

В данном модуле создаем приложение, которое будет взаимодействовать с серверным API. Файлы верстки и готовое API предоставляются. Верстка по ссылке:

<https://github.com/afanasyevadina/touschoolcourse/blob/master/Surveys%20Frontend%20Template.zip>

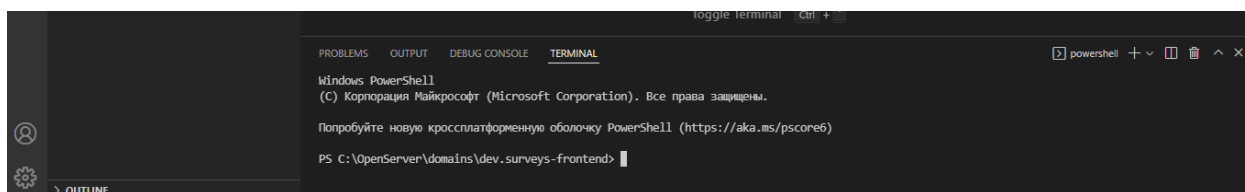
API: <http://afanasso.beget.tech/>

Спецификация:

<https://github.com/afanasyevadina/touschoolcourse/blob/master/%D0%A1%D0%BF%D0%B5%D1%86%D0%B8%D1%84%D0%B8%D0%BA%D0%B0%D1%86%D0%B8%D1%8F.docx>

Исходники проекта: <https://github.com/afanasyevadina/tousurveysfrontend>

Сначала необходимо установить VueJS. Создадим папку dev.surveys-frontend, откроем в VSCode. Терминал открывается сочетанием Ctrl+`:



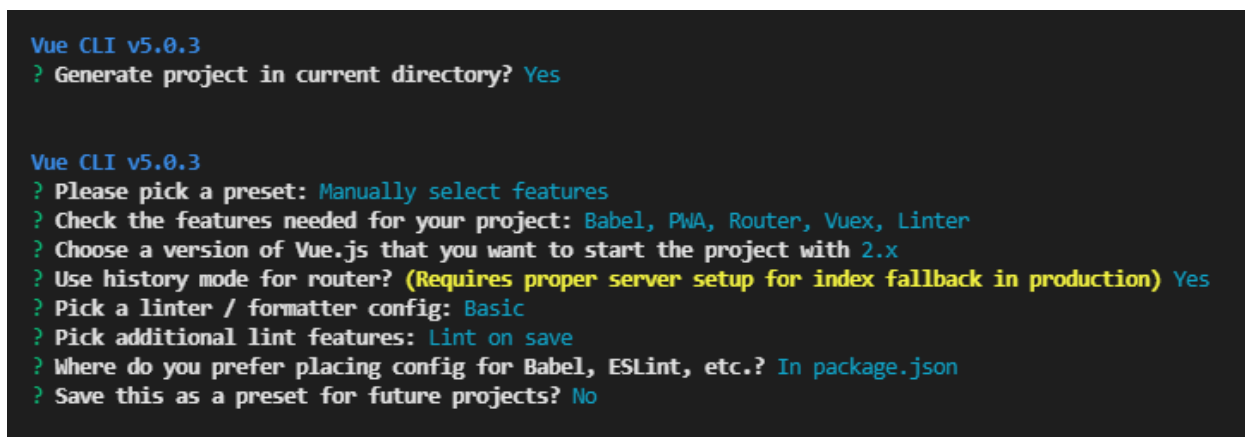
У вас уже должен быть установлен NodeJS и npm. Установим vue-cli:

```
npm i -g @vue/cli
```

Создадим проект:

```
vue create .
```

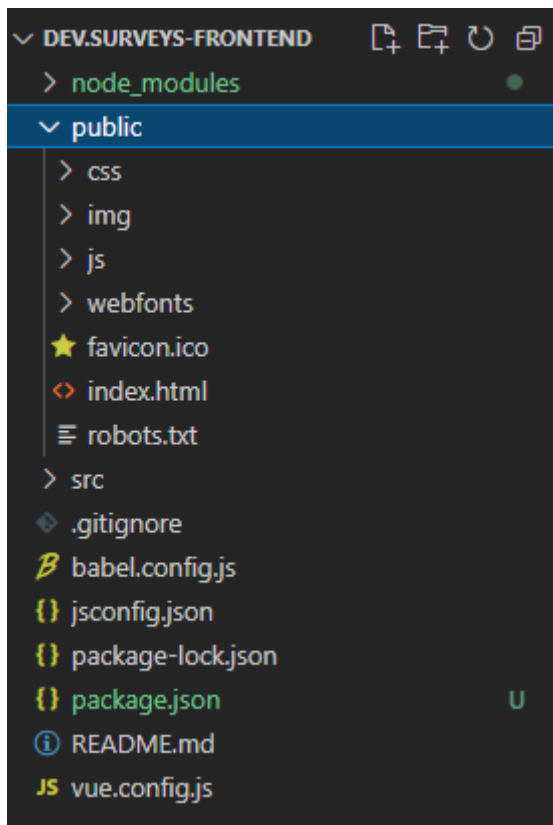
Настройки такие:



Надо создать:

- перенести ассеты
- файлы представлений и скрипты
- роутинг
- настроить хранилище.

Скопируем папки css, webfonts и js в папку public:



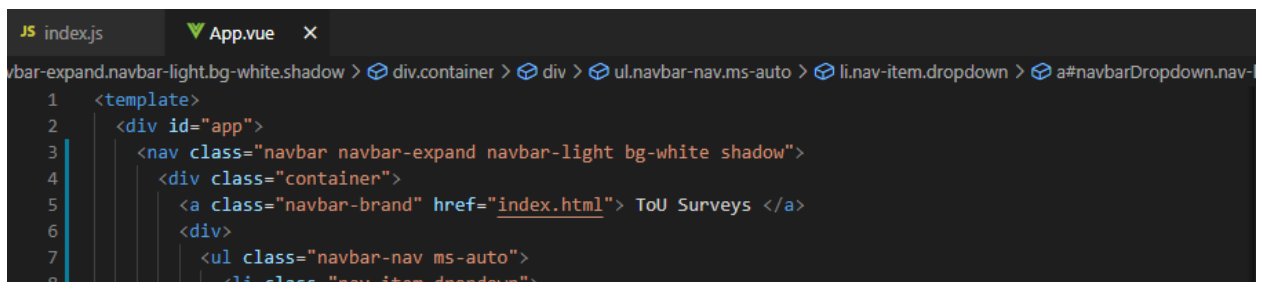
Измените содержимое секции head в файле *public/index.html*:

```
<head>
  <meta charset="utf-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width,initial-scale=1.0">
  <link rel="icon" href="<%= BASE_URL %>favicon.ico">
  <title>ToU Surveys</title>
  <link rel="stylesheet" href="<%= BASE_URL %>css/bootstrap.min.css">
  <script src="<%= BASE_URL %>js/bootstrap.bundle.min.js" defer></script>
</head>
```

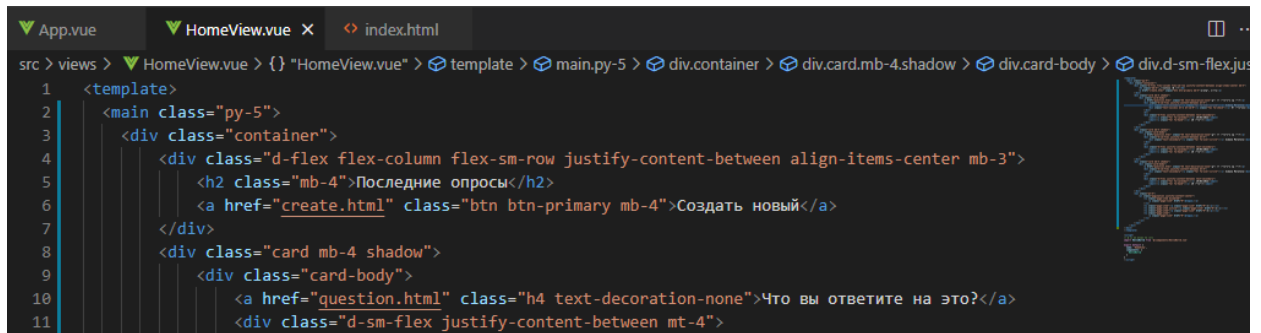
Теперь начнем переносить верстку в представления vue.

Каждый шаблон содержит блок template, в котором будет html, script, где будет сам скрипт, и style, где можно прописать стили (опционально).

Общая у нас шапка. Откроем *src/App.vue*. заменим тег nav на такой же тег из файла *index.html* в предоставленной папке. Секцию style удалим, она не нужна.



Откроем `src/views/HomeView.vue`. В `template` вставим тег `main` из файла `index.html` шаблона:



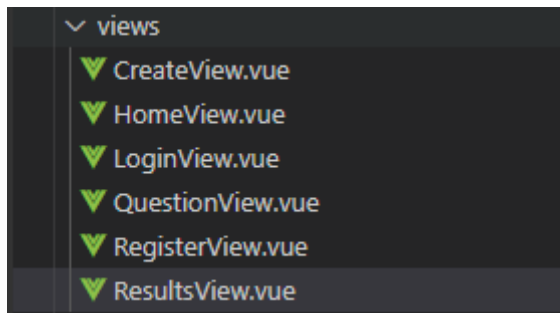
```
1 <template>
2   <main class="py-5">
3     <div class="container">
4       <div class="d-flex flex-column flex-sm-row justify-content-between align-items-center mb-3">
5         <h2 class="mb-4">Последние опросы</h2>
6         <a href="create.html" class="btn btn-primary mb-4">Создать новый</a>
7       </div>
8       <div class="card mb-4 shadow">
9         <div class="card-body">
10          <a href="question.html" class="h4 text-decoration-none">Что вы ответите на это?</a>
11          <div class="d-sm-flex justify-content-between mt-4">
```

Из скрипта удалим `HelloWorld`, оставим пока пустой экспорт:

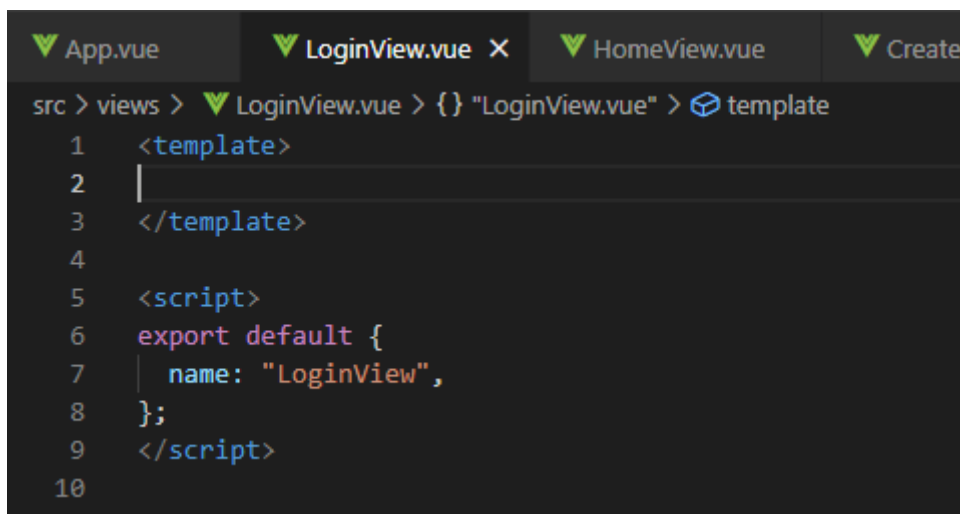
```
<script>
export default {
  name: 'HomeView'
}
</script>
```

`AboutView.vue` удалим.

Создадим файлы:



Откроем `LoginView.vue`. Создадим секции `template` и `script`:



```
1 <template>
2   |
3 </template>
4
5 <script>
6   export default {
7     name: "LoginView",
8   };
9 </script>
10
```

В `template` помещаем тег `main` из соответствующего файла верстки.

```
App.vue LoginView.vue X login.html QuestionView.vue RegisterView.vue index.html M
src > views > LoginView.vue > {} "LoginView.vue" > template > main.py-5 > div.container.pt-5 > div.row.justify-content-center > div.c
1 <template>
2   <main class="py-5">
3     <div class="container pt-5">
4       <div class="row justify-content-center">
5         <div class="col-lg-6 col-sm-8">
6           <div class="card shadow">
7             <div class="card-body p-4">
8               <h2 class="mb-5 text-center">Авторизация</h2>
```

Аналогично проделаем со всеми оставшимися файлами в папке views, не забывая менять name в export.

Теперь, когда есть все шаблоны, можно настроить роутинг. Откроем src/router/index.js и изменим следующим образом:

```
import Vue from 'vue'
import VueRouter from 'vue-router'
import HomeView from '../views/HomeView.vue'
import LoginView from '../views/LoginView.vue'
import RegisterView from '../views/RegisterView.vue'
import QuestionView from '../views/QuestionView.vue'
import ResultsView from '../views/ResultsView.vue'
import CreateView from '../views/CreateView.vue'

Vue.use(VueRouter)

const routes = [
  {
    path: '/',
    name: 'home',
    component: HomeView
  },
  {
    path: '/login',
    name: 'login',
    component: LoginView
  },
  {
    path: '/register',
    name: 'register',
    component: RegisterView
  },
  {
    path: '/question/:id',
    name: 'question',
    component: QuestionView,
    props: true
  },
  {
    path: '/results/:id',
    name: 'results',
    component: ResultsView,
```

```
    props: true
  },
  {
    path: '/create',
    name: 'create',
    component: CreateView
  },
]

const router = new VueRouter({
  mode: 'history',
  base: process.env.BASE_URL,
  routes
})

export default router
```

В верхней части мы подключаем файлы представлений и VueRouter. Далее указываем приложению, что надо использовать роутер. В объекте routes перечисляются соответствия путей и представлений. С помощью :id и props: true указываем, что будет передаваться параметр id (например, айди вопроса). Позже в этом же файле мы настроим проверку на авторизацию. А сейчас проверим, как настроилось все. Выполним

```
npm run serve
```

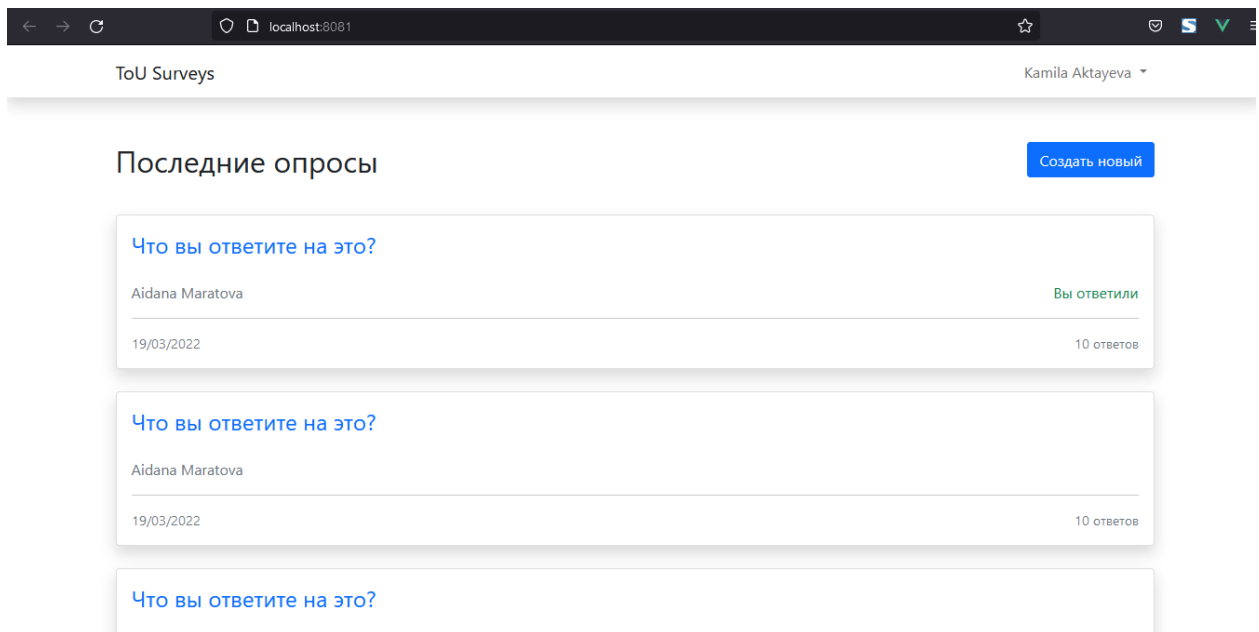
в терминале. Перейдем по ссылке из терминала:

```
DONE Compiled successfully in 2360ms

App running at:
- Local:   http://localhost:8081/
- Network: http://192.168.0.14:8081/

Note that the development build is not optimized.
To create a production build, run npm run build.
```

По адресу в корне должны видеть главную:



Пробуем поочередно вводить адреса из роутов:

localhost:8081

localhost:8081/login

localhost:8081/register

localhost:8081/create

localhost:8081/question/1

localhost:8081/results/1

если все правильно, должны видеть соответствующие страницы.

Теперь поработаем с авторизацией и настройкой ссылок.

Настроим хранилище. Вынесем в него объект текущего пользователя. Изменим файл `src/store/index.js`:

```
import Vue from 'vue'
import Vuex from 'vuex'

Vue.use(Vuex)

export default new Vuex.Store({
  state: {
    user: null
  },
  getters: {
    user(state) {
      return state.user
    }
  },
  mutations: {
    login(state, user) {
```

```

    state.user = user
    localStorage.setItem('user', JSON.stringify(user))
  },
  logout(state) {
    state.user = null
    localStorage.removeItem('user')
  },
  restoreUser(state) {
    if(localStorage.getItem('user')) {
      state.user = JSON.parse(localStorage.getItem('user'))
    }
  }
},
actions: {
},
modules: {
}
})

```

У нас есть объект user, получить его сможем через геттер, манипулируем значением через мутаторы. Так, определим методы для запоминания нового пользователя и автосохранение через localStorage.

Теперь поработаем с файлом App.vue. Во-первых, отредактируем ссылку на главную таким образом:

```
<router-link class="navbar-brand" to="/"> ToU Surveys </router-link>
```

Ссылки делаются не через тег a, а через router-link.

Добавим script со следующим кодом:

```

<script>
import {mapGetters} from 'vuex'
export default {
  computed: mapGetters(['user']),
  methods: {
    logout: function() {
      this.$store.commit('logout')
      this.$router.push('/login')
    }
  },
  created() {
    this.$store.commit('restoreUser')
  }
}
</script>

```

Здесь мы подключаем геттер, он будет работать как вычисляемое свойство. Определим метод выхода, так как кнопка выхода у нас как раз в шапке и восстановление сессии при открытии приложения. Теперь отредактируем html шапки. Добавим проверку на авторизованного пользователя, динамически подставим имя и повесим обработчик на кнопку выхода/

1. router-link
2. v-if="user"
3. {{ user.name }}
4. @click.prevent="logout"

```
<template>
  <div id="app">
    <nav class="navbar navbar-expand navbar-light bg-white shadow">
      <div class="container">
        <router-link class="navbar-brand" to="/"> ToU Surveys </router-link>
        <div v-if="user">
          <ul class="navbar-nav ms-auto">
            <li class="nav-item dropdown">
              <a id="navbarDropdown" class="nav-link dropdown-toggle" href="#"
role="button" data-bs-toggle="dropdown" aria-haspopup="true" aria-
expanded="false" >
                {{ user.name }}
              </a>
              <div class="dropdown-menu dropdown-menu-end" aria-
labelledby="navbarDropdown" >
                <a class="dropdown-item" href="#" @click.prevent="logout"> Выход
              </a>
            </div>
          </li>
        </ul>
      </div>
    </nav>
    <router-view />
  </div>
</template>
```

Теперь ограничим доступ по роутам для неавторизованного пользователям. Перейдем в src/routes/index.js. Перед экспортом напишем функцию:

```
router.beforeEach((to, from, next) => {
  if (!localStorage.getItem('user') && ![ 'login', 'register' ].includes(to.name))
    next({ name: 'login' })
  else next()
})
```

Для всех путей, кроме авторизации, будет перенаправлять на логин, если пользователя нет. Попробуем открыть главную страницу. Нас должно выкинуть на логин.

Теперь надо реализовать логин. Перейдем в LoginView.vue и свяжем поля с объектом и повесим на форму обработчик:

```
<template>
  <main class="py-5">
    <div class="container pt-5">
      <div class="row justify-content-center">
        <div class="col-lg-6 col-sm-8">
          <div class="card shadow">
            <form action="#" class="card-body p-4" @submit.prevent="login">
              <h2 class="mb-5 text-center">Авторизация</h2>
              <div class="mb-4">
                <input type="email" class="form-control" :class="{ 'is-invalid':
errors.email || errors.error}" placeholder="Ваш email" v-model="form.email" />
                <span class="invalid-feedback">{{ errors.email || errors.error
}}</span>
              </div>
              <div class="mb-4">
                <input type="password" class="form-control" :class="{ 'is-
invalid': errors.password}" placeholder="Ваш пароль" v-model="form.password" />
                <span class="invalid-feedback">{{ errors.password }}</span>
              </div>
              <button class="btn w-100 btn-primary mb-3">Войти</button>
              <div class="text-center">
                <small>
                  Нет аккаунта? <router-link to="/register">Регистрация</router-
link>
                </small>
              </div>
            </form>
          </div>
        </div>
      </div>
    </div>
  </main>
</template>
```

1. @submit.prevent="login"
2. v-model
3. :class="{ 'is-invalid': errors.field }
4. {{ errors.field }}
5. router-link

Напишем скрипт, который будет посылать запросы. Дальше парсится ответ, отображаются ошибки, если есть. Если нет ошибок, логиним и отправляем на главную.

```
<script>
export default {
  name: "LoginView",
  data() {
```

```


return {
  form: {
    email: '',
    password: ''
  },
  errors: {}
}
},
methods: {
  login: function() {
    this.errors = {}
    fetch('http://afanasso.beget.tech/api/login', {
      method: 'POST',
      body: JSON.stringify(this.form),
      headers: {
        'Content-Type': 'application/json'
      }
    })
    .then(response => response.json())
    .then(json => {
      if(json.errors) {
        this.errors = json.errors
      } else {
        this.$store.commit('login', json.data)
        this.$router.push('/')
      }
    })
  }
}
};
</script>

```

В data заводим объект form для хранения данных формы. Errors – объект для сообщений об ошибках. Отправляем запрос и получаем ответ в JSON. Проверим:

Авторизация

example@example.com

Ваш пароль 

The password field is required.

Войти

Нет аккаунта? [Регистрация](#)

При верных данных должно отправлять на главную. Если ошибки – полям добавляется класс is-invalid и отображается текст ошибки.

Проделаем те же действия со страницей регистрации. Template:

```

<template>
  <main class="py-5">
    <div class="container pt-5">
      <div class="row justify-content-center">
        <div class="col-lg-6 col-sm-8">
          <div class="card shadow">
            <form action="#" class="card-body p-4"
@submit.prevent="register">
              <h2 class="mb-5 text-center">Регистрация</h2>
              <div class="mb-4">
                <input type="name" class="form-control" :class="{ 'is-
invalid': errors.name}" placeholder="Ваше имя" v-model="form.name">
                <span class="invalid-feedback">{{ errors.name
}}</span>
              </div>
              <div class="mb-4">
                <input type="email" class="form-control"
:class="{ 'is-invalid': errors.email}" placeholder="Ваш email" v-
model="form.email" />
                <span class="invalid-feedback">{{ errors.email
}}</span >
              </div>
              <div class="mb-4">
                <input type="password" class="form-control"
:class="{ 'is-invalid': errors.password}" placeholder="Ваш пароль" v-
model="form.password" />
                <span class="invalid-feedback">{{ errors.password
}}</span>
              </div>
              <button class="btn w-100 btn-primary mb-3">Создать
аккаунт</button>
              <div class="text-center">
                <small>
                  Уже зарегистрированы? <router-link
to="/login">Войти</router-link>
                </small>
              </div>
            </form>
          </div>
        </div>
      </div>
    </div>
  </main>
</template>

<script>
export default {
  name: 'RegisterView',
  data() {
    return {
      form: {

```

```

        name: '',
        email: '',
        password: ''
      },
      errors: {}
    }
  },
  methods: {
    register: function() {
      this.errors = {}
      fetch('http://afanasso.beget.tech/api/register', {
        method: 'POST',
        body: JSON.stringify(this.form),
        headers: {
          'Content-Type': 'application/json'
        }
      })
      .then(response => response.json())
      .then(json => {
        if(json.errors) {
          this.errors = json.errors
        } else {
          this.$store.commit('login', json.data)
          this.$router.push('/')
        }
      })
    }
  }
}
</script>

```

Все аналогично логину.

Теперь поработаем над главной. Добавим router-link, получим questions с сервера и выведем в шаблоне. Отредактируем скрипт:

```
<script>
import { mapGetters } from "vuex";

export default {
  name: "HomeView",
  data() {
    return {
      questions: [],
      links: [],
    };
  },
  computed: mapGetters(["user"]),
  methods: {
    loadQuestions: function (url = "http://afanasso.beget.tech/api/questions") {
      fetch(url, {
        headers: {
          Authorization: `Bearer ${this.user.api_token}`,
        },
      })
        .then((response) => response.json())
        .then((json) => {
          this.questions = json.data;
          this.links = json.meta.links;
        });
    },
  },
  created() {
    this.loadQuestions();
  },
};
</script>
```

Здесь уже GET запрос. При загрузке компонента происходит обращение к серверу. Авторизация через api_token текущего пользователя.

Данные приходят примерно в таком виде:

```
{
  "data": [
    {
      "id": 5,
      "text": "Что вы ответите простому пользователю а не админу?",
      "answers_count": 0,
      "has_answers": false,
      "user": {
        "id": 2,
        "name": "Simple user"
      },
      "created_at": "2022-03-13T05:42:33Z"
    }
  ]
}
```

```

    },
    {
        "id": 4,
        "text": "Что вы ответите?",
        "answers_count": 0,
        "has_answers": false,
        "user": {
            "id": 1,
            "name": "Admin"
        },
        "created_at": "2022-03-13T05:40:31Z"
    },
    {
        "id": 3,
        "text": "Что вы ответите?",
        "answers_count": 0,
        "has_answers": false,
        "user": {
            "id": 1,
            "name": "Admin"
        },
        "created_at": "2022-03-13T05:39:53Z"
    },
    {
        "id": 2,
        "text": "Вы что-то поняли?",
        "answers_count": 3,
        "has_answers": true,
        "user": {
            "id": 1,
            "name": "Admin"
        },
        "created_at": "2022-03-06T12:35:41Z"
    },
    {
        "id": 1,
        "text": "Когда мы начнем саморазвиваться?",
        "answers_count": 3,
        "has_answers": true,
        "user": {
            "id": 1,
            "name": "Admin"
        },
        "created_at": "2022-02-26T18:53:15Z"
    }
],
"links": {
    "first": "http://afanasso.beget.tech/api/questions?page=1",
    "last": "http://afanasso.beget.tech/api/questions?page=1",
    "prev": null,
    "next": null
},
"meta": {

```

```

    "current_page": 1,
    "from": 1,
    "last_page": 1,
    "links": [
      {
        "url": null,
        "label": "&laquo; Previous",
        "active": false
      },
      {
        "url": "http://afanasso.beget.tech/api/questions?page=1",
        "label": "1",
        "active": true
      },
      {
        "url": null,
        "label": "Next &raquo;",
        "active": false
      }
    ],
    "path": "http://afanasso.beget.tech/api/questions",
    "per_page": 10,
    "to": 5,
    "total": 5
  }
}

```

Здесь в объекте data сами данные, а в meta информация о пагинации. Отобразим в шаблоне.

```

<template>
  <main class="py-5">
    <div class="container">
      <div class="d-flex flex-column flex-sm-row justify-content-between align-items-center mb-3">
        <h2 class="mb-4">Последние опросы</h2>
        <router-link to="/create" class="btn btn-primary mb-4">Создать
        НОВЫЙ</router-link>
      </div>
      <div class="card mb-4 shadow" v-for="question in questions"
      :key="question.id" >
        <div class="card-body">
          <router-link :to="`question/${question.id}`" class="h4 text-decoration-
          none">
            {{ question.text }}
          </router-link>
          <div class="d-sm-flex justify-content-between mt-4">
            <div class="text-secondary">
              <i class="far fa-user-circle"></i> {{ question.user.name }}
            </div>
            <div class="text-success mt-2 mt-sm-0" v-if="question.has_answers">
              <i class="fas fa-check"></i> Вы ответили

```

```

        </div>
      </div>
    <hr />
    <div class="d-flex justify-content-between text-secondary">
      <small><i class="far fa-calendar"></i>
        {{ new Date(question.created_at).toLocaleDateString() }}</small>
      <small><i class="far fa-hand"></i>
        {{ question.answers_count }} ответов</small>
    </div>
  </div>
</div>
<nav class="pt-4" v-if="links.length > 3">
  <ul class="pagination justify-content-center">
    <li
      class="page-item"
      :class="{ disabled: !link.url, active: link.active }"
      v-for="(link, l) in links"
      :key="l"
    >
      <a
        class="page-link"
        href="#"
        v-html="link.label"
        @click.prevent="loadQuestions(link.url)"
      ></a>
    </li>
  </ul>
</nav>
</div>
</main>
</template>

```

Здесь мы работаем с объектом `meta.links`, так как в запросе использована пагинация. Мы разглядим функционал, когда записей станет много. Шаблон взят из бутстрапа.

Ссылка в каждом блоке ведет на страницу вопроса через `:to="/question/${question.id}"`. Сейчас она статична. Сделаем так, чтобы данные подгружались с сервера.

Метод загрузки будет похож на предыдущий. Откроем `QuestionView.vue` и напишем скрипты:

```

<script>
import { mapGetters } from "vuex";

export default {
  name: "QuestionView",
  props: ['id'],
  data() {
    return {

```



```

        question: null,
        variant_id: null
    };
},
computed: mapGetters(["user"]),
methods: {
    loadQuestion: function () {
        fetch(`http://afanasso.beget.tech/api/questions/${this.id}`, {
            headers: {
                Authorization: `Bearer ${this.user.api_token}`,
            },
        })
        .then((response) => response.json())
        .then((json) => this.question = json.data);
    },
    send: function() {
        fetch(`http://afanasso.beget.tech/api/questions/${this.id}/answer`, {
            method: 'POST',
            body: JSON.stringify({variant_id: this.variant_id}),
            headers: {
                'Content-Type': 'application/json',
                Authorization: `Bearer ${this.user.api_token}`,
            },
        })
        .then((response) => {
            if(response.ok) {
                this.$router.push(`/results/${this.id}`)
            }
        })
    },
},
created() {
    this.loadQuestion();
},
};
</script>

```

В нашем объекте помимо data и computed есть также поле props. Это переданные параметры, в данном случае id. В loadQuestion мы загружаем вопрос с сервера. Метод send отправляет постом выбранный ответ. В случае успеха сервер должен вернуть 201 статус, в случае ошибки объект с ошибками. Если успех, то есть response.ok == true, мы перекидываем на страницу результатов.

В шаблоне повесим обработчик на форму, подтянем данные о вопросе, а также свяжем через v-model радиокнопки.

А еще сделаем, чтобы кнопка отправки была активна, только когда выбран ответ. Таким образом, защита будет реализована на клиентской части тоже. Это будет через динамический :disabled="!variant_id".

```
<template>
```

```

<main class="py-5">
  <div class="container">
    <div class="card shadow" v-if="question">
      <form class="card-body p-4" action="#" @submit.prevent="send">
        <div class="d-sm-flex align-items-center justify-content-
between">
          <h2 class="mb-3 mb-sm-0">{{ question.text }}</h2>
          <small class="text-secondary"><i class="far fa-user-
circle"></i>
            {{ question.user.name }}, {{ new
Date(question.created_at).toLocaleDateString() }}
          </small>
        </div>
        <hr class="my-4">
        <label class="mb-3 d-block" v-for="variant in question.variants"
:key="variant.id">
          <input type="radio" name="variant_id" :value="variant.id"
class="me-2" v-model="variant_id">
            {{ variant.text }}
          </label>
        <hr>
        <div class="d-flex justify-content-between align-items-center">
          <button class="btn btn-primary"
:disabled="!variant_id">Ответить</button>
          <small class="text-secondary"><i class="far fa-hand"></i> {{
question.answers_count }} ответов</small>
        </div>
      </form>
    </div>
  </div>
</main>
</template>

```

Пока не выбрано:

- ☐ Да
- ☐ Нет
- ☐ Кто я?

Ответить

Когда выбрано:

- ☐ Да
- ☐ Нет
- ☒ Кто я?

Ответить

Теперь сделаем страницу результатов. Скрипт следующий. Здесь дополнительно создается вычисляемое свойство – все ответы и метод для вычисления процентов:

```
<script>
import { mapGetters } from "vuex";

export default {
  name: "ResultsView",
  props: ['id'],
  data() {
    return {
      question: null
    };
  },
  computed: {
    ...mapGetters(["user"]),
    allAnswers: function() {
      return this.question.variants.reduce((a, c) => a.concat(c.users), [])
    }
  },
  methods: {
    percent: function(value) {
      return this.allAnswers.length ? Math.round(value /
this.allAnswers.length * 100) : 0
    },
    loadResults: function () {
      fetch(`http://afanasso.beget.tech/api/questions/${this.id}/results`, {
        headers: {
          Authorization: `Bearer ${this.user.api_token}`,
        },
      },
    )
      .then((response) => response.json())
      .then((json) => {
        if(json.errors) {
          this.$router.push(`/question/${this.id}`)
        } else {
          this.question = json.data
        }
      })
    }
  }
}
```

```

    }
  });
}
},
created() {
  this.loadResults();
},
};
</script>

```

При изначально загрузке проверка – если доступ запрещен, то кидает на страницу вопроса. Все ответы через функцию reduce работы с массивами.

Отообразим в шаблоне. Кнопка Изменить ответ превращается в router-link. Значение прогресс бара передается динамически:

```

<template>
  <main class="py-5">
    <div class="container">
      <div class="card shadow" v-if="question">
        <div class="card-body p-4">
          <div class="d-sm-flex align-items-center justify-content-between">
            <h2 class="mb-3 mb-sm-0">{{ question.text }}</h2>
            <small class="text-secondary"><i class="far fa-user-circle"></i>
              {{ question.user.name }}, {{ new
Date(question.created_at).toLocaleDateString() }}
            </small>
          </div>
          <hr class="my-4">
          <div class="mb-4" v-for="variant in question.variants"
:key="variant.id">
            <div class="mb-2">{{ variant.text }} - {{
percent(variant.users.length) }}%</div>
            <div class="progress">
              <div class="progress-bar" role="progressbar"
:style="{width: `${percent(variant.users.length)}%`}" :aria-
valuenow="percent(variant.users.length)" aria-valuemin="0" aria-
valuemax="100"></div>
            </div>
            <small class="text-secondary">
              {{ variant.users.slice(0, 3).map(v => v.name).join(', ')
            }}
            <span v-if="variant.users.length > 3">и еще {{
variant.users.length - 3 }}</span>
            </small>
          </div>
          <hr>
          <router-link :to="`/question/${id}`" class="btn btn-sm btn-
light">

```

```

        <i class="fas fa-pencil"></i>
        Изменить свой ответ
      </router-link>
    </div>
  </div>
</div>
</main>
</template>

```

Теперь, кстати, можно сделать так, чтобы при переходе с главной на страницу вопроса была условная адресация. Если уже есть ответ, вести на результаты, иначе на вопрос. Вернемся в HomeView.vue и отредактируем ссылку на вопрос следующим образом:

```

<router-link
:to="`${question.has_answers ? 'results' : 'question'}/${question.id}`"
class="h4 text-decoration-none">
  {{ question.text }}
</router-link>

```

Теперь в зависимости от статуса будут разные ссылки.

И у нас осталось только создание вопроса. Перейдем в CreateView.vue. Создадим скрипты:

```

<script>
import { mapGetters } from "vuex";

export default {
  name: 'CreateView',
  data() {
    return {
      question: {
        text: '',
        variants: [{
          text: ''
        }]
      },
      errors: {}
    }
  },
  computed: mapGetters(["user"]),
  methods: {
    addVariant: function() {
      this.question.variants.push({text: ''})
    },
    deleteVariant: function(index) {
      this.question.variants.splice(index, 1)
    },
    send: function() {

```

```

    this.errors = {}
    fetch(`http://afanasso.beget.tech/api/questions`, {
      method: 'POST',
      body: JSON.stringify(this.question),
      headers: {
        'Content-Type': 'application/json',
        Authorization: `Bearer ${this.user.api_token}`,
      },
    },
  })
  .then((response) => {
    if(response.ok) {
      this.$router.push('/')
    } else {
      response.json()
        .then(json => this.errors = json.errors)
    }
  })
}
}
}
</script>

```

Вспомогательные методы addVariant и deleteVariant похожи на те, что использовали в предыдущем модуле по todo list.

В случае успеха отправляем на главную, иначе отображаем ошибки.

Теперь все это привязать к шаблону:

```

<template>
  <main class="py-5">
    <div class="container">
      <div class="card shadow">
        <form class="card-body p-4" action="#" @submit.prevent="send">
          <h2 class="mb-4">Новый опрос</h2>
          <textarea class="form-control mb-4" :class="{ 'is-invalid': errors.text}" placeholder="Текст вопроса" v-model="question.text"></textarea>
          <span class="invalid-feedback">{{ errors.text }}</span>
          <div class="mb-4 row" v-for="(variant, v) in question.variants" :key="v">
            <div class="col">
              <input type="text" class="form-control" :class="{ 'is-invalid': errors[`${variants.${v}.text}`]" :placeholder="`Вариант #${v + 1}`" v-model="variant.text">
              <span class="invalid-feedback">{{ errors[`${variants.${v}.text}`] }}</span>
            </div>
            <div class="col-auto">
              <button type="button" class="btn btn-light" @click="deleteVariant(v)">
                <i class="fas fa-trash-alt"></i>
            </div>
          </div>
        </form>
      </div>
    </div>
  </main>
</template>

```

```

        </button>
      </div>
    </div>
    <a href="#" class="btn btn-light" @click.prevent="addVariant">
      <i class="fas fa-add"></i>
      Добавить вариант ответа
    </a>
    <hr>
    <div class="text-end">
      <button class="btn btn-primary">Опубликовать</button>
    </div>
  </form>
</div>
</div>
</main>
</template>

```

Здесь на кнопки и форму повесили обработчики, поля ввода связали через v-model и условно отображаем ошибки.

ToU Surveys
Simple user

Новый опрос

Текст вопроса

The text field is required.

123

456

+ Добавить вариант ответа

Опубликовать

Итак, реализован функционал по:

1. Логину и ограничению доступа
2. Регистрации
3. Просмотру всех опросов
4. Отправки ответа
5. Просмотра результатов (только для ответивших)
6. Созданию своих опросов.

Осталось сбилдить проект. Выполняем в терминале команду

```
npm run build
```

В папке dist будет готовый проект. Скопируем эту папку в domains на Open Server, переименуем в surveys-frontend.

В этой папке создать .htaccess с кодом:

```
<IfModule mod_rewrite.c>
RewriteEngine On
RewriteBase /
RewriteRule ^index\.html$ - [L]
RewriteCond %{REQUEST_FILENAME} !-f
RewriteCond %{REQUEST_FILENAME} !-d
RewriteRule . /index.html [L]
</IfModule>
```

Теперь запустим сервер, обратимся в браузере по адресу <http://surveys-frontend>

Наше приложение должно работать.

ToU Surveys

Авторизация

Ваш email

Ваш пароль

Войти

Нет аккаунта? [Регистрация](#)