page.

The router also makes use of the history mode, which replaces the Vue-routers default hash mode. The hash mode puts a hash into the URL to simulate a full URL so that the page is not reloaded when it changes. However, using the history mode requires adding a fallback route to the server in case the URL does not match any existing assets. So, the *nginx.conf* file located in */etc/nginx* directory of the server was modified with a few additional lines as described in Figure 5, to set the default index page in case the URL entered does not exist.

```
server {
    listen 80;
    server_name dev.keibify.com;
    index index.html;
    location / {
            root /var/www/html;
            index index.html;
            try_files $uri $uri/ /index.html;
    }
}
```

Figure 5. The configuration of the nginx server located in /etc/nginx/nginx.conf in the server.

4.4 Views

There are currently five views available:

- Login
- Questions
- Question
- Edit shard
- Search results

There are two static components shown across all views except the login page. Those are the navigation sidebar on the left which contains all the shards the user can navigate through, and the top navigation bar containing the current username and the search button with plans to also contain the notifications button.

The plan for the notifications dropdown list is to include all of the new notifications the user has received on the items that they are following. It would be possible for the user to click on a notification which would take them to the related item. It should also be possible to mark all the notifications as read; in case the user follows many items, and it becomes too difficult to keep up with them all.

4.4.1 Login

The login view displayed in Figure 6, consists of only one form contained in a Vuetify card component to acquire the user's login credentials and then redirect them to the questions view by clicking the login button. However, as shown in Figure 4, the view after logging in will only be rendered if the *authenticated* condition is met. If the condition is not met the user will be redirected back to the login view.

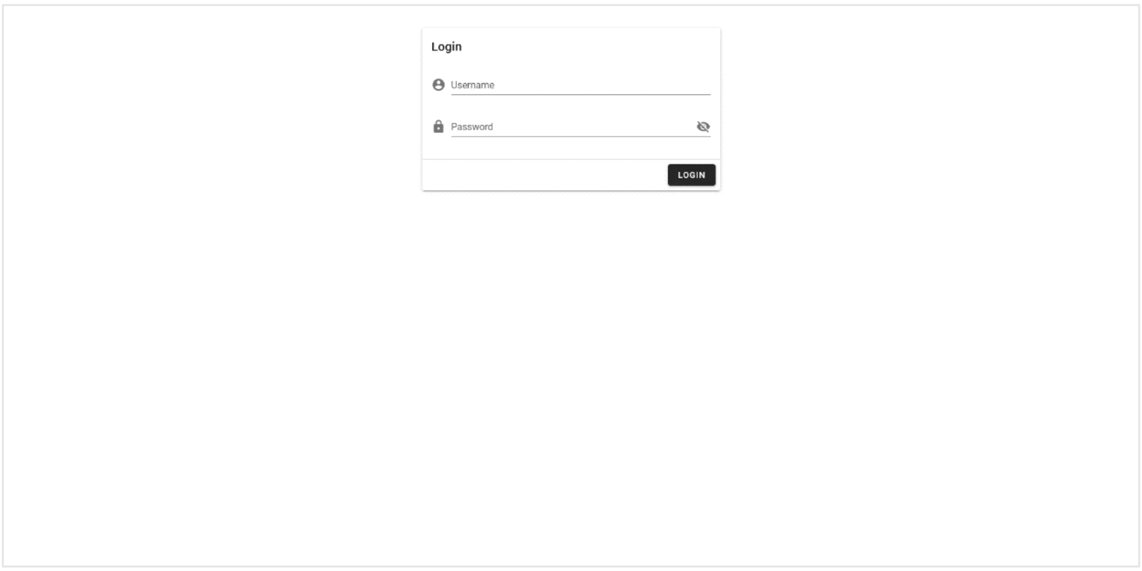
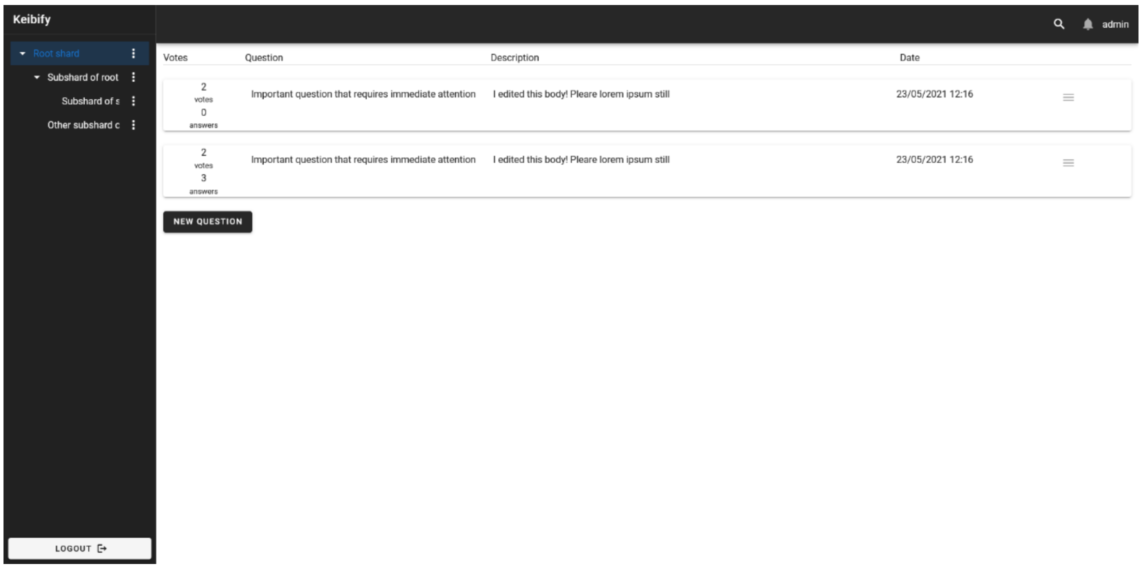


Figure 6. Login view.

4.4.2 Questions

The purpose of the questions view that is displayed in Figure 7, is to show the user all the questions that have been added to a particular shard. It is also conveyed to the user if the question has any answers added to them or if the question has been voted on, in order to show the user if there has been any activity regarding that question.



Votes	Question	Description	Date
2 votes 0 answers	Important question that requires immediate attention	I edited this body! Pleare lorem ipsum still	23/05/2021 12:16
2 votes 3 answers	Important question that requires immediate attention	I edited this body! Pleare lorem ipsum still	23/05/2021 12:16

Figure 7. Entirety of the questions view.

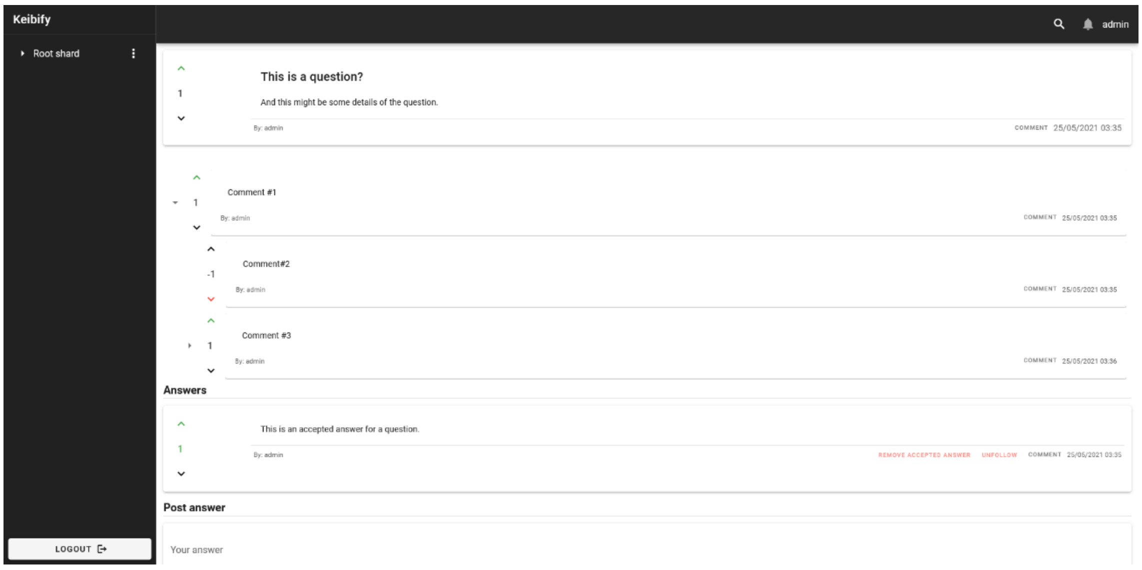
Each question object is rendered in a Vuetify card component which is separated into five columns containing information about that question with the last column containing a button that could be used to perform some extra actions, deleting the question for example. The first column contains the number of answers added and the score of the question based on votes from other users to indicate that there has been some activity regarding that question. In order to improve this view and the UX, the component containing the question could indicate to the user if an accepted answer exists.

The view also contains a button at the bottom of the page that can be used to create a new question and appears only if the user has the *CONTRIBUTE_CONTENT* permission.

The view for the search results is very similar to the questions view due to the card component that displays the data of the questions being re-used. The only differences are that the user cannot create a new question from this view and that the questions displayed are from the result of a request made for a single shard. The search results view could be improved by displaying the details of the search as well, meaning that the user always has an overview of what these results are in relation to. One way this could be implemented is to group the results together by shards and displaying them under a label of that specific shard, indicating to the user where that question belongs. This would resolve the issue of having multiple questions with the same keywords but having no way to identify them without closer examination.

4.4.3 Question

The question view, which is displayed on figure 8, contains the question itself and the relevant answers and comments. The comments are displayed in a tree view in order to comply with the requirement that comments can have their own sub-comments.



This is a question?

And this might be some details of the question.

By: admin COMMENT 25/05/2021 03:35

Comment #1

By: admin COMMENT 25/05/2021 03:35

Comment#2

By: admin COMMENT 25/05/2021 03:35

Comment #3

By: admin COMMENT 25/05/2021 03:36

Answers

This is an accepted answer for a question.

By: admin REMOVE ACCEPTED ANSWER UNFOLLOW COMMENT 25/05/2021 03:35

Post answer

Your answer

Figure 8. Entirety of the question view.

Currently the comment components are somewhat difficult to differentiate from both the question and the answer components. The things that separate them now are the comments being slightly offset due to being displayed as a tree view and that the other components have an outline which bring them forward a bit to capture the attention of the user. However, as there is no theme currently present in the application, all the components are displayed with a white background and use a very similar layout, which may blend them together in a way that makes it more difficult for the user to read.

This could be fixed by adding some styling to the comment card component that is inserted into the label slot of the Vuetify v-treeview component or to the tree view item itself.

4.5 Components

This section goes over a few of the components used in the creation of the project.

4.5.1 Sidebar

The sidebar component is where all the available shards are displayed in. It allows users to seamlessly switch between shards at any given time when they need to. This is an element which follows the forgiveness principle of UX design.

The difficulty of implementing the tree-view of the shards lied in getting the response from the API to the correct format which then the Vuetify v-treeview component could use. The JSON object from the response listed all the shards that the user has access to, with sub-shards being indicated by their ids in an array but the Vuetify component requires the objects to be represented instead of the ids. So, in order to avoid reinventing the wheel and after trying some other libraries, the list-to-tree [11] library was found that could parse the response. The library uses the *parent_id* parameter of a shard object to construct the key instead of the *subshard_ids* array.

The parse happens in a Vuex action described in Figure 9, when the sidebar is rendered which happens right after the user logs in.

```
async listToTree ({ dispatch, state, commit }) {
      const arrayToTree = require('array-to-tree')
      await dispatch('getShards')
      const tree = arrayToTree(state.shards)
      commit('SET_TREE', tree)
    }
```

Figure 9. Vuex action to parse the API response to the correct format for treeview component.

The action makes use of the Vuex dispatch method which calls the *getShards* action. The *await* keyword specifies that the request should be completed before moving on to the next step. That action will call a Vuex mutation that will save the response of the API into a state. That state is then given to the *arrayToTree* method as a parameter and a new mutation is called which will set the return value of the *tree* state.

The same action as displayed in Figure 9 is also called each time a new shard is added by the user. This way the getter set up in the sidebar component will recognise that a change has happened in the state and will automatically re-render the component with the new shard added. This makes it so that the user does not have to refresh the whole page to see any change that was made which complies with the predictability principle.

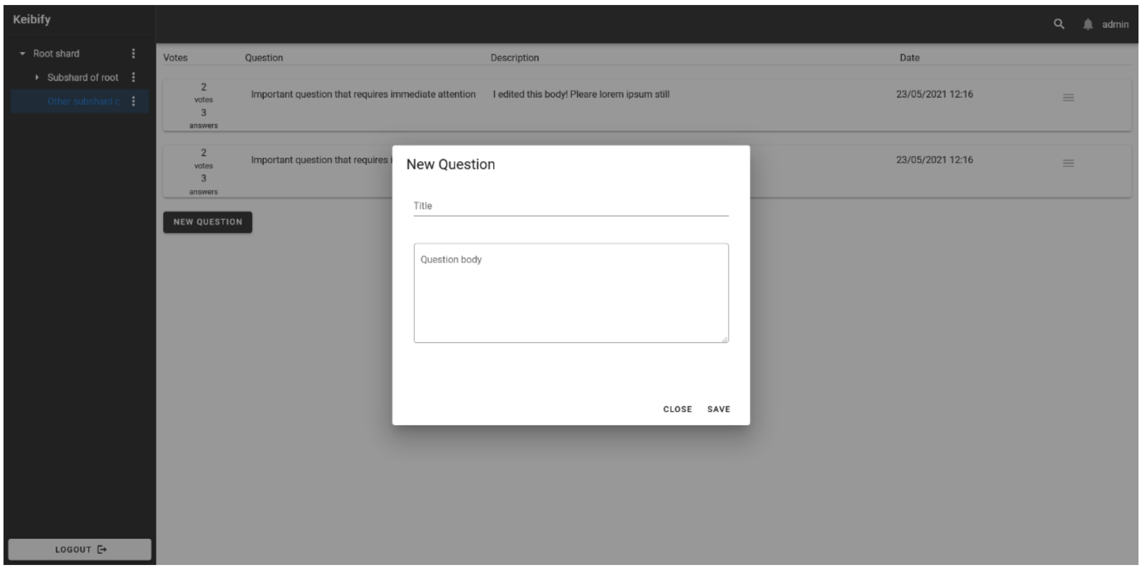
It is also possible to edit and delete shards depending on whether the user has the permissions to do so. The user needs to have a *CHANGE_SHARD* permission to edit and *SHARD_DELETE* permission to delete. The permissions are gotten from the user object stored in a Vuex state upon logging in. The buttons are visible even if the user does not have the permission, however the Vuetify *v-btn* component has the disabled property enabled on the condition if the user has the needed permissions.

There is currently a problem with displaying the name of the shard in the tree. If the name is too long to display on one row, then instead of continuing on a new line, the text is just cut off. Even if the name of the shard is not supposed to be very long since there is a description field on creation, it can still become a bigger problem when there are many sub-shards, because each sub-shard's label slot is a little bit smaller than the parent's.

Also, clicking on the label slot of a currently active shard from the question view does not direct the user back to the questions view. This is because clicking on the label will deactivate the node of the *v-treeview* component and will render an empty page and since a sensible solution was not found, clicking on an active label slot was disabled. However, this behaviour still occurs when clicking on any other slot on the shard node.

4.5.2 Modal dialogues

Adding a new shard or question will open a modal dialogue window with the user, as displayed in Figure 10. Both of them contain a form to get the user input necessary to create the new object.



Votes	Question	Description	Date
2 votes 3 answers	Important question that requires immediate attention	I edited this body! Pleare lorem ipsum still	23/05/2021 12:16
2 votes 3 answers	Important question that requires i		23/05/2021 12:16

Figure 10. Modal dialogue window for creating a new question.

Filling the text fields will bind the input to the Vue data property which can then be accessed in the submit method. When the text fields have been filled and the user clicks *save*, a submit method bound to the button, described in Figure 11, is executed and an Axios POST request is made to the API and the application is rerouted to the question page.

```
submit () {
  axios.post('/v1/content/qa', this.params).then(response => {
    this.$router.push({ name: 'question', params: { id: response.data.id } })
  })
}
```

Figure 11. The method that runs when clicking the save button.

The params in Figure 11 contains the question title, body and the shard id where the question will be created. The id of the shard is found in the parameters of the router object. The request is not handled in the Vuex store since there is no need to update the state of questions immediately as the user is redirected to the question view based on the id of the new question provided by the response.

Searching for content also opens a modal dialogue window, where the user can enter the keywords that they want to find and select the shards they would expect those keywords to be found. However, if the user is not sure which shard the question might be in, they can also opt to search from all the sub-shards of the selected shards. The back-end API is also prepared for the event of additional content types being added in the form of expecting a *content_type* parameter in the request, however currently while making the request it is hard-coded to only include Questions.

Since the search function opens a dialogue with the user, it can be considered as being an advanced search. This option should remain for the users that know what they are looking for, but a simpler option should exist for users that do not. This could be implemented by just using a text field that opens when clicking on the search icon and allowing the user to write in their query without the extra options. This, however, requires making some changes to the request that is currently being ran by searching. Specifically, the request should use only the id of the root shard that the user has access to and the *includeFromSubshards* parameter should be set to *true* by default.

The choice to use modals instead of directing the user to a new page, was made so that creating a new item would not take the user out of their current flow. Since adding a new item does not require a lot of user input, the dialog windows are small and do not take up much space. Also, the user can easily exit them by clicking on the close button or anywhere other than the window. However, if in the future more user input is needed, such as adding members or groups to a new shard, directing the user to a separate creation page might be a better solution.

4.5.3 Comments

Like the shards in the sidebar component, the comments for both the question and answers also appear in a tree view under the relevant item, so that the sub-comments could be displayed in a way that is easy to navigate. The comments are received from the API by dispatching a Vuex action, described in Figure 12, in the created lifecycle hook of the question view.

```
async getQuestion ({ commit }, id) {
  await axios.get(`/v1/content/qa/${id}`).then(response => {
    commit('GET_QUESTION', response.data)
  })
}
```

Figure 12. A Vuex action that executes an axios get request to get the question data.

Created hook is one of the first hooks that runs in the creation of a component. It allows to perform actions before the component is rendered, which helps to saturate it with data. In the other case the component might want to access data of an object that does not exist yet.

In the question view the *getQuestion* action in the Vuex store will perform a request to get the data of the relevant question which includes the comments. The comments from the API are already in the correct format for the tree-view so parsing the response here is not necessary. The only thing needed to modify for the Vuetify treeview component, is declaring which parameter contains the sub-commnet objects of a comment.

To add a new comment, a text field will appear in the subtitle field of the Vuetify v-card component as seen in Figure 13. Whether the form is visible or not is dictated by a custom flag which is set to *true* when the user clicks on the *comment* button. In turn the flag is set to *false* when the *comment* button is clicked again or when the user clicks the *post comment* button.

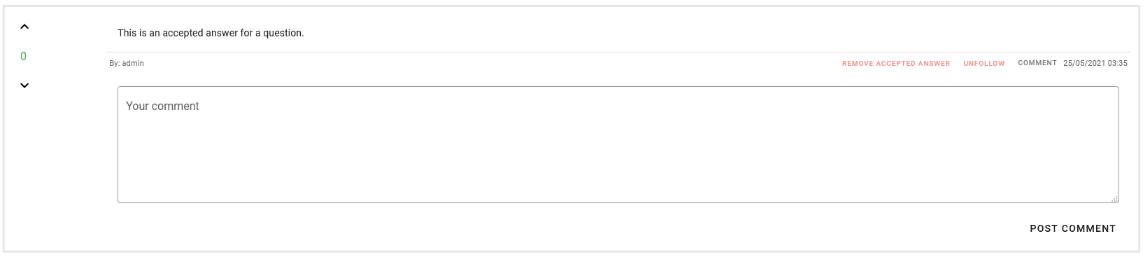


Figure 13. The card component of a posted answer with the new comment form active.

4.5.4 Question and Answer

The Question and Answer components are grouped together since they are very similar with some minor differences. They are both made using the Vuetify v-card component separated into two columns. The first column is filled by the Votes component while the other uses the v-card's own components.

The differences of these components are that the Question component has the initial question in the *v-card-title* component with the details of the question being in the *v-card-text* component. The Answer component only has the answer in the *v-card-text* component.

Both components have buttons added to them, with the Answer component having two more, based on some conditions. In the view of the user that posted the initial question, the *accept answer* button is added. This is done by comparing the id of the creator of the question against the id of the user stored in the state. When the button is pressed, the *accepted* data property of the component is set to *true*, and passed to the Voting component, which will then turn the vote score label green, indicating that the answer has been marked as accepted. The *accepted* property also dictates what button is displayed to the user. If the value is false, then the button displayed is to accept the answer. In the other case, the button text and the colour red will indicate that clicking on it, will remove the accepted status from the answer.

The Answer component also has the button to add it to the user's followed list. This button will appear only if the user has a *READ* permission for the shard the question is in.

The style of the buttons is quite minimalistic, with only the label appearing as the button. The choice was made to keep the look of the component clean without cluttering it with unnecessarily bulky buttons.

4.5.5 Votes

The voting system was made into a separate component so that it could be reused throughout the project where necessary. It includes the upvote and downvote buttons and the label that displays the current vote score of an item.

The current vote object of an item as seen in Figure 14, is passed down to the component by its parent, be it a question, answer or comment. That object is then saved into the data property of the component so it would be reactive when any modifications are made to it. The POST request that is made against the API on button press will respond with a new vote object which will be the replacement for the current one.

```
{
    "vote_score": 1,
    "vote_count": 3,
    "item_id": 35,
    "user_vote_type": -1
}
```

Figure 14. Object returned by making a POST request upon voting.

The *user_vote_type* indicates what sort of vote the user made, either upvote or downvote. With that being in the data property, the component can easily access these parameters and change the behaviour of the buttons. For example, the *user_vote_type* value being -1 means that the user has downvoted the item. The downvote button will turn red, and when it is pressed again, instead of downvoting a second time, the request will be made with the parameter *vote_type* being 0 instead of -1 as described in Figure 15.

```
downVote () {
  if (this.votes.user_vote_type === -1) {
    axios.post(`/v1/content/vote/item/${this.item.id}`, {
      vote_type: 0
    }).then(response => (this.votes = response.data))
  } else {
    axios.post(`/v1/content/vote/item/${this.item.id}`, {
      vote_type: -1
    }).then(response => (this.votes = response.data))
  }
}
```
Figure 15. Method that runs when clicking the downvote button in the Voting component.

4.6 Authentication

At first the authentication happened before each Axios request made. That however meant that each time a new request needed to be implemented an additional request had to be made in order to get the access token needed for the authorisation header of the request, so a better solution was needed to keep the code DRY. This is where Vuex first was applied to the project.

Creating the authentication process begins with setting up a Vuex action that performs a request against the API with the user credentials from the input, to get the authorisation token. Then another action is dispatched with the token as a parameter from the previous request response to call a Vuex *SET_TOKEN* mutation to save the user data and token in a state if the token is present.

After that, a Vuex store subscriber, described in Figure 16, is created and listens for any *SET_TOKEN* mutation made. When the mutation is called with the authorisation token as its parameter, the token is added to Axios default headers and the browsers localStoage. This way it is not necessary to make an additional request for the headers before each Axios call. After that, a request to the API is made to get the user information. If the token is invalid or is not present, the request will fail, which will nullify the token in the localStorage.

```
store.subscribe((mutation) => {
  switch (mutation.type) {
    case 'auth/SET_TOKEN':
      if (mutation.payload) {
        axios.defaults.headers.common.authorization = `Bearer
${mutation.payload}`
        localStorage.setItem('token', mutation.payload)
      } else {
        axios.defaults.headers.common.authorization = null
        localStorage.removeItem('token')
      }
  }
})
```

Figure 16. The subscriber function that listens for SET_TOKEN mutation calls.

In order to not let the user type in a valid URL themselves to gain access to the application without authorisation, a simple middleware function displayed in Figure 17 was created which checks if the user is currently authenticated and if not, redirects them back to the login page. The middleware function is used in the Vue Router *beforeEnter* parameter as seen in Figure 3.

```
export default function (to, from, next) {
  if (!store.getters['auth/authenticated']) {
    return next({
      name: 'login'
    })
  }
  next()
}
```

Figure 17. Middleware function that redirects the user if not authenticated.

4.7 Reactive rendering

Reactive rendering in this context means that a view is updated when changes are made to a state. This is applied to shards in the sidebar, questions view and question view. Whenever an item is added or removed, the relevant request is handled in a Vuex store. This allows to update a certain state with a mutation, which then can be referenced from within the view.

The reason for applying this method is so that the change would be rendered in the component immediately and will not leave the user wondering if their action was a success or not. In the other case the user would need to refresh the page to see the change and that would not comply with the predictability principle.

4.8 Theme

There currently is not a specific custom colour scheme applied to the project. That is because during development, the focus was more on making a functional MVP. However, as mentioned in section 3.2, Vuetify components all natively use the material design. It comes with a built-in baseline design that is usable as-is and can easily be modified to meet the requirements of a product [12]. Using a set theme helps keep the look and feel of an application consistent and if the design is attractive, it will attract new users and will keep the current ones interested.

The theme of the current site can be attributed to using the *light* and *dark* properties of the Vuetify components. The default theme of a component is *light* however when the *dark* property is set the colours will be inverted. The background colour becomes dark while the text displayed becomes white. The dark property in the project is set to two components: the sidebar and the top navigation bar. The reason for applying the property for those two components was to give the page some contrast and separate navigation elements from the content elements.

To create a specific theme for the application, some further research into UX and UI design is needed. Colours play an important part in what kind of feel a website has, so it needs to be very clear what the application is trying to convey and who the target audience is.

4.9 Deployment

Access to the server where the application lies was given by the back-end developer of the project. First, node and NPM needed to be installed in order to set up an apache2 webserver however the webserver was later changed to NGINX. NPM is also needed to install dependencies when deploying the project.

The project makes use of the GitLab CI/CD feature to automatically deploy the updated application when pushing changes to the master branch. The deployment consists of updating or installing new dependencies, building the project and then copying the public build directory to the webserver. The commands necessary to run those actions can be found in the *.gitlab-ci.yml* file in the root directory of the project and are shown in Figure 18.

```yaml
stages:
  - deploy
deploy:
  stage: deploy
  tags:
    - group-hetzner-shell
  script:
    - npm install
    - npm run build
    - cp -R dist/* /var/www/html
  only:
    - master
```

Figure 18. Content of the .gitlab-ci.yml file.

5 Future plans

The plans for the future of the project are to keep developing it further. A good part of the endpoints that the back-end offers are not yet implemented.

Since there is currently a possibility to follow questions, a notifications system needs to be developed to notify users when a change has happened to the followed item.

The interest to develop the project further also stems from the author wanting to learn the Vue framework further and increase overall knowledge in front-end development.

6 Summary

The aim of this thesis was to create a functional user interface for a knowledge sharing platform for organisational use. The result is an MVP that provides the users the possibility to ask or find questions which are categorised into shards based on the organisational requirements and receive answers which would be helpful for solving a problem. The author of the thesis plans to keep developing the application further in order to meet the possibilities the back-end of the system provides and to increase their knowledge of front-end development.

7 References

[1] S. Garg, “User Experience or UX Design,” GeeksforGeeks, 29 January 2020. [Online]. Available: https://www.productplan.com/glossary/user-experience/. [Accessed 12 May 2021].

[2] N. Babich, “What are the Similarities & Differences Between UI Design & UX Design?,” Xd Ideas, 4 October 2019. [Online]. Available: https://xd.adobe.com/ideas/process/ui-design/ui-vs-ux-design-understanding-similarities-and-differences/. [Accessed 13 May 2021].

[3] N. Babich, “The 4 Golden Rules of UI Design,” Xd Ideas, 7 October 2019. [Online]. Available: https://xd.adobe.com/ideas/process/ui-design/4-golden-rules-ui-design/. [Accessed 13 May 2021].

[4] T. Krotoff, “Frontend Frameworks Popularity,” 5 May 2019. [Online]. Available: https://gist.github.com/tkrotoff/b1caa4c3a185629299ec234d2314e190. [Accessed 17 May 2021].

[5] ThemeSelection, “The Most Popular JavaScript Frameworks in 2021,” Medium, 24 March 2021. [Online]. Available: https://javascript.plainenglish.io/the-most-popular-javascript-frameworks-in-2021-a2fe62174df6. [Accessed 13 May 2021].

[6] Vuejs, “What is Vuex?,” Vuejs, 10 May 2021. [Online]. Available: https://next.vuex.vuejs.org/. [Accessed 14 May 2021].

[7] Vuejs, “Vue CLI guide,” Vuejs, 7 October 2019. [Online]. Available: https://cli.vuejs.org/guide/. [Accessed 14 May 2021].

[8] Vuejs, “Vue Router,” Vuejs, 2021. [Online]. Available: https://router.vuejs.org/. [Accessed 14 May 2021].

[9] SaaS Design, “Figma Material Design UI Kit & Component Library,” Figma freebies, 30 April 2019. [Online]. Available: https://www.figmafreebies.com/download/figma-material-design-ui-kit-component-library/. [Accessed 15 May 2021].

[10] Vuejs, “Navigation Guards,” Vuejs, 2021. [Online]. Available: https://router.vuejs.org/guide/advanced/navigation-guards.html. [Accessed 17 May 2021].

[11] alferov, “array-to-tree,” 7 December 2019. [Online]. Available: https://github.com/alferov/array-to-tree. [Accessed 16 May 2021].

[12] “Material theming,” Google, 2021. [Online]. Available: https://material.io/design/material-theming/overview.html#material-theming. [Accessed 16 May 2021].

Appendix 1 – Non-exclusive licence for reproduction and publication of a graduation thesis[1]

1 The non-exclusive licence is not valid during the validity of access restriction indicated in the student's application for restriction on access to the graduation thesis that has been signed by the school's dean, except in case of the university's right to reproduce the thesis for preservation purposes only. If a graduation thesis is based on the joint creative activity of two or more persons and the co-author(s) has/have not granted, by the set deadline, the student defending his/her graduation thesis consent to reproduce and publish the graduation thesis in compliance with clauses 1.1 and 1.2 of the non-exclusive licence, the non-exclusive license shall not be valid for the period.

Appendix 2 – List of the used back-end API endpoints

HTTP verb	**API endpoint, root URL: api.dev.keibify.com**	**Description**
GET	/v1/user/self	Current user details with permissions
POST	/v1/content/vote/item/:id	Add a vote to an item
POST	/v1/content/qa	Create a question
GET	/v1/shard	Get list of all shards
POST	/v1/search	Search for a question from specific shards, can include sub-shards
GET	/v1/content/qa/:id	Get question details, includes answers and comments
POST	/v1/shard	Create a new shard
DELETE	/v1/favourite/shard/:id	Remove shard from favourites
POST	/v1/favourite/shard	Add shard to favourites
DELETE	/v1/shard/:id	Delete shard
PUT	/v1/shard/:id	Edit shard
POST	/v1/content/qa/:id/accept	Accept an answer

DELETE	/v1/content/qa/:id/accept	Remove accepted answer
DELETE	/v1/follow/item/:id	Stop following an item
POST	/v1/follow/item/:id	Follow an item
POST	/v1/content/qa/:id/answer	Post a new answer to question
POST	/v1/content/qa/:id/comment	Post a comment to an item