

Министерство образования Республики Беларусь

Учреждение образования «Белорусский государственный университет
информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра информатики

Дисциплина: Модели данных и системы управления базами данных

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовому проекту

на тему

«Сайт с курсами о здоровом питании»

Студент гр. 753501

И. А. Винничек

Руководитель

И. А. Удовин

Минск 2020

Содержание

Введение	3
1. Анализ предметной области	4
1.1. Краткие теоретические сведения	4
1.2. Технологии и средства разработки	5
1.3. Модули программы и их описание	8
2. Реализация баз данных в приложении	10
2.1. Mongodb Database	10
2.1.1. Авторизация/Регистрация	10
2.1.2. Добавление треков в избранное, список «Вы слушали»	15
2.1.3. Добавление треков в базу данных	34
2.1.4. Отображение семинаров и новостей	20
Заключение	...32
Список использованной литературы.....	33
Приложение 1. Исходные файлы.....	34

Введение

Интернет представляет собой идеальный источник для получения информации, а также является великолепным инструментом для коммуникации и построения собственного бизнеса. Другими словами, интернет предоставляет для человека большие возможности, которые при правильном использовании могут пойти ему на пользу. В рамках данной курсовой работы я рассмотрел Сеть как возможность получения пользы для здоровья. Тема моей работы: Сайт с курсами о здоровом питании, своего рода площадка, на которой пользователи смогут прочитать про здоровое питание, новости связанные с этим, а также записаться на семинары. Основной целью данной работы является свидетельство о моей квалифицированности в области баз данных.

1. Анализ предметной области

1.1. Краткие теоретические сведения

База данных — это организованная структура, предназначенная для хранения, изменения и обработки взаимосвязанной информации, преимущественно больших объемов. Базы данных активно используются для динамических сайтов со значительными объемами данных — часто это интернет-магазины, порталы, корпоративные сайты. Такие сайты обычно разработаны с помощью серверного языка программирования (как пример, PHP) или на основе CMS (как пример, WordPress), и не имеют готовых страничек с данными по аналогии с HTML-сайтами. Странички динамических сайтов формируются «на лету» в результате взаимодействия скриптов и баз данных после соответствующего запроса клиента к веб-серверу.

Нереляционная база данных — это база данных, в которой в отличие от большинства традиционных систем баз данных не используется табличная схема строк и столбцов. В этих базах данных применяется модель хранения, оптимизированная под конкретные требования типа хранимых данных. Например, данные могут храниться как простые пары "ключ — значение", документы JSON или граф, состоящий из ребер и вершин.

1.2. Технологии и средства разработки

Visual Studio Code — редактор исходного кода, разработанный Microsoft для Windows, Linux и macOS. Позиционируется как «лёгкий» редактор кода для кроссплатформенной разработки веб- и облачных приложений. Включает в себя отладчик, инструменты для работы с Git, подсветку синтаксиса, IntelliSense и средства для рефакторинга. Имеет широкие возможности для кастомизации: пользовательские темы, сочетания клавиш и файлы конфигурации. Распространяется бесплатно, разрабатывается как программное обеспечение с открытым исходным кодом, но готовые сборки распространяются под проприетарной лицензией. Visual Studio Code основан на Electron и реализуется через веб-редактор Monaco, разработанный для Visual Studio Online (см. рис. 1) [4].



Рисунок 1. Иконка VS Code

HTML5 (HyperText Markup Language) — язык для структурирования и представления содержимого всемирной паутины. Это пятая версия HTML. Хотя стандарт был завершён (рекомендованная версия к использованию) только в 2014 году (предыдущая, четвёртая, версия опубликована в 1999 году), уже с 2013 года браузерами оперативно осуществлялась поддержка, а разработчиками — использование рабочего стандарта (англ. HTML Living Standard). Цель разработки HTML5 — улучшение уровня поддержки мультимедиа-технологий с одновременным сохранением обратной совместимости, удобочитаемости кода для человека и простоты анализа для парсеров (см. рис. 2) [5].



Рисунок 2. Иконка HTML5

CSS3 (Cascading Style Sheets) — формальный язык описания внешнего вида документа (веб-страницы), написанного с использованием языка разметки (чаще всего HTML или XHTML). Также может применяться к любым XML-документам, например, к SVG или XUL. CSS используется создателями веб-страниц для задания цветов, шрифтов, стилей, расположения отдельных блоков и других аспектов представления внешнего вида этих веб-страниц. Основной целью разработки CSS являлось отделение описания логической структуры веб-страницы (которое производится с помощью HTML или других языков разметки) от описания внешнего вида этой веб-страницы (которое теперь производится с помощью формального языка CSS) (см. рис. 3) [6].



Рисунок 3. Иконка CSS3

JavaScript — мультипарадигменный язык программирования. Поддерживает объектно-ориентированный, императивный и функциональный стили. Является реализацией стандарта ECMAScript (стандарт ECMA-262). JavaScript обычно используется как встраиваемый язык для программного доступа к объектам приложений. Наиболее широкое применение находит в браузерах как язык сценариев для придания интерактивности веб-страницам. Основные архитектурные черты: динамическая типизация, слабая типизация, автоматическое управление памятью, прототипное программирование, функции как объекты первого класса. (см. рис. 4) [7].



Рисунок 4. Иконка JavaScript

React — JavaScript-библиотека с открытым исходным кодом для разработки пользовательских интерфейсов.

React разрабатывается и поддерживается Facebook, Instagram и сообществом отдельных разработчиков и корпораций.

React может использоваться для разработки **одностраничных** и мобильных приложений. Его цель — предоставить высокую скорость, простоту и масштабируемость. В качестве библиотеки для разработки пользовательских интерфейсов React часто используется с другими библиотеками, такими как MobX, **Redux** и GraphQL

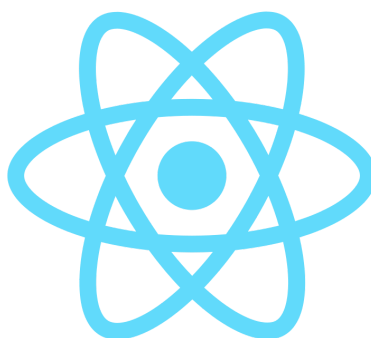


Рисунок 5. Иконка React

mLab — это полностью управляемая служба облачной базы данных, в которой хранятся базы данных MongoDB. mLab работает на облачных провайдерах Amazon, Google и Microsoft Azure и сотрудничает с компаниями, предоставляющими услугу platform-as-a-service.

В мае 2011 года mLab получила 3 миллиона долларов в первом раунде финансирования от Foundry Group, Baseline Ventures, Upfront Ventures, Freestyle Capital. В октябре 2012 года mLab получила последующие инвестиции в размере 5 миллионов долларов, а вскоре после этого, mlab был назван Network World одной из 10 самых полезных облачных баз данных, наравне с Amazon Web Services, Google Cloud SQL, Microsoft Azure, Rackspace и другими.

В июне 2014 года MongoDB Inc. объявила о выпуске полностью доступного для настройки MongoDB-as-a-Service в магазине Microsoft Azure. Предложение предоставляется в сотрудничестве с Microsoft и mLab.

В феврале 2016 года mLab изменил свое название с MongoLab на mLab.



Рисунок 6. Иконка mLab

Webpack — это сборщик модулей [JavaScript](#) с [открытым исходным кодом](#). Он создан в первую очередь для JavaScript, но может преобразовывать внешние ресурсы, такие как HTML, CSS и изображения, если включены соответствующие загрузчики^[7]. webpack принимает модули с зависимостями и генерирует статические ресурсы, представляющие эти модули^[8].

webpack принимает зависимости и генерирует граф зависимостей, позволяющий веб-разработчикам использовать модульный подход для разработки своих веб-приложений. Его можно использовать из командной строки или настроить с помощью файла конфигурации с именем *webpack.config.js*. Этот файл используется для определения правил, плагинов и т. д. для проекта. (webpack сильно расширяем с помощью правил, которые позволяют разработчикам писать задачи, которые они хотят выполнять при объединении файлов.)



Рисунок 7. Иконка Webpack

MongoDB — [документоориентированная система управления базами данных](#), не требующая описания схемы таблиц. Считается одним из

классических примеров [NoSQL](#)-систем, использует [JSON](#)-подобные документы и схему базы данных. Применяется в веб-разработке, в частности, в рамках [JavaScript](#)-ориентированного стека [MEAN](#).



Рисунок 8. Иконка MongoDB

1.3. Модули программ и их описание

В рамках исследования эффективности создания базы данных в данном курсовом проекте реализовано несколько модулей, каждый из которых выполняет определенную задачу:

- **Клиент (sayt)**, который представлен блоками кода HTML, CSS, JS, REACT, Webpack, Babel, и необходим для выполнения задач отрисовки элементов в браузере пользователя.
- **Клиент (admin panel)**, который представлен блоками кода HTML, CSS, JS, REACT, Webpack, Babel, REACT, REDUX и необходим для выполнения задач отрисовки элементов в браузере пользователя. Также выполняет функцию взаимодействия с пользователем, включая запись определенных полей в mongodb.
- **Локальный Сервер**, который представлен блоком Node js, который служит для решения задач, связанных с mongodb.

Кроме того, в состав проекта вошло множество файлов, которые отвечают за различные операции в программе. Рассмотрим основные из них:

- **index.js** – основной исходный файл, в котором происходит подключение react.
- **app.js** – файл, который содержит логику системы маршрутизации, рендеринга уникального контента, ссылки на все страницы, имеющиеся в проекте. Отвечает за постоянное нахождение хедера и футера на своих местах.
- **server.js** – файл локального сервера, написанного с помощью фреймворка Node js. Решает задачи, связанные с mongodb.

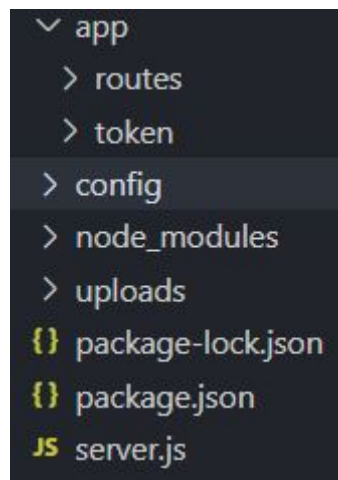


Рисунок 9. Каталог сервера

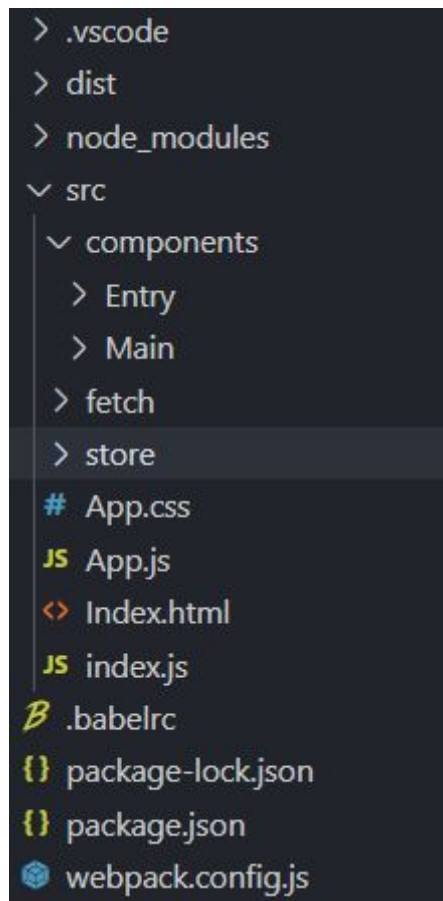


Рисунок 10. Каталог клиент(admin panel)

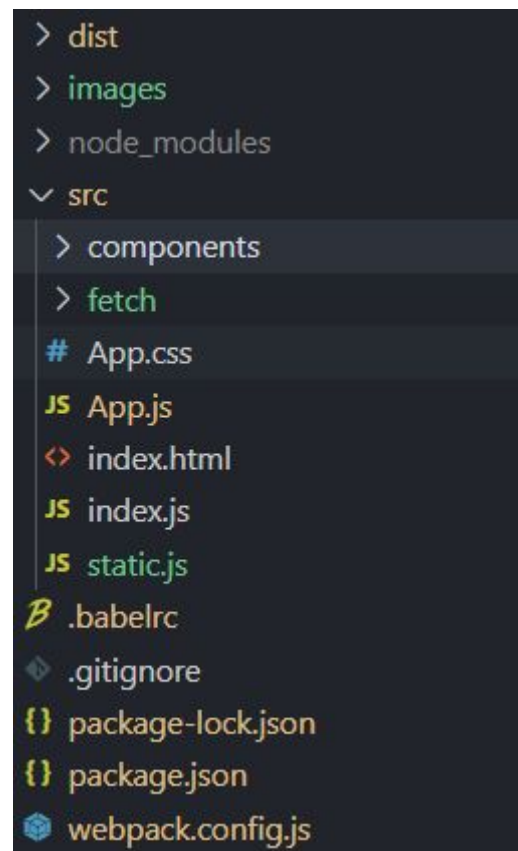


Рисунок 11. Каталог клиент(sayt)

2. Реализация баз данных в приложении

2.1. MongoDB Database

База данных **mongodb** -- для хранения своей бд я выбрал **mLab**. Данные хранятся в формате **JSON** и синхронизируются в реальном времени с каждым подключенным клиентом.

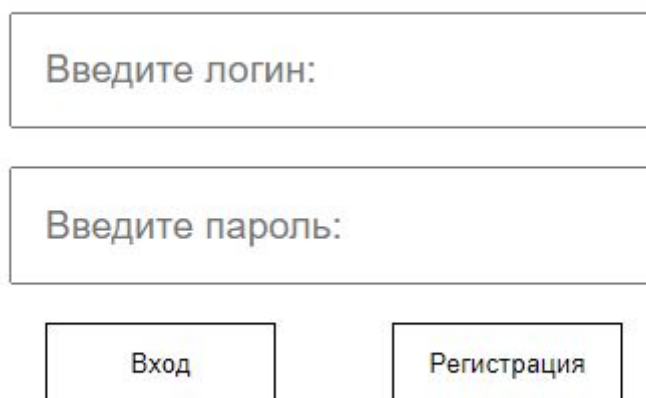
Для начала работы с данной базой данных нужно сперва подключить клиент к хостингу в корневом файле, также настроить **CORS** ну и для небольшой простоты я подключил **express** и **body-parser**:

```
1  const express      = require('express');
2  const MongoClient  = require('mongodb').MongoClient;
3  const bodyParser   = require('body-parser');
4  const db           = require('./config/db');
5  const app          = express();
6  const port         = 8000;
7  const { static } = require('express');
8  app.use(function (req, res, next) {
9
10     res.setHeader('Access-Control-Allow-Origin', '*');
11
12     res.setHeader('Access-Control-Allow-Methods', 'GET, POST, OPTIONS, PUT, PATCH, DELETE');
13
14     res.setHeader('Access-Control-Allow-Headers', 'X-Requested-With,content-type, x-access-token');
15
16     res.setHeader('Access-Control-Allow-Credentials', true);
17     if (req.method === "OPTIONS") {
18         return res.status(200).end();
19     }
20     next();
21 });
22 app.use('/images/', static('./uploads/'));
23 app.use(bodyParser.json());
24 app.use(bodyParser.urlencoded({ extended: true }));
25 MongoClient.connect(db.url, (err, database) => {
26     if (err) return console.log(err)
27     require('./app/routes')(app, database);
28     app.listen(port, () => {
29         console.log('We are live on ' + port);
30     });
31 })
```

Рисунок 12. Конфигурация удаленной базы данных

2.1.1. Авторизация/Регистрация

Сперва рассмотрим систему авторизации и регистрации. Код находится в файле **Entry.jsx**. На странице авторизации имеются следующие поля:



The form consists of two input fields stacked vertically. The first field is labeled "Введите логин:" (Enter login:). The second field is labeled "Введите пароль:" (Enter password:). Below these fields are two buttons: "Вход" (Login) on the left and "Регистрация" (Registration) on the right.

Рисунок 13. Поля ввода для авторизации

При нажатии на кнопку «Вход» срабатывает обработчик события, который проверяет данные на корректность и отправляет в функцию **SignInFetch**:

```
const sendData = () =>{
  console.log(userLogin,userPassword)
  signInFetch({login: userLogin, password: userPassword}).then(data =>{
    if(data.status){
      localStorage.setItem('token', data.token)
      setStatusRedirect(true)
      alert("welcome")
      window.location.replace('http://localhost:8080/#/target')
    } else {
      alert("ne welcome")
    }
  })
}
```

Рисунок 14. Код обработчика события нажатия на кнопку Вход

За проверку нового пользователя в базе данных отвечает метод **signInFetch**. В качестве параметров он принимает два значения: логин пользователя и его пароль. Метод отправляет данные на сервер где происходит проверка введенные значения с имеющимися в базе данных. При успешном входе клиенту отправляется токен.

```
export async function signInFetch(User){
  const response = await fetch(usersCheck,{
    method: "POST",
    body: JSON.stringify(User),
    headers: {
      "Content-type": "application/json"
    },
  })
  const data = await response.json()
  return data
}
```

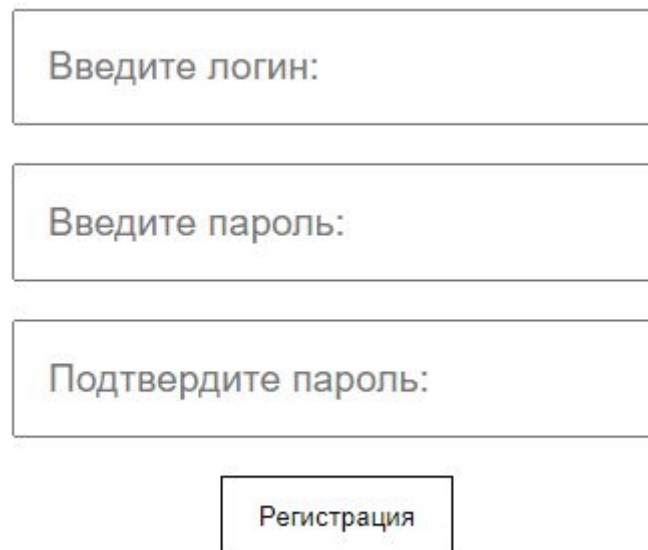
Рисунок 15. Отправка данных на сервер

На сервере в свою очередь вызывается и как я писал выше, если всё хорошо возвращается токен

```
app.post('/userschecking', (req,res) => {
  const details = { 'login': req.body.login };
  const password = req.body.password
  someDB.collection('users').findOne(details, (err, item) => {
    if (err) {
      res.send({'error':'An error has occurred'});
    } else {
      if(bcrypt.hashSync(password, config.secret) === item.password){
        var token = jwt.sign({ id: item._id }, config.secret, {
          expiresIn: 86400 // expires in 24 hours
        });
        res.status(200).send({status: true, token: token})
        console.log("нужный пароль")
      } else {
        res.status(401).send({status: false, token: none})
        console.log("неверный пароль");
      }
    }
  })
});
```

Рисунок 16. Обработка запроса на сервере

С регистрацией дела обстоят немного иначе:



Введите логин:

Введите пароль:

Подтвердите пароль:

Регистрация

Рисунок 17. Поля ввода для регистрации

```
const entry = () => {  
  if(counter === 0){  
    setCounter((preState) => counter + 1)  
    setRegStatus((preState) => !preState)  
    setLoginStatus((preState) => !preState)  
  } else if(counter === 1){  
    if((userPassword === userPasswordAgain) && (userPassword !== '')) {  
      signUpFetch({login: userLogin, password: userPassword, status: 3}).then(  
        data => {  
          setCounter(0)  
          setRegStatus((preState) => !preState)  
          setLoginStatus((preState) => !preState)  
          alert("Регистрация прошла успешно!")  
        })  
      ).catch(response => alert("Юзер с таким логином уже зарегистрирован"))  
    } else {  
      alert("Пароли не совпадают!")  
    }  
  }  
}
```

Рисунок 18. Код обработчика события нажатия на кнопку регистрации

За проверку нового пользователя в базе данных отвечает метод **signUpFetch**. В качестве параметров он принимает три значения: логин пользователя, его пароль и статус(роль пользователя). Метод отправляет данные на сервер где происходит проверка введенные значения с имеющимися в базе данных. При успешной регистрации пользователя перекидывает на страницу Входа.

```
app.post('/usersadd', (req,res) => {
  const details = { 'login': req.body.login };
  const newPassword = bcrypt.hashSync(req.body.password, config.secret)

  const user = { login: req.body.login, password: newPassword, status: updateStatusUser(req.body.status)}
  someDB.collection('users').findOne(details, (err, item) => {
    if (err) {
      res.send({'error': 'An error has occurred'});
    } else {
      if(item){
        res.status(403).send(403)
      } else {
        someDB.collection('users').insert(user, (err,result) =>{
          if(err){
            res.send({'error': 'An error has occurred' });
          } else {
            res.status(200).send(result.ops[0]);
          }
        })
      }
    }
  });
});
```

Рисунок 19. Добавление нового юзера

Также можно давать разные права юзерам имеется **admin**, **user** и **moder**. Есть возможность удаления пользователя

<div> <div>ЮЗЕРЫ</div> <div>СЕМИНАРЫ</div> <div>НОВОСТИ</div> </div>			
НОМЕР #	ЛОГИН	РОЛЬ	ИЗМЕНИТЬ
1	i	user	<div>удалить</div> <div>поднять</div> <div>опустить</div>

Рисунок 20. Удаление пользователя

```

app.post('/usersadd', (req,res) => {
  const details = { 'login': req.body.login };
  const newPassword = bcrypt.hashSync(req.body.password, config.secret)

  const user = { login: req.body.login, password: newPassword, status: updateStatusUser(req.body.status)}
  someDB.collection('users').findOne(details, (err, item) => {
    if (err) {
      res.send({'error': 'An error has occurred'});
    } else {
      if(item){
        res.status(403).send(403)
      } else {
        someDB.collection('users').insert(user, (err,result) =>{
          if(err){
            res.send({'error': 'An error has occurred' });
          } else {
            res.status(200).send(result.ops[0]);
          }
        })
      }
    }
  });
});
})

```

Рисунок 21. Создание нового пользователя

```

app.delete('/users/:id', (req, res) => {
  const id = req.params.id;
  const details = { '_id': new ObjectId(id) };
  someDB.collection('users').removeOne(details, (err, item) => {
    if (err) {
      res.send({'error': 'An error has occurred'});
    } else {
      res.send('user ' + id + ' deleted!');
    }
  });
});
});

```

Рисунок 22. Удаление пользователя

```

app.post('/userschange', (req, res) => {
  const id = req.body.id;
  const details = { '_id': new ObjectId(id) };
  const user = { login: req.body.body, password: req.body.title };
  someDB.collection('users').findOne(details, (err, item) => {
    item.status = updateStatusUser(req.body.status)
    someDB.collection('users').update(details, item, (err, result) => {
      if (err) {
        res.send({'error': 'An error has occurred'});
      } else {
        res.send(user);
      }
    });
  });
});
});

```

Рисунок 23. Изменение прав

2.1.2 Добавление семинаров и новостей

Добавление семинаров и новостей сильно отличается от авторизации и регистрации: если авторизация и регистрация использовали только `react`, то для семинаров и новостей нужно подключить **Redux**.

ЮЗЕРЫ СЕМИНАРЫ НОВОСТИ				
НОМЕР #	ID:	НАЗВАНИЕ:	ДАТА:	создать
1	5fbcf6569e730948fcc67895	77777	1.1.1970	удалить изменить
2	5fbd05a79e730948fcc67896	qqqqq	1.1.1970	удалить изменить
3	5fc7bd9570953d0e08552455	M0RDIR.pug	1.1.2021	удалить изменить
4	5fcbad3658a0994268f2537a	привет	10.12.2020	удалить изменить
5	5fb638609b9d4c3ffc450f36	11111	1.1.1970	удалить изменить
6	5fbceb9f9e730948fcc67894	3333333333	3.11.2020	удалить изменить

Рисунок 24. Общий список семинаров

ЮЗЕРЫ СЕМИНАРЫ НОВОСТИ				
НОМЕР #	ID:	НАЗВАНИЕ:	ДАТА:	создать
1	5fcb855a8d238c36ccd31492	Есть ли польза детокса? Мнение врача	5.12.2020	удалить изменить
2	5fc7cf2a77f1ec41f8105eee	Есть ли польза детокса? Мнение врача	5.12.2020	удалить изменить
3	5fcbadb858a0994268f2537c	postSeminar(seminar)	5.12.2020	удалить изменить

Рисунок 25. Общий список новостей

Т.к пишем на **React**, то всё на нашей странице стараемся бить на компоненты, каждый компонент отвечает за определенный функционал на сайте, помимо этого каждый компонент мы еще бьем на два компонента: тупой и умный. Тупой отвечает непосредственно за отрисовку элемента, умный же связывается с базой данных, работает с **redux** и передает данные в тупой.


Рассмотрим на примере редактирования основной информации у семинара:

Основная информация:

Название:

qqqqq

Дата проведения:

дд . мм . гggg 

Авторы:

qqq

Формат:

qqq


Количество мест:

qqq

Прикрепить картинку:

Выберите файл

Файл не выбран



Активация

Чтобы активир

раздел "Парам

Рисунок 26. Редактирование основной информации семинара

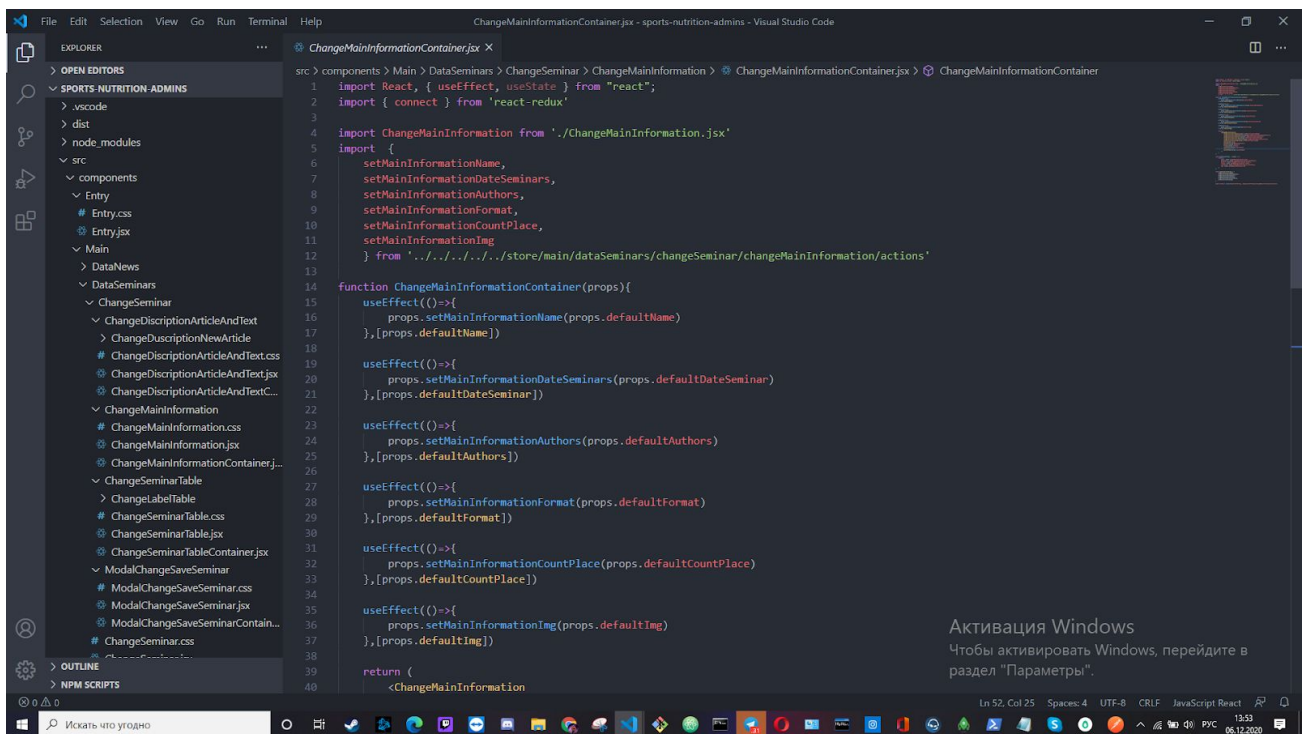


Рисунок 27. Умный компонент

Рисунок 20 демонстрируют работу умного компонента **ChangeMainInformationContainer**: он принимает данные с **Redux** (**name**, **author**, **img...**), принимает **action**(функции обработки данных в зависимости от действий пользователя), а также **dispatcher**(связь action с нашим хранилищем). Как только мы получаем данные или они обновляются мы их пробрасываем в компонент **ChangeMainInformation**, где непосредственно их и отрисовываем:

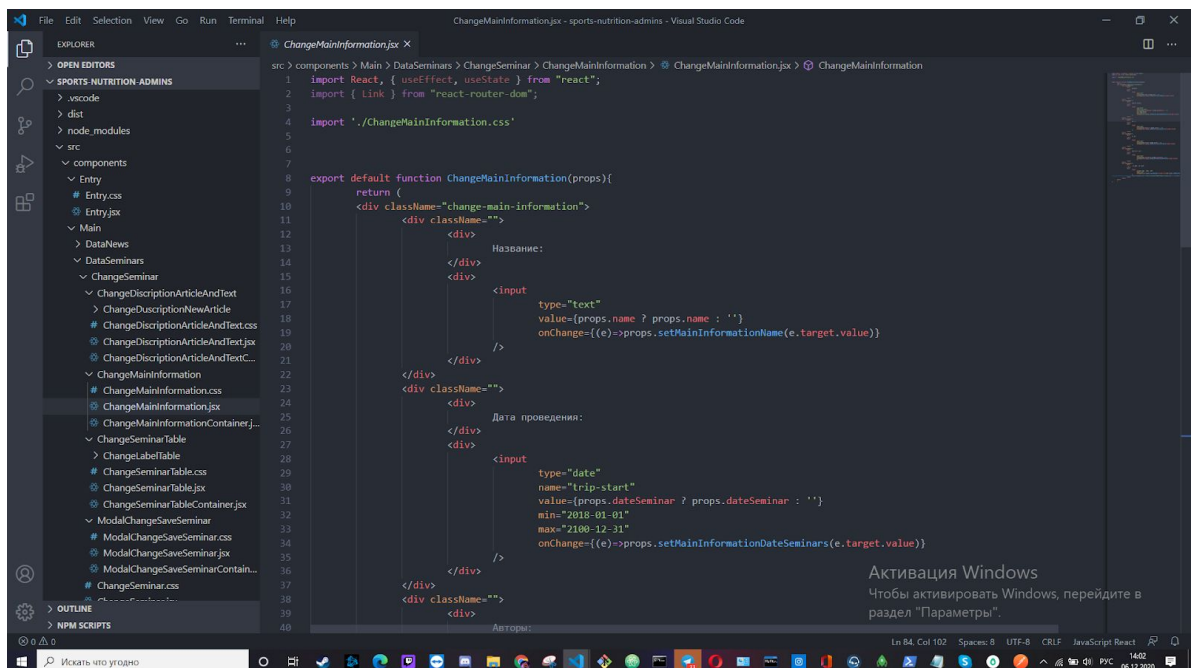


Рисунок 28. Тупой компонент

Фиксирование изменений осуществляется через **Redux** (чтобы мы могли в любой момент времени получить данные для любого компонента)

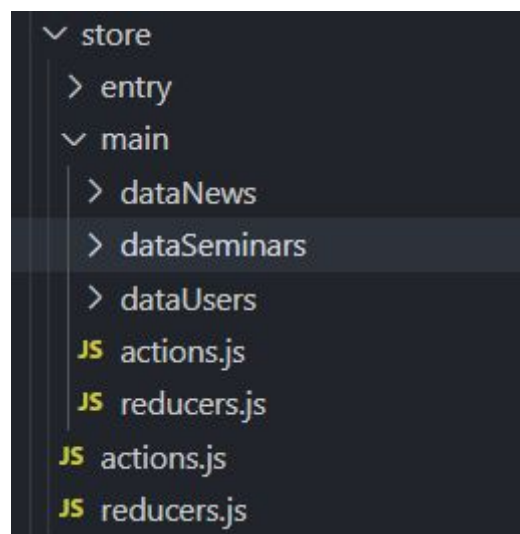
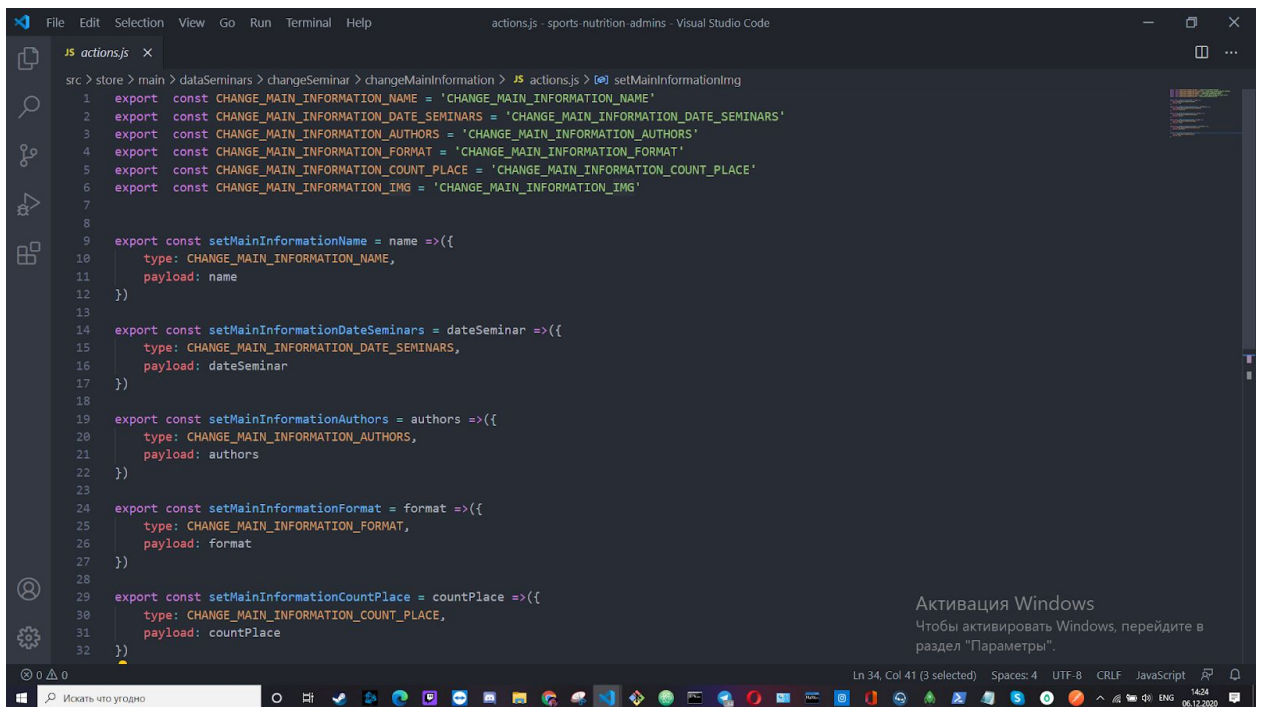


Рисунок 29. Структура Redux

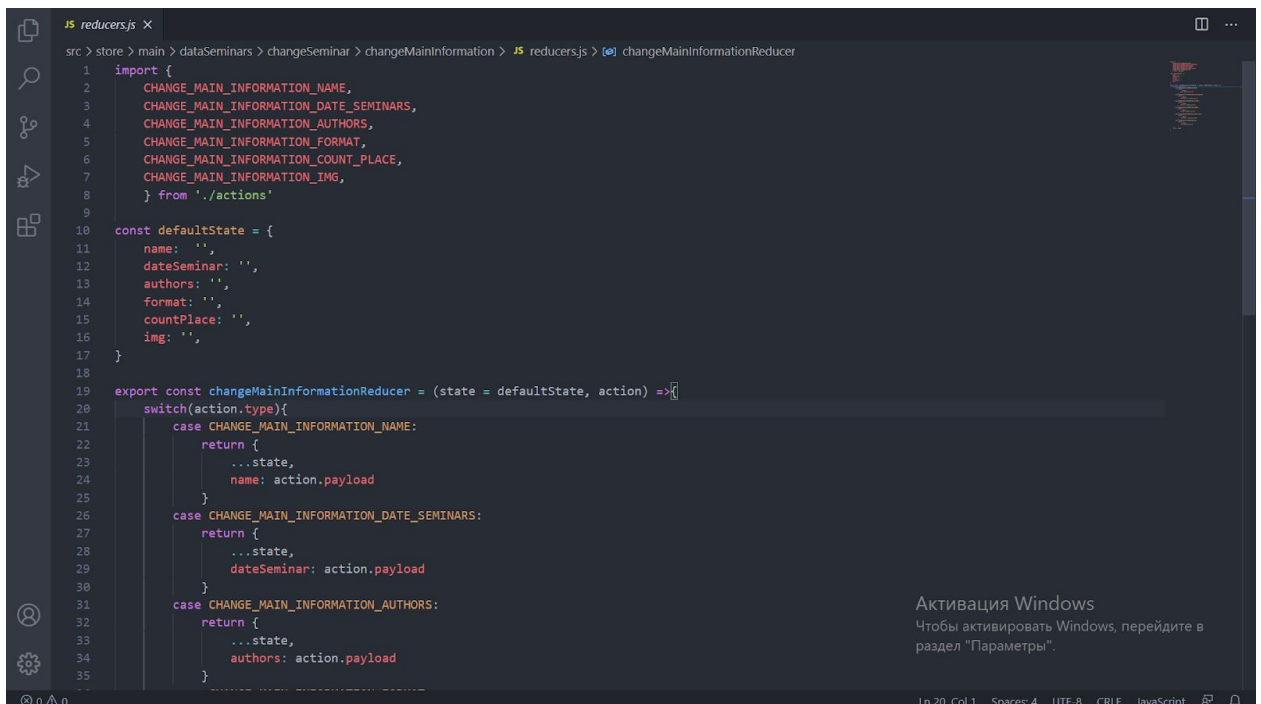
Для каждого компонента есть свой **actions** и **reducers**:



```
1 export const CHANGE_MAIN_INFORMATION_NAME = 'CHANGE_MAIN_INFORMATION_NAME'
2 export const CHANGE_MAIN_INFORMATION_DATE_SEMINARS = 'CHANGE_MAIN_INFORMATION_DATE_SEMINARS'
3 export const CHANGE_MAIN_INFORMATION_AUTHORS = 'CHANGE_MAIN_INFORMATION_AUTHORS'
4 export const CHANGE_MAIN_INFORMATION_FORMAT = 'CHANGE_MAIN_INFORMATION_FORMAT'
5 export const CHANGE_MAIN_INFORMATION_COUNT_PLACE = 'CHANGE_MAIN_INFORMATION_COUNT_PLACE'
6 export const CHANGE_MAIN_INFORMATION_IMG = 'CHANGE_MAIN_INFORMATION_IMG'
7
8
9 export const setMainInformationName = name =>({
10   type: CHANGE_MAIN_INFORMATION_NAME,
11   payload: name
12 })
13
14 export const setMainInformationDateSeminars = dateSeminar =>({
15   type: CHANGE_MAIN_INFORMATION_DATE_SEMINARS,
16   payload: dateSeminar
17 })
18
19 export const setMainInformationAuthors = authors =>({
20   type: CHANGE_MAIN_INFORMATION_AUTHORS,
21   payload: authors
22 })
23
24 export const setMainInformationFormat = format =>({
25   type: CHANGE_MAIN_INFORMATION_FORMAT,
26   payload: format
27 })
28
29 export const setMainInformationCountPlace = countPlace =>({
30   type: CHANGE_MAIN_INFORMATION_COUNT_PLACE,
31   payload: countPlace
32 })
```

Рисунок 30. actions

При вводе данных в `input` вызывается **action** с определенным **type** и нужное значение кладется в поле **payload**.



```
1 import {
2   CHANGE_MAIN_INFORMATION_NAME,
3   CHANGE_MAIN_INFORMATION_DATE_SEMINARS,
4   CHANGE_MAIN_INFORMATION_AUTHORS,
5   CHANGE_MAIN_INFORMATION_FORMAT,
6   CHANGE_MAIN_INFORMATION_COUNT_PLACE,
7   CHANGE_MAIN_INFORMATION_IMG,
8 } from './actions'
9
10 const defaultState = {
11   name: '',
12   dateSeminar: '',
13   authors: '',
14   format: '',
15   countPlace: '',
16   img: '',
17 }
18
19 export const changeMainInformationReducer = (state = defaultState, action) =>{
20   switch(action.type){
21     case CHANGE_MAIN_INFORMATION_NAME:
22       return {
23         ...state,
24         name: action.payload
25       }
26     case CHANGE_MAIN_INFORMATION_DATE_SEMINARS:
27       return {
28         ...state,
29         dateSeminar: action.payload
30       }
31     case CHANGE_MAIN_INFORMATION_AUTHORS:
32       return {
33         ...state,
34         authors: action.payload
35       }
36   }
37 }
```

Рисунок 3. reducers

После вызова action данные передаются в **reducers**, где всё очень просто: по **type** выбирается нужный **case**, после этого перезаписываем именно то поля для которого пришел **payload**. И наш store(хранилище всех данных redux) обновляется.

После заполнения всех полей и таблиц есть кнопка «Сохранить семинар»

The screenshot shows a web form for creating a seminar. It includes several input fields for 'Обяз' (Required) with values like '11', '1', '111', and '1111'. There are buttons labeled 'удалить' (delete) next to each field. Below the fields are two buttons: 'добавить обзор с текстом к этому заголовку' and 'добавить новый заголовок'. A large green button labeled 'Сохранить семинар' is prominently displayed. At the bottom, there is a table with two rows of data, each with a 'удалить' button and an 'изменить' button. A Windows activation watermark is visible on the right side.

Рисунок 31. Заполненный семинар

После нажатия на кнопку открывается предпросмотр семинара и если всё устраивает нажимаем кнопку «Сохранить »

The screenshot shows a confirmation dialog box titled 'Проверьте данные:' (Check the data:). It contains three sections: 'Основная информация:' (Main information), 'Информация для таблицы:' (Information for the table), and 'Информация о семинаре:' (Information about the seminar). Each section lists various details like name, date, author, and format. At the bottom of the dialog are two buttons: 'Сохранить' (Save) and 'Отмена' (Cancel). The background shows the same seminar form as in Figure 31, but it is dimmed.

Рисунок 32. Подтверждение изменений

При нажатии на кнопку «Сохранить », обертываем данные

```
const sendData = ()=>{
  const formData = new FormData();
  formData.append('description', JSON.stringify({
    allArticles: props.allArticles,
    allParagraph: props.allParagraph,
    authors: props.authors,
    countPlace: props.countPlace,
    dateSeminar: props.dateSeminar,
    format: props.format,
    name: props.name,
    img: props.img,
    keyTable: props.keyTable,
    valuesTable: Array.from(props.valuesTable),
    id: props.id
  }));

  formData.append('image', props.image);
  props.fetchPOSTSeminar(formData)
}
```

Рисунок 33. Обертка данных

С помощью функции **fetchPOSTSeminar** мы отправляем данные в postSeminar

```
const fetchPOSTSeminar = (seminar) =>{
  postSeminar(seminar).then(data => location.reload())
}
```

Рисунок 34. Передача данных в postSeminar

Вызываем функцию **postSeminar**, а после обновляем страницу

```
export async function postSeminar(formData){
  const response = await fetch(urlPOSTSeminars,{
    method: "POST",
    body: formData,
  })
  const data = await response.json()
  return data
}
```

Рисунок 35. Передача данных на сервер

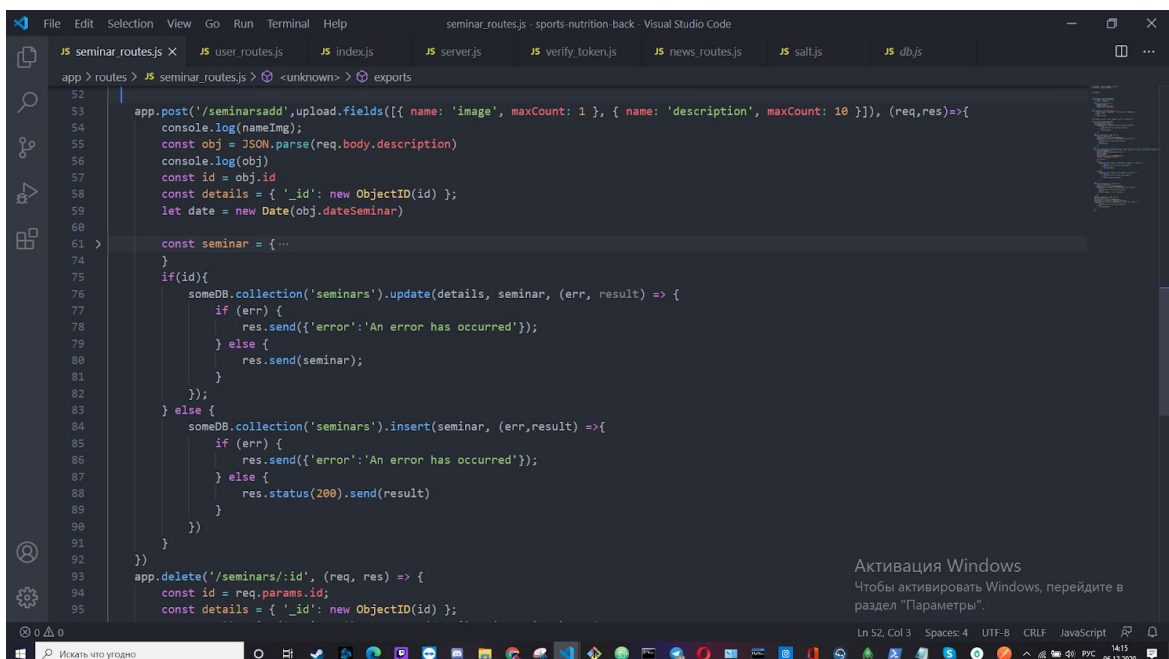


Рисунок 36. Обработка запроса на сервере

Как только к нам приходит семинар на сервер мы его распарсиваем отделяем картинки от текста. Сразу проверяем новый ли это семинар или же измененный старый и если всё хорошо сохраняем его в базу данных.

Хотелось бы добавить, что реализовано добавление картинок: на фронте мы все данные которые отправляем оборачиваем в **FormData** (потому что в **redux** без плагинов нельзя записать **file**, а также файл нельзя передать в **json** (только если кодировкой, но мне это не понравилось)) поэтому был выбран вариант **FormData**. С помощью команды **append** мы добавляем данные в **FormData**, в **description** кладем всю текстовую информацию, а в **image** - картинку.

```
const sendData = ()=>{
  const formData = new FormData();
  formData.append('description', JSON.stringify({
    allArticles: props.allArticles,
    allParagraph: props.allParagraph,
    authors: props.authors,
    countPlace: props.countPlace,
    dateSeminar: props.dateSeminar,
    format: props.format,
    name: props.name,
    img: props.img,
    keyTable: props.keyTable,
    valuesTable: Array.from(props.valuesTable),
    id: props.id
  }));

  formData.append('image', props.image);
  props.fetchPOSTSeminar(formData)
}
```

Рисунок 37. обертка данных с помощью FormData

На сервере тоже не всё так просто, нам нужен специальный модуль **multer** для работы с файлами которые к нам приходят с клиента, в **storage** - описываем сохранение картинок, в **fileFilter** - фильтруем картинки.

Система сохранения картинок выглядит так, на фронте мы отправляем картинку файлом и в **description** записываем название, она приходит на сервер сохраняется и на рендер мы же отправляем просто название картинки

Что касается добавления новостей, единственная разница с семинарами, что там меньше информации и больше картинок. А так по своей структуре добавление, удаление, изменение, сохранение тоже самое.

2.1.4. Отображение семинаров и новостей

Для отображения семинаров и новостей был написан другой сайт, хотя это всё можно увидеть и через редактирование семинаров. На основном сайте нас встречает главная страница переходим в раздел семинары

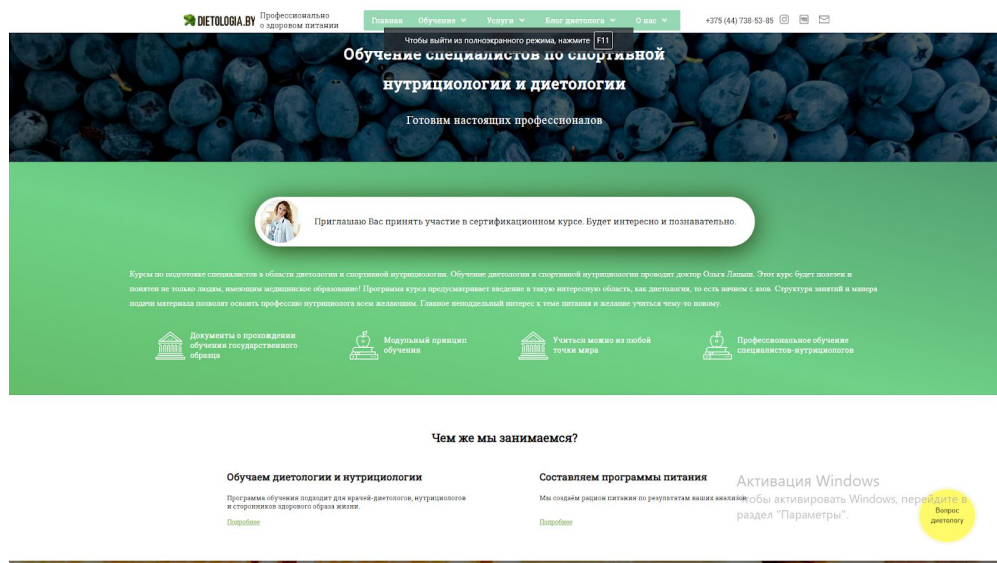


Рисунок 38.. Главная страница

Переходим в раздел Обучение → Семинары

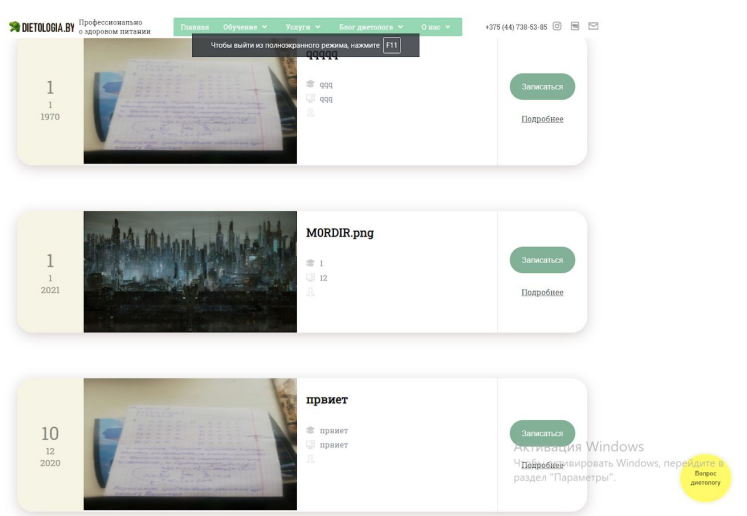


Рисунок 39. Вкладка семинары

```

1 import React, {useRef} from 'react';
2 import { useState, useEffect } from 'react';
3
4 import Seminary from './Seminary.jsx'
5 import { getSeminars } from '../fetch/fetchSeminar'
6
7 export default function SeminaryContainer(props){
8     const [allSeminars, setAllSeminars] = useState([])
9
10     const fetchGETSeminars = () =>{
11         getSeminars().then( data=>setAllSeminars(data))
12     }
13     useEffect(()=>(
14         fetchGETSeminars()
15     ),[])
16     return (
17         <Seminary
18             allSeminars={allSeminars}
19         />
20     )
21 }
22

```

Рисунок 40. Получение всех семинаров с сервера

С помощью запроса **fetchGETSeminars**, получаем данные с сервера (это массив семинаров) и пробрасываем их в компонент **Seminary**.

```

export async function getSeminars(){
    const response = await fetch(urlGETSeminars, {
        method: "GET",
        headers: {
            "Content-Type": "application/json",
        },
    })
    const data = response.json()
    return data
}

```

Рисунок 34 Реализация метода getSeminars


```

{
  props.allSeminars ? props.allSeminars.map((value,index) =>(  

    <div>  

      <p>  

        <label htmlFor="">{value.dateDay}</label>  

        <label htmlFor="">{value.dateMounth}</label>  

        <label htmlFor="">{value.dateYear}</label>  

      </p>  

      <div>  

        {  

          console.log(value.img)  

        }  

        <img src={` ${urlBack}/${value.img}`} alt="" />  

      </div>  

      <div className="discription-seminar">  

        <p>  

          <label htmlFor="">{value.name}</label>  

        </p>  

        <p>  

           <label htmlFor="">{value.authors}</label>  

        </p>  

        <p>  

           <label htmlFor="">{value.format}</label>  

        </p>  

        <p>  

           <label htmlFor="">{value.place}</label>  

        </p>  

      </div>  

      <div className="novigation-seminar">  

        <button onClick={()=>{openModal();setNameSeminar(value.name)}}>Записаться</button>  

        <Link to={{pathname:`seminary/${value._id}`,  

          value: value}}>Подробнее</Link>  

      </div>  

    </div>  

  )) : ''
}

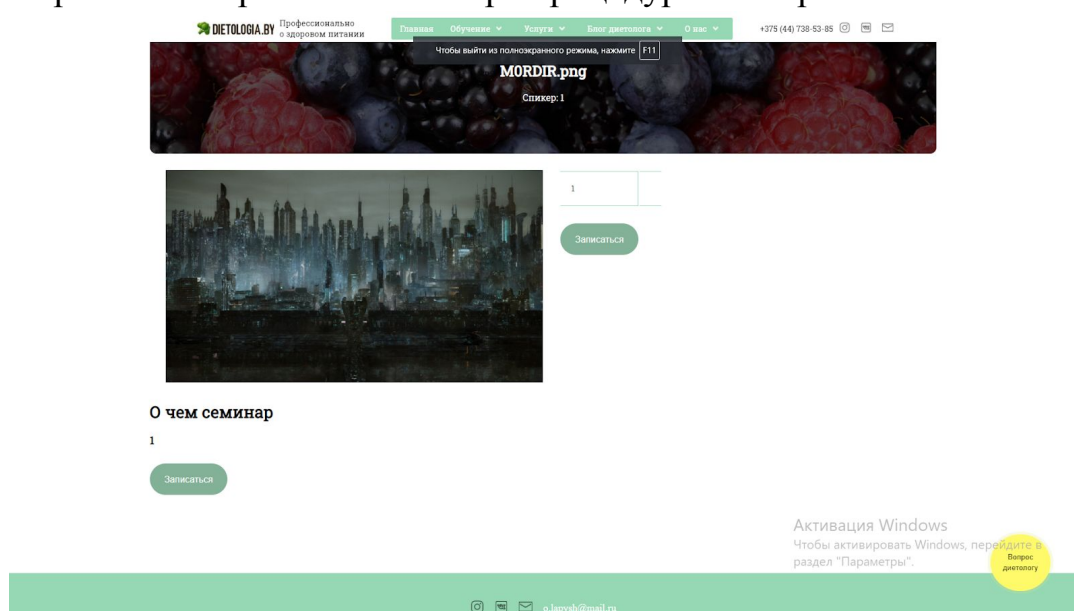
```

Активация Windows
Чтобы активировать Windows, перейдите в раздел "Параметры".

Рисунок 34 Отображение всех семинаров

Отображение выглядит так, нам прилетает массив семинаров мы с помощью **map** проходимся по каждому семинару и рендерим его информацию

При открытии конкретного семинара процедура повторяется



Активация Windows
Чтобы активировать Windows, перейдите в раздел "Параметры".

Рисунок 35 Отображение конкретного семинара

Разница заключается в том, что вместо всех семинаром мы по **Id** просим прислать нам конкретный семинар и всю его информацию отображаем.

С новостями та же история

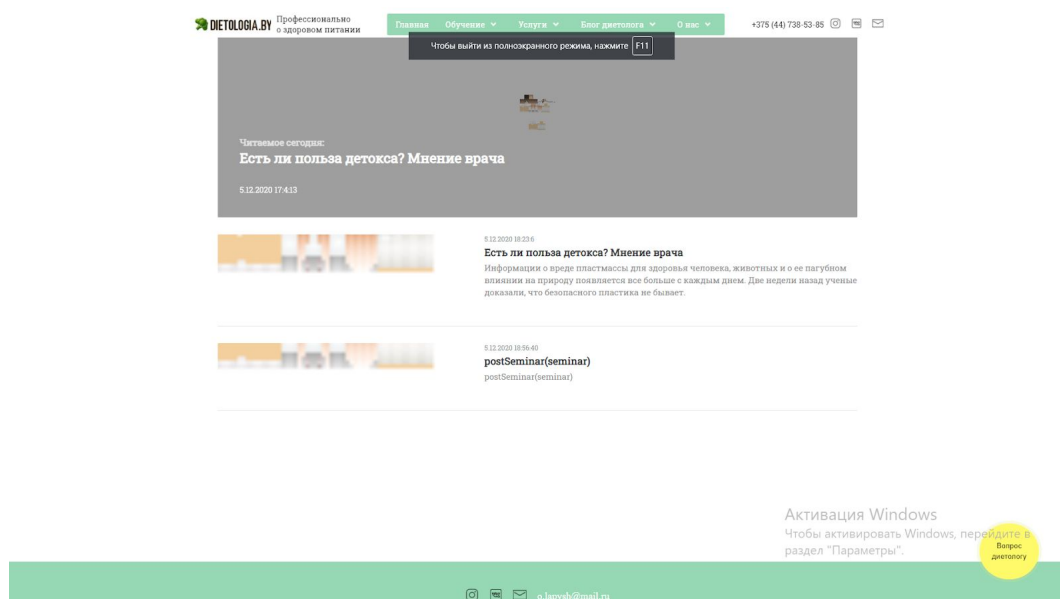


Рисунок 36 Отображение всех новостей

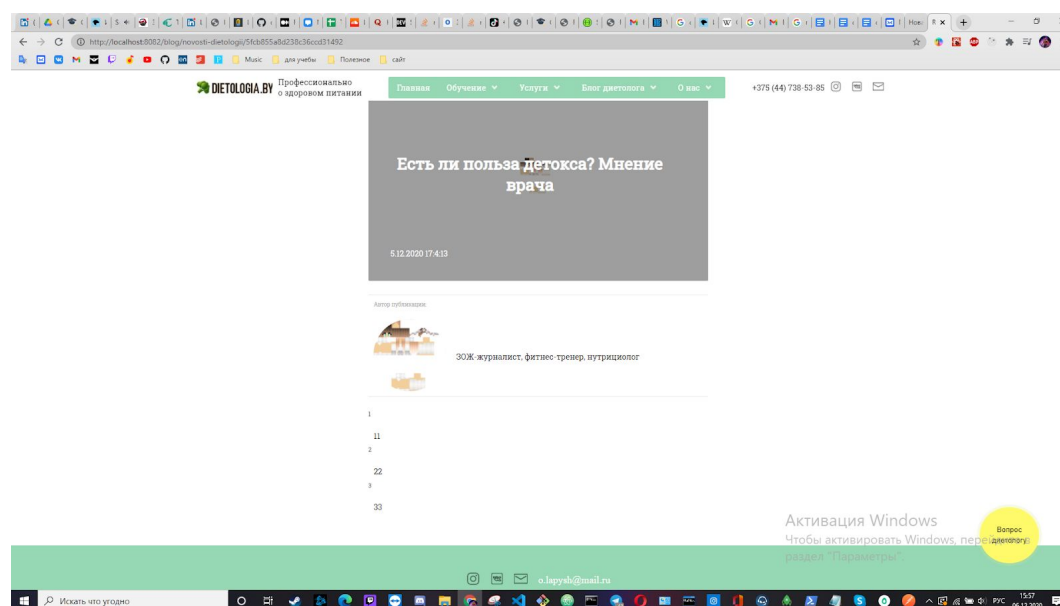


Рисунок 37 Отображение конкретной новости

Заключение

В рамках данной курсовой работы я показал работу как mongodb показывает себя для разработки сайтов. Как можно было увидеть mongodb , базы работала хорошо и даже в какой-то степени взаимодействовала с js. Основные плюсы mongodb – это их понятность и простота в использовании, отсутствие схемы, данная БД основана на коллекциях различных документов, количество полей, содержание и размер этих документов может отличаться, (т.е. различные сущности не должны быть идентичны по структуре), для хранения используемых в данный момент данных используется внутренняя память, что позволяет получать более быстрый доступ, Данные хранятся в виде JSON документов. Минус MongoDB нет положений о хранимых процедурах или функциях, поэтому не получится реализовать какую-либо бизнес-логику на уровне базы данных, что можно сделать в реляционных БД.

Список использованной литературы

1. Базы данных – Что это такое? [Электронный ресурс]. - Электронные данные. - Режим доступа: (<https://hostiq.ua/wiki/database/>);
2. What is a Relational Database? [Электронный ресурс]. - Электронные данные. - Режим доступа: (<https://aws.amazon.com/relational-database/>);
3. Non-relation data and NoSQL [Электронный ресурс]. - Электронные данные. - Режим доступа: (<https://docs.microsoft.com/azure/architecture/data-guide/big-data/non-relational-data>);
4. Visual Studio Code [Электронный ресурс]. - Электронные данные. - Режим доступа: (https://ru.wikipedia.org/wiki/Visual_Studio_Code);
5. HTML5 [Электронный ресурс]. - Электронные данные. - Режим доступа: (<https://ru.wikipedia.org/wiki/HTML5>);
6. CSS [Электронный ресурс]. - Электронные данные. - Режим доступа: (<https://ru.wikipedia.org/wiki/CSS>);
7. JavaScript [Электронный ресурс]. - Электронные данные. - Режим доступа: (<https://ru.wikipedia.org/wiki/JavaScript>);
8. React [Электронный ресурс]. - Электронные данные. - Режим доступа: (<https://ru.reactjs.org/>);
9. Mongodb[Электронный ресурс]. - Электронные данные. - Режим доступа: (<https://www.mongodb.com/>);
- 10.Redux[Электронный ресурс]. - Электронные данные. - Режим доступа: (<https://redux.js.org/>);

Приложение 1. Исходные файлы

1. Ссылка на исходные файлы основного сайта:
<https://github.com/gomelviiv/sports-nutrition-cod>
2. Ссылка на исходные файлы сайта с админ панелью:
<https://github.com/gomelviiv/sports-nutrition-admins-code>
3. Ссылка на исходные файлы сервера сайта:
<https://github.com/gomelviiv/sports-nutrition-back-code>