МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

#### «БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ им. В. Г. ШУХОВА»

**(БГТУ им. В.Г. Шухова)**

Кафедра программного обеспечения вычислительной техники и автоматизированных систем

## Курсовой проект

по дисциплине: Базы данных

тема: «Веб-приложение платформа для онлайн курсов»

Автор работы Буйвало Анастасия Андреевна ПВ-202

(подпись)

Руководитель проекта Панченко Максим Владимирович

(подпись)

Оценка

Белгород 2022 г.

## Введение

#### Задание на разработку

* + - Изучить основы языка программирования TypeScript и JavaScript и их применение в области веб-разработки и работы с БД c применением Prisma;
    - Ознакомиться с фреймворком React и его применением в области разработки веб-приложений;
    - Разработать веб-приложение «Платформа для онлайн курсов» для работы с базой данных. Предусмотреть следующий функционал:
      * Разделение ролей пользователей, разработка механизмов аутентификации;
      * Выгрузка данных из БД по сформированным запросам на выборку;
    - При выполнении проектов, основанных на технологиях, изложенных ниже необходимо использовать систему контроля версий (предпочтительно git). Проект принимается в виде ссылки на репозиторий (GitHub);
    - Язык программирования выбирается произвольно. Разработанное программное обеспечение может быть как нативным, так и выполненным в форм-факторе веб-приложения;
    - В качестве СУБД рекомендуется выбирать PostgreSQL \ MySQL;
    - При разработке необходимо воспользоваться технологией ORM (Object Relation Mapping);
    - Допускается разработка проекта в небольших группах (2-3 человека) по согласованию с преподавателем;
    - Провести тестирование проекта;

## Требования к программному продукту

* + - Язык программирования: TypeScript, JavaScript;
    - Фреймворк: React;
    - Среда разработки: Visual Studio Code;
    - Интерфейс: Веб-приложение;
    - Система контроля версий: git;
    - ORM: Prisma.

1. **Краткое введение в JavaScript**

JavaScript – это мультипарадигменный язык программирования, который обычно применяется в качестве встраиваемого инструмента для программного доступа к различным объектам приложений. С точки зрения веб-разработки, без знаний этой технологии невозможно заниматься созданием современных интерактивных сайтов. Язык JS – это то, что «оживляет» разметку страниц (HTML) и пользовательский функционал (CMS) сайтов. С помощью этого языка реализуется возможность реакции страницы или отдельных ее элементов на действия посетителя. Сегодня JavaScript является базовым языком программирования для браузеров. Он полностью совместим с операционными системами Windows, Linux, Mac OS, а также всеми популярными мобильными платформами.

JavaScript относится к интерпретируемым языкам. Для его запуска нужна специальная программа – интерпретатор. Например, когда JavaScript код выполняется на сервере, интерпретатором чаще всего выступает Node.js, когда на фронтенде — сам браузер. В других местах, таких как встраиваемые устройства или автомобили, могут быть свои собственные интерпретаторы.

Основные архитектурные черты: [динамическая типизация](https://ru.wikipedia.org/wiki/%D0%94%D0%B8%D0%BD%D0%B0%D0%BC%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B0%D1%8F_%D1%82%D0%B8%D0%BF%D0%B8%D0%B7%D0%B0%D1%86%D0%B8%D1%8F), [слабая типизация](https://ru.wikipedia.org/wiki/%D0%A1%D0%BB%D0%B0%D0%B1%D0%B0%D1%8F_%D1%82%D0%B8%D0%BF%D0%B8%D0%B7%D0%B0%D1%86%D0%B8%D1%8F), [автоматическое управление памятью](https://ru.wikipedia.org/wiki/%D0%A1%D0%B1%D0%BE%D1%80%D0%BA%D0%B0_%D0%BC%D1%83%D1%81%D0%BE%D1%80%D0%B0_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)), [прототипное программирование](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D1%82%D0%BE%D1%82%D0%B8%D0%BF%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5" \o "Прототипное программирование), функции как [объекты первого класса](https://ru.wikipedia.org/wiki/%D0%9E%D0%B1%D1%8A%D0%B5%D0%BA%D1%82_%D0%BF%D0%B5%D1%80%D0%B2%D0%BE%D0%B3%D0%BE_%D0%BA%D0%BB%D0%B0%D1%81%D1%81%D0%B0).

С помощью JavaScript специалист может:

* быстро интегрировать в код страницы любые теги;
* определять внешний вид элементов;
* выполнять позиционирование объектов;
* получать пользовательские данные;
* взаимодействовать с сервером (AJAX).

1. **Краткое введение в TypeScript**

Этот язык является надстройкой JavaScript.

Код на TypeScript компилируется в JS и подходит для разработки любых проектов под любые браузеры — тем более что можно выбрать версию JS, в которую будет компилироваться код.

Два главных преимущества TS перед JS:

* Строгая типизация: многие проблемы в JavaScript появляются из-за динамической типизации и в целом странного поведения типов данных:



В TypeScript типизация статическая, что избавляет от множества проблем. Есть числовой тип, строковый, логический и другие. Возможно описывать и свои типы данных

* Расширенная поддержка ООП

И в JS, и в TS есть поддержка объектно-ориентированного программирования: классы, объекты, наследование. Однако TypeScript шагнул чуть дальше и использует больше возможностей ОПП. В том числе, например, интерфейсы. Другой большой плюс — модификаторы доступа. Их в TypeScript три: public, private и protected.

Также есть и другие возможности:

* определение полей в конструкторе;
* преобразование типов;
* абстрактные классы;
* обобщение и так далее.

1. **Краткое введение в React**

React — [JavaScript](https://ru.wikipedia.org/wiki/JavaScript)-[библиотека](https://ru.wikipedia.org/wiki/%D0%91%D0%B8%D0%B1%D0%BB%D0%B8%D0%BE%D1%82%D0%B5%D0%BA%D0%B0_JavaScript) с [открытым исходным кодом](https://ru.wikipedia.org/wiki/%D0%9E%D1%82%D0%BA%D1%80%D1%8B%D1%82%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B5_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D0%B5) для разработки [пользовательских интерфейсов](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%82%D0%B5%D1%80%D1%84%D0%B5%D0%B9%D1%81_%D0%BF%D0%BE%D0%BB%D1%8C%D0%B7%D0%BE%D0%B2%D0%B0%D1%82%D0%B5%D0%BB%D1%8F).

React разрабатывается и поддерживается [Facebook](https://ru.wikipedia.org/wiki/Facebook), [Instagram](https://ru.wikipedia.org/wiki/Instagram" \o "Instagram) и сообществом отдельных разработчиков и корпораций.

React может использоваться для разработки [одностраничных](https://ru.wikipedia.org/wiki/%D0%9E%D0%B4%D0%BD%D0%BE%D1%81%D1%82%D1%80%D0%B0%D0%BD%D0%B8%D1%87%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%B8%D0%BB%D0%BE%D0%B6%D0%B5%D0%BD%D0%B8%D0%B5) и мобильных приложений. Его цель — предоставить высокую скорость разработки, простоту и [масштабируемость](https://ru.wikipedia.org/wiki/%D0%9C%D0%B0%D1%81%D1%88%D1%82%D0%B0%D0%B1%D0%B8%D1%80%D1%83%D0%B5%D0%BC%D0%BE%D1%81%D1%82%D1%8C). В качестве библиотеки для разработки пользовательских интерфейсов React часто используется с другими библиотеками, такими как MobX, [Redux](https://ru.wikipedia.org/wiki/Redux" \o "Redux) и [GraphQL](https://ru.wikipedia.org/wiki/GraphQL" \o "GraphQL).

Эта библиотека действительно может упростить жизнь разработчикам:

— С её помощью можно построить интерфейс из отдельных компонентов, которые легко поддерживать.

— Она добавляет удобный слой абстракции, избавляя от необходимости работать с DOM напрямую.

— React — это уже не новая библиотека, за ней стоит крупная компания Facebook и большое сообщество разработчиков. Поэтому она отлично протестирована, регулярно поддерживается и стабильно обновляется, а переход к новым версиям проходит максимально гладко.

— Благодаря сообществу у React хорошо проработанная документация и большой опыт, накопленный в статьях, курсах и конференциях. Это значительно облегчает не только изучение библиотеки новичками, но и поиск ответов на всевозможные вопросы в процессе работы.

— На Гитхабе можно найти готовые React-компоненты почти на все случаи жизни. А если их нет, но есть нужные независимые библиотеки, то можно поискать интеграцию либо сделать её самостоятельно.

— В React-сообществе со временем сформировались определённые подходы и соглашения по организации проекта, кода, решению частых проблем. Для разработчиков это значит, что нужно тратить меньше времени на обсуждение некоторых договорённостей — можно использовать готовые и уже устоявшиеся.

— Хотя в проекте, скорее всего, будет налажен процесс сборки с помощью webpack, parcel, rollup или другого упаковщика, стоит иметь в виду, что это необязательно для использования React. Работая с этой библиотекой, вы пишете на чистом JS, и никакие диалекты HTML, CSS или JS изучать не нужно. Конечно, с React почти всегда используется JSX, но и это тоже [опционально](http://ru.reactjs.org/docs/react-without-jsx.html).

— React — это проект с открытым исходным кодом. Благодаря этому его можно безопасно использовать даже в коммерческих приложениях.

Однако при использовании React есть особенности, которые важно учитывать:

— React увеличивает размер приложения, которое нужно загрузить пользователям (~40 kB для пакетов React и React-dom).

— Загруженный код нужно выполнить в браузере: для пользователя это означает, что приложение будет запускаться медленнее. Подробнее про это можно почитать в [материале](http://v8.dev/blog/cost-of-javascript-2019) «The cost of JavaScript in 2019» от команды V8 (перевод: «[Цена JavaScript в 2019 году](http://habr.com/ru/company/ruvds/blog/459296/)»).

— Вместе с виртуальным DOM возникают свои накладные расходы: во-первых, по времени выполнения (сравнение виртуальных деревьев происходит не моментально); во-вторых, по памяти, ведь виртуальные деревья нужно где-то хранить и не в единственном экземпляре. С увеличением количества элементов на странице эти расходы растут, что может стать реальной проблемой на мобильных устройствах. Поэтому при изучении React важно уделить внимание способам оптимизации рендеринга приложения – необходимые инструменты для этого есть в самой библиотеке.

— Средний порог входа в React. Чтобы начать разрабатывать на React, программист должен не только изучить саму библиотеку, но и привыкнуть к используемой парадигме. Про это есть хорошая [статья](http://ru.reactjs.org/docs/thinking-in-react.html) «Философия React» на официальном сайте.

Эти недостатки, конечно, не повод совсем отказаться от использования React и подобных ему библиотек при создании проектов. Но о них нужно помнить для того, чтобы использовать этот инструмент осознанно и понимая его ограничения.

Несмотря на то, что виртуальное дерево расходует дополнительные время и память, для большинства приложений это некритично: React остаётся достаточно быстрым и позволяет оптимизировать проблемные места там, где это нужно.

1. **Краткое введение в Prisma**

Prisma представляет собой «открытую ORM нового поколения для Node.js и TypeScript», реализующую «новую парадигму объектно-реляционного отображения».  
  
Поддерживаемые языки программирования:

* JavaScript
* TypeScript
* Go (в разработке)

Поддерживаемые базы данных:

* MySQL
* PostgreSQL
* SQLite
* MSSQL (в разработке)
* MongoDB Connector (в разработке)

Предоставляемые инструменты:

* [Prisma Client](https://www.prisma.io/client): автоматически генерируемый и типобезопасный клиент для БД
* [Prisma Migrate](https://www.prisma.io/migrate): декларативное моделирование данных и настраиваемые миграции
* [Prisma Studio](https://www.prisma.io/studio): современный пользовательский интерфейс для просмотра и редактирования данных
* [Prisma VSCode Extension](https://marketplace.visualstudio.com/items?itemName=Prisma.prisma): расширение для VSCode, обеспечивающее подсветку синтаксиса, автозавершение, быстрые исправления и др.

Prisma Client может быть использован в любом Node.js или TypeScript серверном приложении.

Все начинается с определения [схемы](https://www.prisma.io/docs/concepts/components/prisma-schema). Схема позволяет разработчикам определять модели с помощью интуитивно понятного языка. Она также содержит соединение с БД и определяет генератор.

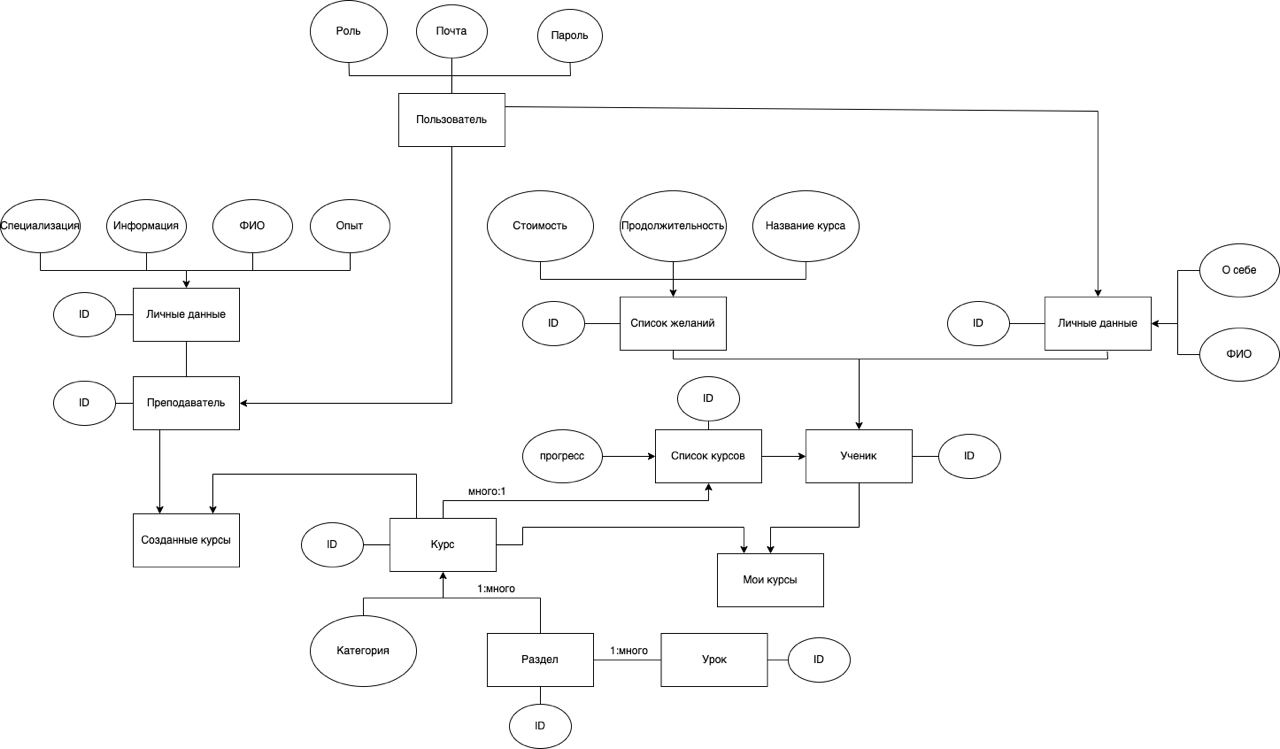
Каждая модель привязана к таблице в БД и является основой для генерируемого Prisma Client интерфейса доступа к данным.

Prisma Client позволяет разработчикам мыслить в категориях объектов. Другими словами, вместо концепции экземпляров модели, в ответ на запрос к БД возвращаются обычные JavaScript-объекты. Кроме того, запросы являются полностью типизированными. Рассмотрим несколько примеров

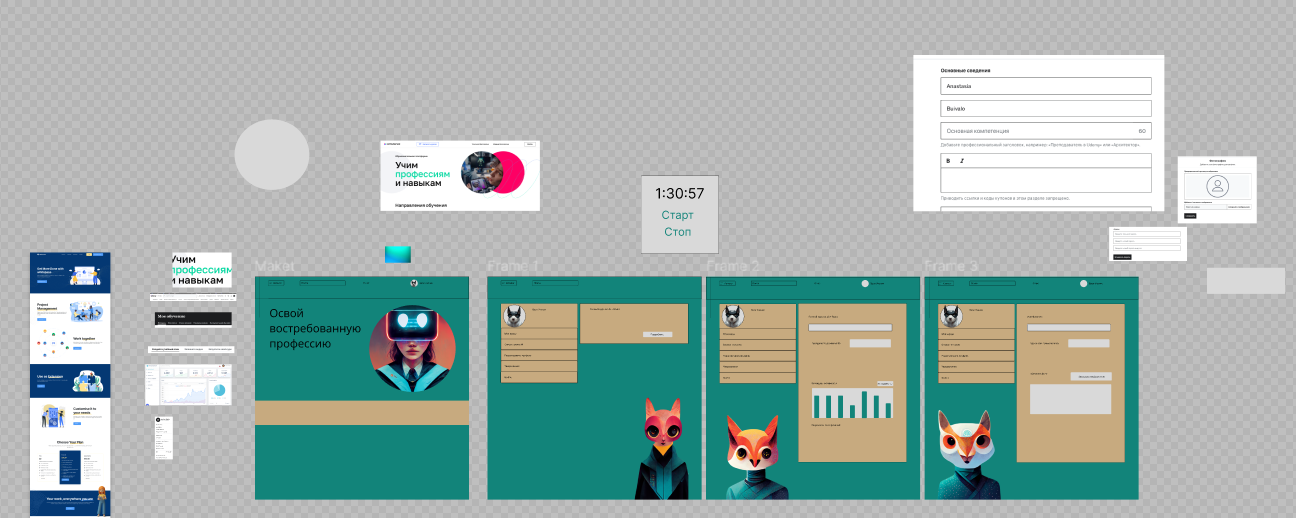
1. **Референсы**

****

1. **ER модель**

****

1. **Дизайн макет в Figma**



Картинки были сгенерированы с помощью нейросети Midjourney

1. **Серверная часть приложения**
   1. **Подключение Prisma**

Выбрала ORM Prisma(первый во всех списках)

Для Prisma существует хорошая пошаговая инструкция, поэтому остается выполнить эти шаги.

Устанавливаем пакет Prisma: npm i -D prisma

Инициализируем проект: npx prisma init

Автоматически создались файлы: prisma/schema.prisma и .env.

В файле .env содержится переменная DATABASE\_URL, значением которой является путь к (адрес) БД.

Сгенерируем модели на основе БД: npx prisma db pull

Создание миграций:

Сначала нужно создать каталог migrations и добавить в него каталог с именем миграции: mkdir -p prisma/migrations/0\_init.

Затем создется файл миграции с расширением prisma migrate diff.  Аргументы:

* --from-empty: предполагается, что модель данных, с которой выполняется миграция, пуста.
* --to-schema-datamodel: текущее состояние базы данных с использованием URL-адреса в datasourceблоке
* --script: вывод SQL-скрипта

Проверяем содержимое файла и помечаем миграцию как примененную

npx prisma migrate resolve --applied 0\_init

* 1. **Запросы**

Я решила написать сначала sql запросы, а потом преобразовать их в запросы prisma.

Пример запроса sql запроса:

getMyCourseSQL: (student\_id:Number) => Prisma.sql`

    select \* from all\_course

    join student\_course on course\_id = all\_course.id

    where course\_id = (select course\_id from student\_course

                                  where student\_id = ${student\_id});

  `,

Запрос prisma:

getMyCourseSQL:(student:any)=>prisma.$queryRaw(SQL\_QUERIES.getMyCourseSQL(student.student\_id)),

Теперь можно использовать запрос из объекта с prisma запросами (приведен участок кода из get запроса):

const data = await prismaQueries[req.params.query](req.query);

* 1. **Использование Express**

Express- самый популярный веб-фреймворк для Node. Он является базовой библиотекой для ряда других популярных [веб-фреймворков Node](https://expressjs.com/en/resources/frameworks.html). Он предоставляет следующие механизмы:

* Написание обработчиков для запросов с различными HTTP-методами в разных URL-адресах (маршрутах).
* Интеграцию с механизмами рендеринга «view», для генерации ответов, вставляя данные в шаблоны.
* Установка общих параметров веб-приложения, такие как порт для подключения, и расположение шаблонов, которые используются для отображения ответа.
* «промежуточное ПО» для дополнительной обработки запроса в любой момент в конвейере обработки запросов.

Использование express:

import express, { Router } from 'express';

export const mainRouter = Router();

mainRouter.use(

  '',

  express.json({ limit: '5kb'})

);

mainRouter.use(express.urlencoded({extended:false}))

Определим обработчик маршрута для для HTTP-запросов GET к '/queries/:query'

mainRouter.get(

  '/queries/:query',

  async (req: any, res: any, next: any) => {

    console.log(req.query);

    //@ts-ignore

    const data = await prismaQueries[req.params.query](req.query);

    try {

      res.status(200).json({

        data

      })

    } catch(e) {

      console.log('Server error', e);

      res.status(500).json({

        message: 'Server Error',

      })

    }

  }

)

В index.ts также подключаем и используем express

const app = express();

Используем уже созданный mainRouter

app.use('/api/main', mainRouter)

Чтобы отследить запущен ли сервер, вывожу host и port

    server.listen(PORT, HOST, () => {

      console.log('API server running on host:', HOST, 'port:', PORT)

    })

* 1. **Бэкапы**

В серверной части приложения происходят бэкапы в два файла, используется fs для асинхронной записи в файл

const backupLecturerCourse= await prisma.$queryRaw(SQL\_QUERIES.backupLecturerCourse());

    const backup ={backupCourses, backupLecturerCourse, backupLecturers, backupStudentCourse, backupStudents, backupUsers};

    fs.writeFile("backap.txt", JSON.stringify(backup, null, "\t"), (err) => {

        if (err)

          console.log(err);

        else {

          console.log("File written successfully\n");

        }

      });

    fs.writeFile("backap.json", JSON.stringify(backup, null, "\t"), (err) => {

        if (err)

          console.log(err);

        else {

          console.log("File2 written successfully\n");

        }

      });

1. **Клиентская часть приложения**
   1. **Стили (Sass)**

При разработке приложения попробовала несколько вариантов задания стилей и остановилась на sass, для этого были созданы файлы .scss для каждого .js файла с компонентами и страницами

.scss файл для карточки курса:

.mini\_card{

    border: 2px solid #000000;

    border-radius: 5px;

    &\_right{

        display: flex;

        justify-content: end;

    }

    a{

        text-decoration: none;

    }

    button{

        width: 180px;

        padding: 15px;

        margin-top: 25px;

        font-size: 18px;

        color: #fff;

        border: 0;

        border-radius: 10px;

        background-image: linear-gradient(257deg, #f59c07, #f57507);

    }

}

.header{

    display: flex;

    justify-content: space-around;

}

* 1. **Использование formik для валидации форм**

Для форм была использована библиотека formik, чтобы не делать валидацию с использованием регулярных выражений для валидации форм, я использовала методы валидации из этой библиотеки.

Для этого нужно создать объект, используя useFormik:

    const formik = useFormik({

        initialValues:{

            email:'',

            password:'',

            terms:false,

        },

        onSubmit: values=>{

            alert('Отправка формы');

            //console.log(JSON.stringify(values, null, 2))

        },

        validationSchema: yup.object({

            email: yup.string()

                        .email('неправильный')

                        .required('обзятельное поле'),

            password: yup.string()

                         .min(8, 'Минимум 8 символов')

                         .required('обзятельное поле'),

        })

    });

* 1. **Маршруты и ленивая загрузка**

const CourseList= lazy(()=>import('../courseList/CourseList'));

Для создания маршрутов использовала пакет react-router-dom.

Структура для переходов пользователя:

Router

--route(exact path = «» component = {})

Последний маршрут будет вести к странице с ошибкой, то есть обрабатывается ошибка при использовании путей, для которых нет компонентов

                <Route path="\*">

                     <Page404/>

                </Route>

* 1. **Вкладки профиля**

Для того, чтобы вкладки профиля каждый раз заново не загружались, я использовала мемоизацию, с массивом зависимостей, который содержит идентификатор текущего экрана

    const CurrentScreen = useMemo(()=>{

        switch (idCurrentScreen){

            case 1:

                console.log('1');

                return <Courses/>;

            case 2:

                console.log('2');

                return <WishList/>;

            case 3:

                console.log('3');

                return<EditProfile/>;

            case 4:

                console.log('4');

                return <Notification/>;

        }

    }, [idCurrentScreen])

* 1. **Запросы**

Для того чтобы была возможность отправлять запросы я создала объект api с запросами:

  getCourseSQL:function(data:any){

    return sauce.get<RowData[]>('/main/queries/getCourseSQL', data);

  },

Пример использования объекта api:

const res = await api['getCourseSQL']({start:offset-3,  end:offset});

Прокидываем параметры запроса через все уровни, до sql запроса

* 1. **Разделение ролей**

При регистрации пользователем указывается роль, в зависимости от этой роли данные пользователя вносятся либо в таблицу студентов, либо в таблицу преподавателей, также данные заносятся в таблицу пользователей.

Изображение выглядит как текст

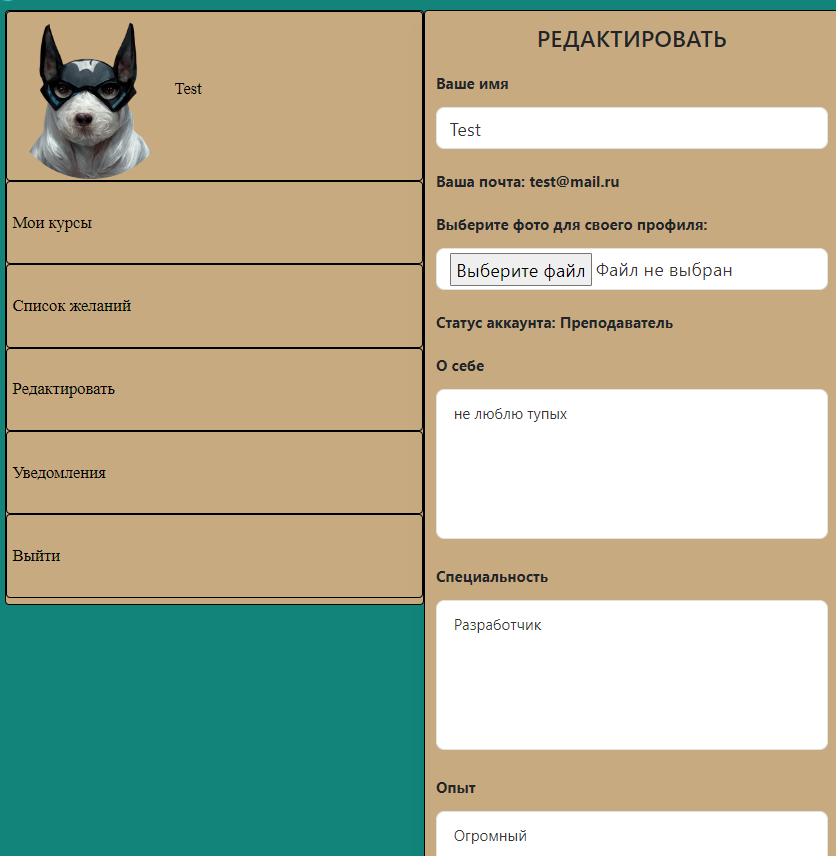
Автоматически созданное описание

* 1. **Редактирование профиля**

Преподаватель и студент могут редактировать свои личные кабинеты, по доступным роли параметрам для этого реализованы update на стороне сервера

Изображение выглядит как текст

Автоматически созданное описание



1. **Подключение клиентской и серверной части**

Чтобы клиентская и серверная часть могли взаимодействовать, нужно подключить их по IP. Для этого в клиентской части приложения создаем файл config.ts:

export const CONFIG = {

    BASE\_URL: 'http://192.168.1.175:8070/api'

  } as const;

В файле api:

const sauce = create({

  baseURL: CONFIG.BASE\_URL,

  headers: { Accept: "application/json" },

});

Теперь можно добавить sauce в объект api, который содержит запросы

В серверной части приложения так же подключаемся по IP

const server = http.createServer(app);

const HOST = '192.168.1.175';

const PORT = 8070;

Теперь можно запустить обе части через npm start

1. **Возможности приложения**

Созданное веб-приложение обладает следующим функционалом:

* Регистрация и авторизация преподавателей и студентов
* Создание преподавателем курса
* Редактирование профиля
* Просмотр каталога курсов
* Просмотров своих курсов в личном кабинете
* Автоматическая выгрузка данных

1. **Вложения**

Ссылка на репозиторий проекта:

<https://github.com/AnastasiaBuivalo/course_platform>

<https://github.com/AnastasiaBuivalo/server_courses>

Ссылка на видео работы:

https://youtu.be/ynHmta-4Mdc

1. **Вывод**

## В ходе выполнения данной курсовой работы было разработано веб-приложение «Платформа для онлайн курсов» для работы с базой данных.

При разработке БД был предусмотрен следующий функционал:

* Разделение ролей пользователей, механизм аутентификации пользователей;
* Выгрузка данных из БД по сформированным запросам на выборку;

Так же был приобретен опыт работы с системой контроля версий git