Отзыв

руководителя на выпускную квалификационную работу студента гр.ИСТТ(дот)-1-15
Венгер Станислав Константиновича
на тему: Автоматизация бизнес-процессов кафедры ИСТТ

Выпускная квалификационная работа Венгер С.К. посвящена разработке веб-приложения для кафедры «ИСТТ» по аналогии с существующим «AVN», используя современные Фреймворки. Данное приложение позволит быстрее обмениваться информацией преподавателям и студентам кафедры «ИСТТ».

Разработанное приложение выполняет следующие функции:

- Авторизация пользователей (преподавателей и студентов);
- Добавление учебного материала;
- Редактирование данных;
- Добавление новых объявлений;
- Личный кабинет;
- Чат.

В ходе выпускной квалификационной работы Венгер С.К. хорошо разобрался с принципами организации онлайн обучения студентов, показал профессиональные знания Node.js, база данных MongoDB, React.js, HTML5, CSS3, JavaScript, проявил себя как инициативный, самостоятельный и ответственный студент.

Выпускная квалификационная работа заслуживает оценки “Отлично”, а его автор Венгер С.К. - присвоения академической степени бакалавра по направлению “ИСТ”.

Преп. ИЭТ КГТУ им. И. Раззакова  Сариев Б.И.

РЕЦЕНЗИЯ

на выпускную квалификационную работу студента:

Венгер Станислав Константиновича

группы: ИСТТдот-1-15

направления: «Информационные системы и технологии»

на тему: «Автоматизация бизнес-процессов кафедры ИСТТ»

Выпускная квалификационная работа посвящена разработке веб-приложения для кафедры «ИСТТ» по аналогии с существующим «AVN», используя современные Фреймворки. Данное приложение позволит быстрее обмениваться информацией преподавателям и студентам кафедры «ИСТТ». Для реализации используется Node.js, база данных MongoDB и клиентская часть реализована при помощи библиотеки React.js. Пояснительная записка выполнена согласно требованиям к выпускным квалификационным работам направления «ИСТ» КГТУ им. И.Раззакова. Она состоит из введения, 4 глав, обозначений и сокращений, заключения и списка литературы. Графическая часть представлена слайдами, которые объективно отражают суть проделанной работы.

Каждая из глав изложена достаточно полно.

В аналитической части грамотно рассмотрено состояние предметной области, проведен анализ систем прототипов. Достаточно подробно сформулирована цель и задача разработки системы и выделены основные требования к проектируемой системе обработки данных.

В главах «Моделирование предметной области и работа системы» хорошо выполнен структурно-функциональный анализ объекта управления и решаемой задачи, разработана структурно-функциональная диаграмма. Подробно описаны и обоснованы решения задачи, представленной в аналитической части. Хорошо представлены результаты моделирования системы, построенные с использованием CASE – средства BPwin. Детально расписано проектирование базы данных с помощью CASE- средства Erwin. Продемонстрировано хорошее владение CASE – средствами. Достаточно подробно описаны общие положения, отражающие стандарты, а также требования к аппаратным и программным ресурсам для успешной эксплуатации программного средства. Описана работа приложения.

Веб-приложение имеет удобный и современный интерфейс.

Замечаний по проектированию нет.

Представленная на рецензирование выпускная квалификационная работа Венгер Станислава заслуживает отличной оценки.

Место работы и должность рецензента:

Ф.И.О. рецензента:

Подпись _____

«25» июня 2020г

КЫРГЫЗ МАМЛЕКЕТТИК ТЕХНИКАЛЫК УНИВЕРСИТЕТИ

КЫРГЫЗ РЕСПУБЛИКАСЫНЫН БИЛИМ БЕРҮҮ ЖАНА ИЛИМ МИНИСТРЛИГИ
И.РАЗЗАКОВ атындагы КЫРГЫЗ МАМЛЕКЕТТИК ТЕХНИКАЛЫК УНИВЕРСИТЕТИ

ИЭТ КГТУ

ЭЛЕКТРОНИКА ЖАНА ТЕЛЕКОММУНИКАЦИЯЛАР ИНСТИТУТУ
«ТЕЛЕКОММУНИКАЦИЯДАГЫ МААЛЫМАТТАР СИСТЕМАЛАРЫ ЖАНА ТЕХНОЛОГИЯЛАРЫ» кафедрасы

«Маалымат системалары жана технологиялары»
даярдоо багытынын
«Телекоммуникациядагы маалымат системалары жана технологиялары» профили боюнча

БҮТҮРҮҮЧҮ КВАЛИФИКАЦИЯЛЫК ИШ

ТМСжТ кафедрасынын бизнес-процесстерин автоматташтыруу

Аткарган	ИСТТ(дот)-1-15 тайпасынын студентти Венгер Станислав Константинович	(кол белгиси)
Иштин жетекчиси	Сариев Бактыбек Имангазыевич	(кол белгиси)
"Жактоого уруксат берилди" Кафедра башчысы	Ф.-м.и.к., доц. Дүйшөков К.Д.	(кол белгиси)
Сын баяндоочу	Бөлөтов Р.Б	(кол белгиси)

Бишкек 2020

КЫРГЫЗ МАМЛЕКЕТТИК
ТЕХНИКАЛЫК УНИВЕРСИТЕТИ

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИКЫРГЫЗСКОЙ РЕСПУБЛИКИ

КЫРГЫЗСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ им. И.РАЗЗАКОВА

ИЭТ КГТУ

ИНСТИТУТ ЭЛЕКТРОНИКИ И ТЕЛЕКОММУНИКАЦИЙ

Кафедра «ИНФОРМАЦИОННЫЕ СИСТЕМЫ И ТЕХНОЛОГИИ В ТЕЛЕКОММУНИКАЦИЯХ»

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА
по направлению «Информационные системы и технологии», профилю подготовки «Информационные системы и технологии в телекоммуникациях»

Автоматизация бизнес-процессов кафедры ИСТТ

Выполнил(а)	ст. группы ИСТТ(дот)-1-15 Венгер Станислав Константинович	(подпись)
Руководитель ВКР	Сариев Бактыбек Имангазыевич	(подпись)
"Допущен к защите" **Заведующий кафедрой**	к.ф-м.н., доц. Дүйшөков К.Д.	(подпись)
Рецензент	Бакытов Р.Б.	(подпись)

Бишкек 2020

РАЗЗАКОВ АТЫНДАГЫ КЫРГЫЗ МАМЛЕКЕТТИК ТЕХНИКАЛЫК УНИВЕРСИТЕТИ
КЫРГЫЗСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
им. И.РАЗЗАКОВА

ЭЛЕКТРОНИКА ЖАНА ТЕЛЕКОММУНИКАЦИЯЛАР ИНСТИТУТУ
ИНСТИТУТ ЭЛЕКТРОНИКИ И ТЕЛЕКОММУНИКАЦИЙ

«ТЕЛЕКОММУНИКАЦИЯДАГЫ МААЛЫМАТТАР СИСТЕМАЛАРЫ ЖАНА ТЕХНОЛОГИЯЛАРЫ» кафедрасы
Кафедра «ИНФОРМАЦИОННЫЕ СИСТЕМЫ И ТЕХНОЛОГИИ В ТЕЛЕКОММУНИКАЦИЯХ»

«БЕКИТЕМ»
«УТВЕРЖДАЮ»

Минбардын (кафедранын) башчысы
Зав.кафедрой

«11» 02 2020г.

ИСТТ(дот)-1-15 тайпасынын студентти Венгер Станислав Константинович "ТМСжТ кафедрасынын бизнес-процесстерин автоматташтыруу" темасы боюнча бүтүрүүчү квалификациялык ишке.

ТАПШЫРМА
ЗАДАНИЕ

на выпускную квалификационную работу студенту группы ИСТТ(дот)-1-15 Венгер Станислав Константинович по теме «Автоматизация бизнес-процессов кафедры ИСТТ».

_____ ж. _____ "_____" № факультеттин буйругу менен бекитилген.

утверждена приказом по факультету №38/27 от "13" 02 2020г.

2. Студенттин жумушту тапшыруу мөөнөтү.

Срок сдачи студентом законченной работы 13.06.2020

3. Квалификациялык жумушка карата алгачкы маалымат

Исходные данные к работе: курсовая работа по дисциплине "Управление данными", документация магазина.

Катар №	Эсептеп түшүндүрмө каттын мазмуну (иштетүүгө тийиштүү маселелердин тизмеси) Содержание расчетно-пояснительной	Көлөмү % менен Объем в %	Аткаруу мөөнөтү Срок выполнения

Айрым болүктөрү боюнча кеңеш берүүчүлөр
(жетекчисинен тышкары)
Консультация по отдельным разделам (помимо руководителя)

Катар № № п/п	Бөлүктөрү (аталышы) Раздел (наименование)	Кеңеш берүүчүлөрдүн аты, ата-теги жана кол белгиси Ф.И.О. консультанта и подпись
1.	Экономическая часть	Ст.преп. Каримова Г.Т.
2.	Н.контроль	К.т.н., доцент Сариев Бактыбек Имангазыевич

Тапшырма берген күнү
Дата выдачи задания 13.02.2020г.

Жетекчи
Руководитель к.т.н. доц. Сариев Б.И.

Аты, ата-теги, окумуштуу даражасы, наамы, кол белгиси
Ф.И.О., ученая степень, звание _____ подпись

Эскертүү: Бул тапшырма бүткөн жумушка тиркелет жана мамлекеттик ымтыканга көрсөтүлөт.

Примечание: Это задание прилагается к законченной работе и предъявляется на ГАК.

АНТИПЛАГИАТ
ТВОРИТЕ СОБСТВЕННЫМ УМОМ

Кыргызский государственный технический университет им. И. Раззакова

СПРАВКА
о результатах проверки текстового документа на наличие заимствований

Проверка выполнена в системе Антиплагиат.ВУЗ

Автор работы	**Венгер Станислав Константинович**
Подразделение	КГТУ им.И.Раззакова, ИЭТ, кафедра ИСТ
Тип работы	Выпускная квалификационная работа
Название работы	Автоматизация бизнес-процессов кафедры ИСТ
Название файла	Глава 2 и 3 для антиплаг_Венгер Станислав Константинович.docx
Процент заимствования	**2.65 %**
Процент самоцитирования	**0.00 %**
Процент цитирования	**0.41 %**
Процент оригинальности	**96.94 %**
Дата проверки	**13:46:11 19 июня 2020г.**
Модули поиска	Модуль поиска ИПС "Адилет"; Модуль выделения библиографических записей; Сводная коллекция ЭБС; Модуль поиска "Интернет Плюс"; Коллекция РГБ; Цитирование; Модуль поиска переводных заимствований; Модуль поиска переводных заимствований по elibrary (EnRu); Модуль поиска переводных заимствований по elibrary (KyRu); Модуль поиска переводных заимствований по интернет (EnRu); Модуль поиска переводных заимствований по интернет (KyRu); Коллекция eLIBRARY.RU; Коллекция ГАРАНТ; Модуль поиска "КГТУ"; Коллекция Медицина; Диссертации и авторефераты НБЕ; Модуль поиска перефразирований eLIBRARY.RU; Модуль поиска перефразирований Интернет; Коллекция Патенты; Модуль поиска общеупотребительных выражений; Кольцо вузов
Работу проверил	Каримова Г. Т.
	ФИО проверяющего
Дата подписи	
	Подпись проверяющего

Чтобы убедиться в подлинности справки, используйте QR-код, который содержит ссылку на отчет.

Ответ на вопрос, является ли обнаруженное заимствование корректным, система оставляет на усмотрение проверяющего. Предоставленная информация не подлежит использованию в коммерческих целях.

Содержание

Введение

В настоящее время скорость роста в нашем мире количество Web – ресурсов увеличивается. На данное время практически каждый компьютер или мобильный телефон имеет доступ в Интернет. Один из самых популярных сервисов сети Интернет является «Всемирная паутина».

Изначально сервис использовался только для предоставления статичной информации. Но в настоящее время этот сервис служит платформой для веб – приложений. Если раньше, перед использованием каких-либо приложений на компьютере, необходимо было установить, то на данный момент любые веб – приложение можно запускать с помощью браузера, достаточно только указать адрес приложение. С развитием Интернета специалистам в области веб – разработке необходимо уметь разрабатывать такие веб – приложения.

Для того чтобы создать веб – приложение разработчикам недостаточно только знать какой-либо язык программирования, но еще должен знать основные стандарты Интернета (HTTP/HTTPS, HTML5, CSS3, JavaScript, URL).

Цель данного проекта – разработать веб-приложение для кафедры «ИСТТ» по аналогии с существующим «AVN», используя современные Фреймворки. Данное приложение позволит обмениваться информацией быстрее преподавателям и студентам кафедры «ИСТТ».

Задачи:

1. Анализ и составление технического задания.
2. Разработка удобного интерфейса для приложения.
3. Разработка веб - приложения используя современные библиотеки.

Общим назначением данного приложение, станет автоматизация процесса работы преподавателей и студентов кафедры «ИСТТ». Для работы приложения будет создана серверная часть при помощи Node.js, база данных MongoDB и клиентская часть будет реализована при помощи библиотеки React.js.

Глава 1. Описание предметной области

1.1 Описание учебного портала для кафедры

Необходимость создания портала для кафедры «ИСТТ» возникла с тем, что портал, который предназначен для всего университета «AVN» на данный момент, устарел и является недостаточно функциональным.

Целью работы стало создание портала для кафедры «ИСТТ». Портал должен содержать объявления и учебные материалы (лекции, лабораторные работы, задачи для экзаменов) которые будут загружать преподаватели для студентов. Можно выделить следующую информацию, которая должна быть в приложении:

- ✓ Платформа готова к работе сразу после регистрации. Поддержка всех видов учебных материалов;
- ✓ Об электронных образовательных ресурсах, доступ к которым обеспечивается обучающимся;
- ✓ Личный кабинет, в котором будет присутствовать чат для обмена сообщениями студентов с преподавателями;
- ✓ Управление пользователями;

Организация внутреннего портала, позволит решить многие задачи автоматизации работы кафедры. Внедрение современных технологий позволит сократить многие виды работ. В первую очередь на портале следует уделить внимание:

- ✓ Авторизации;
- ✓ Доске объявлений;
- ✓ Каталогу учебного материала;
- ✓ Личному кабинету;

Не вся информация, которая содержится на страницах портала, является общедоступной, некоторые разделы и функции будут иметь ограничения в доступе.

1.2 Преимущества и недостатки веб-приложений

Веб-приложения имеют свои положительные и отрицательные стороны.

Плюсы:

- ✓ Простой доступ к приложению.
- ✓ Простая установка – в отличие от обычных приложений веб-приложения после завершения разработки не требуют установки на компьютер пользователя.

- ✓ Автоматическое обновление. Если в приложение что то поменялось или обновилась информация, то эти все изменения будут доступны пользователям без скачивания каких-то дополнительных файлов.
- ✓ Переход по разделам приложения без перезагрузки страницы.
- ✓ Высокий уровень развития и надежности сетевых соединений и веб – технологий.
- ✓ Для работы требуется минимальная аппаратная платформа.
- ✓ Адаптивный дизайн – можно использовать на всех устройствах, где есть доступ в интернет.

Минусы:

- ✓ Сбой сервера может привести к тому, что все данные хранящиеся на сервере не будут доступны.
- ✓ Ограничение доступа в интернет в некоторых населенных пунктах.
- ✓ Открытость доступа к приложениям. В связи в открытым доступом к веб-приложению практически для всех поддерживать безопасность работы пользователей с приложением.
- ✓ Несогласованные подходы к оформлению и способу взаимодействия с пользователями. Веб – сеть предоставляет дизайнерам и разработчикам значительную гибкость в разработке, получающиеся в результате этого разнообразие и несогласованность пользовательских интерфейсов и способов взаимодействия с приложением часто создают проблемы для пользователей. Это связано с тем что пользователи сталкиваются с большим разнообразием визуальных интерфейсов и способов взаимодействия.

1.3 Анализ и составление технического задания

Важной частью работы веб – приложения представляет собой составление технического задания. Это делает нашу работу проще и лучше. В ней мы определяем условия создания веб – приложения, его отладка, его темы, а также его заполнение.

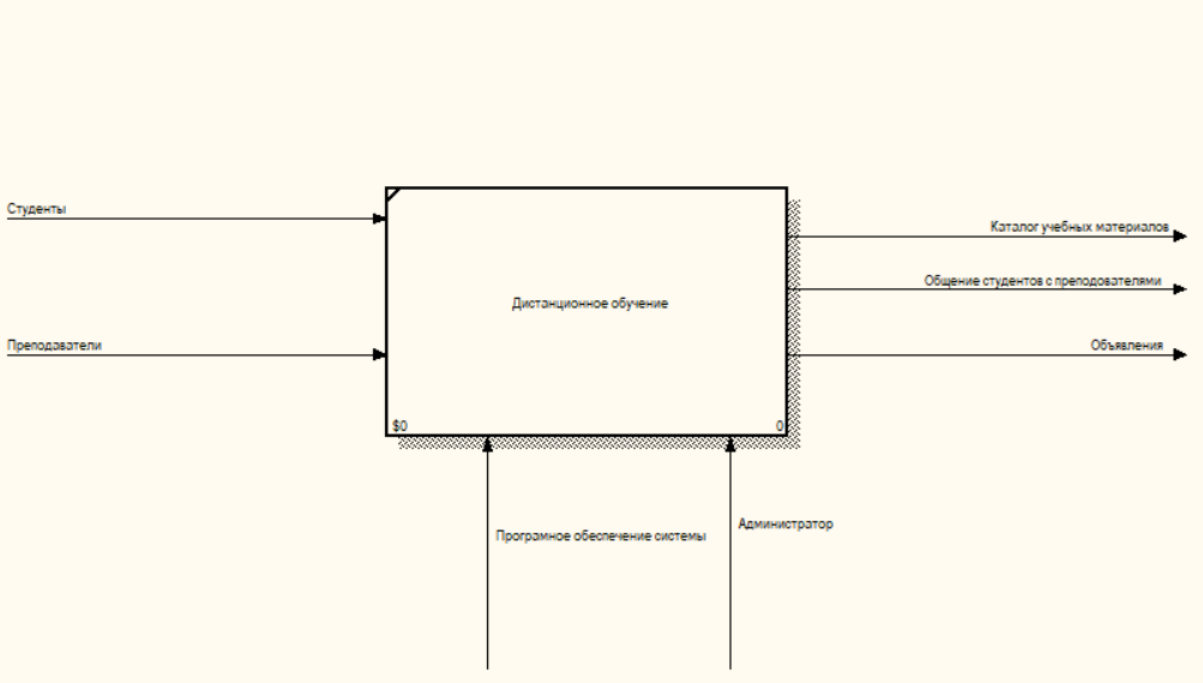

Рисунок 1 - Контекстная диаграмма для дистанционного обучения (IDEF0)

Приложение будет делится на следующие разделы:

- ✓ Авторизация
  - Окно входа в приложение будет главной страницей;
    - Для входа в приложение нужно будет ввести email и пароль
  - Возможность регистрации новых пользователей;
    - Для регистрации необходимо будет заполнить следующие поля:
      - Email (должен быть уникальным)
      - Имя
      - Фамилия
      - Отчество
      - Пароль (должен быть не меньше 8 символов)
      - Повторить пароль (пароли должны совпадать)
      - Права доступа по умолчанию студент

После авторизации пользователя переправит на основную страницу, на которой будет доступно:

- ✓ Верхняя часть приложения (шапка)
  - Название приложения «Информационный портал кафедры ИСТТ»
  - Кнопка выходы

- ✓ Меню навигации по разделам приложения
  - Доска объявления
    - Блоки с объявлением
      - Имя пользователя который создал объявление;
      - Название объявления;
      - Основной текст;
      - Файл (если он был прикреплен при создании объявления);
      - Дата создания объявления;
    - Кнопка добавления новых объявлений (доступна только администратору);
    - Форма для создания нового объявления;
  - Каталог учебных материалов
    - Список преподавателей. Блок будет является как категории для выбора учебного материала;
    - При выборе определенного преподавателя справа от списка будет отображаться учебных материал, который загрузил выбранный преподаватель;
    - Кнопка добавления материала (доступна только преподавателям и администраторам);
    - Форма для добавление материала;
  - Личный кабинет
    - Должно отображать имя, фамилия и отчество пользователя
    - Кнопка для редактирования персональных данных
    - Чат
    - Мои дисциплины (раздел доступен только преподавателям и администратору)
      - Кнопка добавления предмета (доступно администратору);
      - Кнопка назначения преподавателя к определенному предмету (доступно администратору);
      - Форма для добавления предмета;
      - Форма для назначения предмета к преподавателю;
    - Список пользователей (доступно администратору)

- Список всех пользователей с возможностью редактирования прав доступа;
- Кнопка для добавления новых пользователей;
- Форма для добавления новых пользователей. С заполнением следующих полей:
    - Email
    - Имя
    - Фамилия
    - Отчество
    - Пароль
    - Повторите пароль
    - Права доступа (студент, преподаватель, администратор)

Приложение предназначено для преподавателей, студентов и других заинтересованных пользователей, поэтому у приложения должен быть ненавязчивый дизайн, просто и понятный итерфейс для использования.

1.4 Технологические требования

1. Приложение должно быть разработано под любое разрешение экрана;
2. Кроссбраузерность - приложение должно одинаково работать и отображаться в различных браузерах (Internet Explorer, Chrome, Firefox, Safari, Opera, Edge);
3. Кроссплатформенность - это способность приложения работать с более чем одной операционной системой. Приложение должно одинаково работать на таких операционных системах как Windows, Linux, MacOS, Android, IOS
4. Использование строгого стиля;
5. Базовая цветовая гамма – синий (`#0A5AF9`), светло-серый (`#F7F7F7`), черный (`#2A2A2A`)

1.5 Выводы по первой главе

Первая глава в основном состоит только из теоретической информации. Была проанализирована работа веб – приложений, тут я перечислил все преимущества и недостатки, а так же структуру веб – приложений. Было составлено техническое задание для разработки информационного портала кафедры «ИСТТ».

Глава 2. Разработка дизайна и клиентской части

2.1 Разработка дизайна

После составления технического задания, работа передается дизайнеру, который по техническому заданию рисует приложение используя различные программы. Одна из самых востребованных на данное время является Figma. Figma - это облачный инструмент проектирования, похожий на Sketch по функциональным возможностям, но с большими отличиями, которые делают Figma лучше для совместной работы в команде.

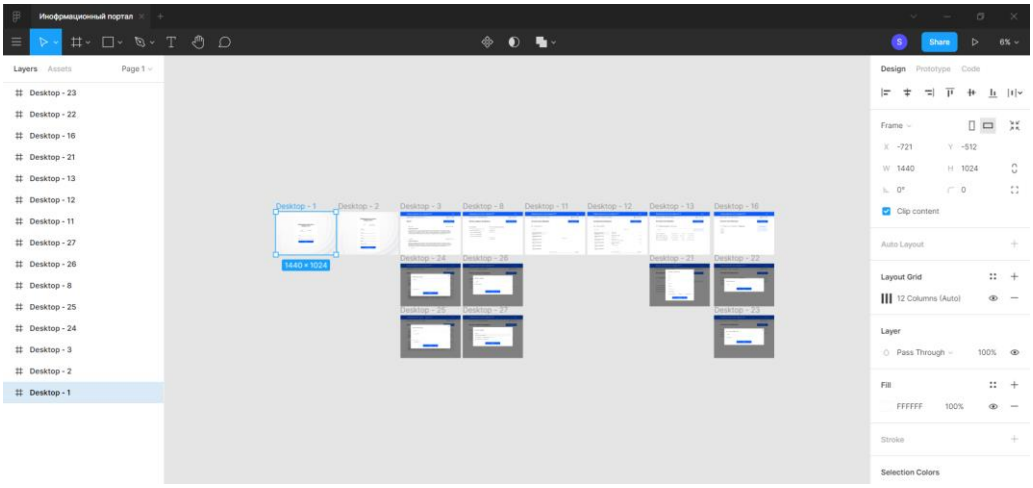

Рисунок 2 – Разработка дизайна в программе Figma

Figma работает на любой операционной системе, в которой можно запустить веб-браузер. Это единственный инструмент дизайна такого типа, который позволяет делиться, открывать и редактировать файлы Figma через браузер. Во многих компаниях дизайнеры используют MacOS, а разработчики используют ПК с Windows или Linux. Figma помогает объединить эти группы и ускорить работу по разработке ресурсов. Все измененные или созданные элементы становятся доступны сразу всем, у кого есть доступ к данному документу.

При разработке дизайна информационного портала были использованы современные принципы разработки UX/UI-дизайна. Были использованы следующие принципы UX:

1. Принцип KISS (Keep it short and simple) – что означает простоту и понятность интерфейса, когда задачи пользователя решаются максимально быстро и очевидно. С таким принципом мы исключаем сложность действий, которые заставляют пользователя долго думать при достижении поставленной задачи.

2. Отсутствие очевидного который помогает пользователю сосредоточиться лишь на необходимых вещах.
3. Проверенные методики. Данный принцип заключается в том, что дизайнер использовал только проверенные элементы интерфейса.
4. Интуитивность и привычность. Пользователю намного проще использовать тот интерфейс, в котором присутствуют привычные и интуитивные понятные элементы, которые позволяют быстро и легко найти нужную информацию.
5. Минимализм информации. Данный принцип направлен на сокращение информационных блоков, так как пользователи не любят много читать. Необходимо отображать информации максимально кратко и понятно.
6. Группировка. Этот принцип позволяет логически сгруппировать блоки так, что пользователь сможет легко сориентироваться при выполнении поставленной цели.
7. Все полезное на виду. В интерфейсе элементы отображаются таким образом, что пользователь может без труда найти необходимую функцию быстро и легко.

Были также использованы принципы UI:

1. Цветовое решение. При разработке дизайна необходимо учитывать целевую аудиторию, поэтому были выбраны нейтральные цвета синий (#0A5AF9), светло-серый (#F7F7F7), черный (#2A2A2A). Данная цветовая гамма особенно подходит для образовательных и информационных проектов.
2. Простота форм элементов. Для разработки информационного портала были выбраны простые формы элементов и иллюстраций, так как данное приложение не носит развлекательный характер.

2.2 Разработка клиентской части React.js

Затем готовый дизайн передают разработчикам. Процесс разработки делится на несколько этапов.

Подготовительный этап. Создается основной раздел, который содержит в себе серверную часть с базой данных и клиентская часть. Для того что чтобы создать само приложение используя React.js необходимо глобально установить Node.js в который встроен npm.

Npm – это менеджер пакетов. Он используется для установки различных пакетов из облачного сервера.

После того как установлен node.js, разработчик приступает к созданию самого React приложения. Для этого необходимо открыть терминал и ввести команду для установки create-react-app <название проекта>, после завершения установки будут скачаны основные файлы и папки. При разработки используются различные программы, в данном случае WebStorm. Следующий шаг, это запуск приложения на локальном хосте по умолчанию localhost::3000, для запуска используется команда npm start после этого приложение откроется в браузере со стандартной разметкой от реакта.
Основные файлы и папки React приложения:

1. node_modules - директория с зависимостями. Вы скорее всего никогда не будете в ней ничего делать вручную, это все работает с помощью npm
2. public - директория, содержащая "публичные файлы", такие как изображения, аудио/видео файлы, иконки, и т.п., то есть файлы, которые должны отдаваться сервером как есть.
3. favicon.ico - иконка сайта, отображаемая в заголовке браузера
4. index.html - "точка входа" в приложение - файл, который будет загружаться главным и подключать весь JS к себе. Вы можете редактировать его, чтобы изменять <title>, добавлять что-то туда.
5. manifest.json - файл-манифест, содержит информацию для отображения, если добавить сайт в закладки (на разных устройствах по-разному).
6. src - директория с кодом вашего приложения. Вы будете вносить изменения в основном внутри этой директории.
7. .gitignore - сразу содержит некоторые предопределенные файлы для игнорирования, которые не подлежат версионному контролю.
8. package.json - файл, содержащий список зависимостей, а также некоторые другие настройки, например, список скриптов для запуска (npm start в их числе)
9. package-lock.json - файл, который фиксирует список зависимостей. Подлежит версионному контролю, содержит список абсолютных версий зависимостей, которые были установлены вами в момент npm install. Если файл существовал на момент запуска установки зависимостей, будут установлены точные версии, указанные в этом файле.
10. README.md - файл-README. Можете изменять, чтобы описывать какие-то особенности проекта. Если вы поместите проект на GitHub или BitBucket, данный файл будет отображаться на главной странице вашего проекта.

Чтобы создавать приложения необходимо еще установить множество дополнительных библиотек, которые будут прописаны в файл package.json такие как:

1. React-router-dom
2. React-redux
3. Redux
4. Redux-thunk
5. Axios
6. React-notifications
7. Material-UI

2.3 Создание репозитория GitLab

Второй этап. Создание репозитория для хранения проекта на удаленном сервере во избежание потери всех файлом, если что-то случится с жестким диском или компьютером в целом. Для данного проекта был использован GitLab, где был создан основной репозиторий и ветки (master, portal). Вся работа введется на рабочей ветке portal после завершения определенной части работы создается pull-request в основную ветку master. Удаленный репозиторий также позволяет работать над одним проектам нескольким разработчикам, которые так же могут заливать свою часть кода.

GitLab — это хранилище открытого исходного кода и платформа для совместной разработки. GitLab предлагает место для онлайн хранения кода и совместной разработки крупных программных проектов. Репозиторий включает в себя контроль версий, позволяющий размещать различные цепочки разработки и версии, что позволяет пользователям проверять предыдущий код и выполнять откат к нему в случае непредвиденных проблем. GitLab является конкурентом GitHub.

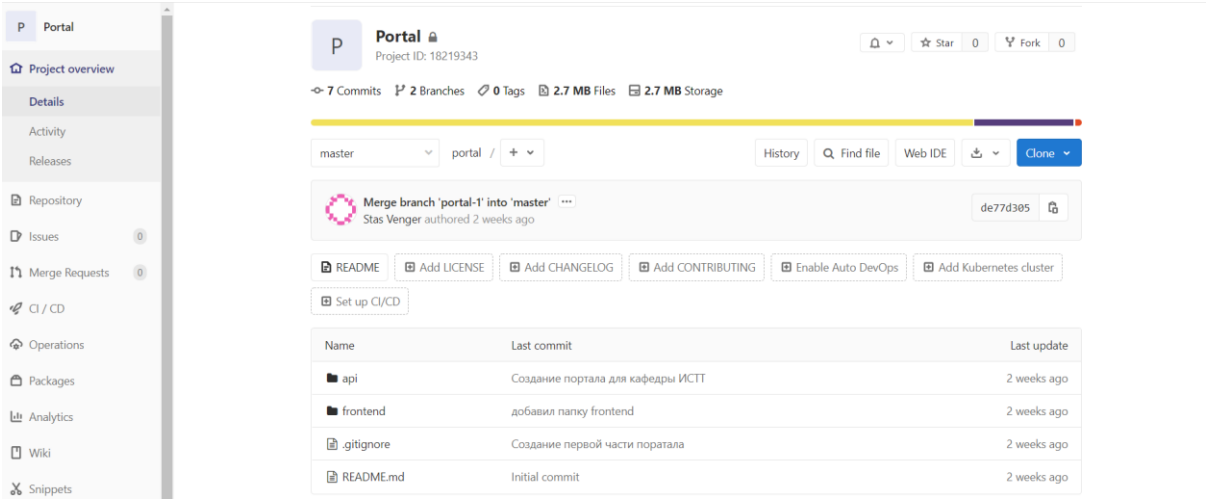

Рисунок 3 – Репозиторий для информационного портала

2.4 Компоненты и структура приложения

Третий этап. Разделение проекта на отдельные контейнеры, компоненты и хранилище store. Основные папки-разделы их еще называют контейнерами они хранятся в основной директории Containers. В каждой папке находится два файлы (js, css). Файл js содержит в себе всю разметку написанной при помощи JSX и основные функции со своим стейтом где будут хранится данные для данного раздела. Разделы, которые содержат в себе разметку, функции и стили:

1. Register –для регистрации нового пользователя
2. Login – для авторизации
3. Personal information – личный кабинет
4. Board – доска объявлений
5. Catalog – каталог учебного материала

Если в приложение встречаются похожие элементы их приняты выносить в отдельную директорию, которая называется components. Компоненты – это одинаковые блоки или разделы, к которым можно отнести такие части кода как шапка, навигационное меню и другие части приложения.

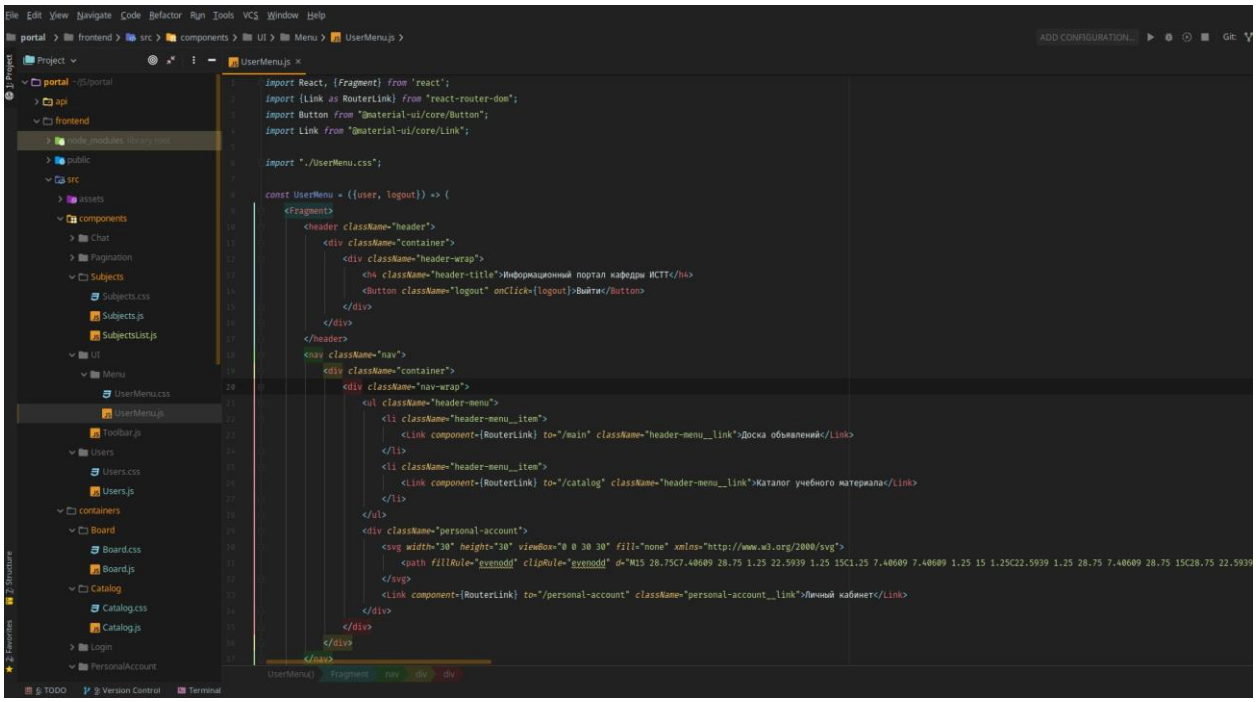

Рисунок 4 – Компонент приложения (UserMenu.js)

Компоненты данного проекта:

1. Toolbar/User menu – навигационное меню и шапка приложения с названием приложения и кнопкой выхода
2. Pagination – пагинация
3. Users – список всех пользователей (доступен только администратору)
4. Subjects – список предметов
5. Chat - чат

2.5 Управление данными. Хранилище приложения Redux

Директория store нужна для того, чтобы хранить в себе такие файлы как actions и reducers. Управлением состоянием приложения может быть сложным. Под состоянием мы определяем различные значения, которые меняются в процессе работы приложения и от которых мы зависим. Например:

1. Данные пользователя
2. Авторизован ли пользователь? Какой у него email, логин или другие параметры
3. Какие есть объявления
4. Список предметов
5. Список всех пользователей

Управление состоянием может быть достаточно сложным, в зависимости от сложности приложения.

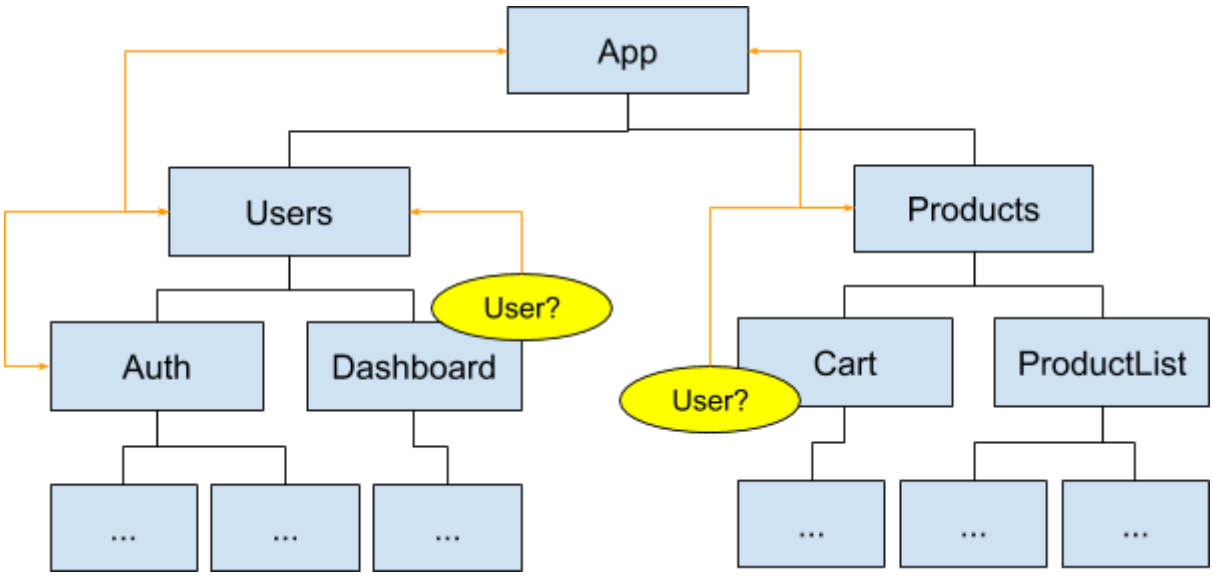

Рисунок 6 – Управление приложением без использования библиотеки Redux

В достаточно сложном приложении, чтобы получить информацию о пользователе в Dashboard достаточно передать его, например из Users. Тогда как компонент Auth, который производит авторизацию пользователя через API, будет тоже таким образом вынужден отправлять информацию в Users.

Теперь если нам надо получить пользователя в компоненте "Объявления или Каталоге учебных материалов" (Cart), то нам придется хранить информацию где-то еще выше по уровню, например в App, и таким образом организовывать сложную связь хранилища из App и функционала, который раскидан по компонентам (Auth, Dashboard, Cart и т.п.).

Тут нам приходит на помощь Redux. Вместо того, чтобы хранить стейт в глобальном компоненте (верхнего уровня), когда стейтом становится достаточно сложно управлять, мы можем воспользоваться Redux, который хранит стейт в собственном хранилище, которое не будет привязано ни к какому компоненту.

Также, Redux организовывает особую схему доступа к "глобальному стейту приложения", в которой компоненты не имеют права изменять его напрямую, а вместо этого они должны выдавать "действия" (actions), которые будут обработаны "редюсерами" (reducers). Также, компоненты теперь могут подписываться на изменения стейта, для того чтобы изменять свое внутреннее состояние и менять пользовательский интерфейс.

Общую схему такого процесса можно описать таким образом:

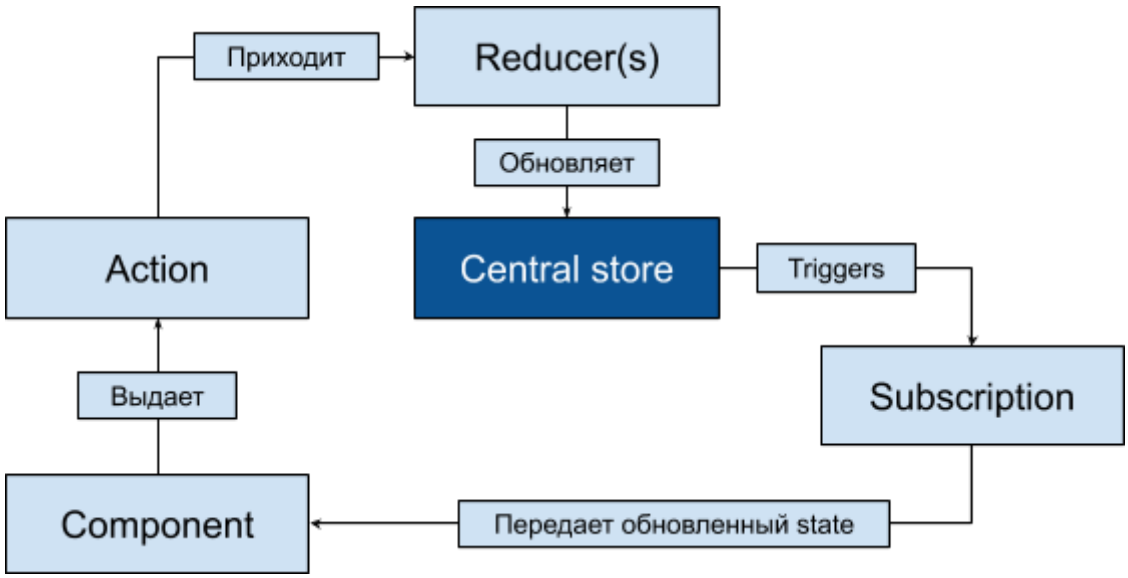

Рисунок 7 – Управление приложением при помощи библиотеки Redux

1. Central Store - JS-объект, который хранит глобальное состояние системы. Его нельзя изменять напрямую, только получать. Можно сказать, что он является read-only.
2. Action - обычный JS-объект с информацией, можно даже назвать его "событием". То есть, что-то должно произойти в системе, о чем сообщает данный компонент. Этот объект как правило имеет поле type - для того, чтобы было понятно, какое именно действие мы выдаем, а также возможно какая-то дополнительная информация, которая должна помочь обновить центральное хранилище.
3. Reducer - функция, которая решает, как именно необходимо изменить центральное хранилище в зависимости от действия, которое в него пришло. Только эта функция может изменить хранилище.
4. Subscription - подписка, которую компонент может осуществить, чтобы подписаться на изменения хранилища. Хранилище при своем обновлении запускает все эти подписки (triggers).

Для работы приложения нужно центральное хранилище - central store. Мы его создаем функцией redux.createStore(). Эта функция принимает один аргумент - главный, или корневой, редьюсер, который мы должны создать заранее.

Эта функция называется "редьюсер" (reducer), потому что похожа на функцию, которую мы передаем в array.reduce(). Также, как и редьюсер для массива она принимает первым аргументом - некий "аккумулятор", в нашем случае это стейт, вторым аргументом - "значение", для нас это будет пришедший экшн.

Рисунок 8 – Управление хранилищем для объявлений (noticeReducers.js)

2.6 Действия приложения (Actions)

Теперь добавим "действия" (actions), которые мы будем выдавать хранилищу, и которые будут приводить к изменению стейта и добавим два вызова разных действий с помощью метода .dispatch

Рисунок 9 – Действия для создания объявлений (noticeActions.js)

Метод dispatch принимает действия в виде JS-объекта. Этот объект должен иметь свойство type, которое указывает, какое действия производится, а также любые другие свойства, несущие дополнительные данные для данного действия.

Свойство type по конвенции имеет значение некой уникальной строки, которая определяет тип действия. Также, по конвенции эта строка должна быть написана заглавными буквами (как некая константа).

1. В первом случае мы создаем действие типа FETCH_NOTICE_SUCCESS, этим действием мы хотим получить весь список наших объявлений.

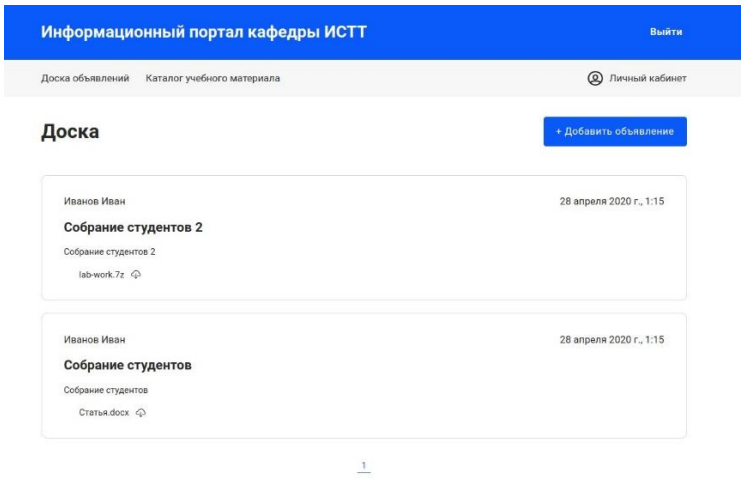

Рисунок 10 – Страница объявлений

2. Во втором – типа CREATE_NOTICE_SUCCESS, этим действием мы хотим создать какое-то новое объявление, в которое мы передадим определенные данные, которые заполнит пользователь в форме.

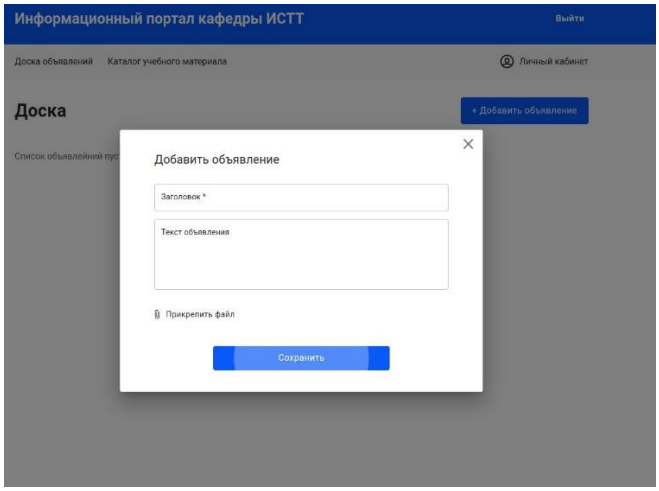

Рисунок 11 – Форма добавления нового объявления

Мы рассмотрели общие принципы Redux, в которых мы рассматривали как отправлять действия (Actions). Эти действия - обычные JS-объекты, состоящие из типа и неких полезных данных (payload).

Также часто используется принцип, когда вместо описания объектов-действий напрямую, мы создаем их с помощью специальных функций - "создателей действий" (action creators).

В самом простом случае — это функция, которая создает объект действия. На нашем примере объявлений, мы можем создать файл src/store/actions/noticeActions.js, который будет содержать логику, связанную с действиями, а также хранить их типы в константах:

```
export const FETCH_NOTICE_SUCCESS = 'FETCH_NOTICE_SUCCESS';
export const CREATE_NOTICE_SUCCESS = 'CREATE_NOTICE_SUCCESS';
export const CREATE_NOTICE_FAILURE = 'CREATE_NOTICE_FAILURE';
export const SET_CURRENT_PAGE = 'SET_CURRENT_PAGE';
export const SET_TOTAL_NOTICES = 'SET_TOTAL_NOTICES';


export const fetchNoticeSuccess = notices => ({type: FETCH_NOTICE_SUCCESS,
notices});
export const createNoticeSuccess = notice => ({type: CREATE_NOTICE_SUCCESS,
notice});
export const createNoticeFailure = error => ({type: CREATE_NOTICE_FAILURE,
error});
export const setCurrentPage = currentPage => ({type: SET_CURRENT_PAGE,
currentPage});
export const setTotalNotices = totalNotices => ({type: SET_TOTAL_NOTICES,
totalNotices});
```

Теперь, вместо того чтобы создавать действия вручную, мы можем использовать эти создатели действий в компоненте. Сначала нам необходимо импортировать нужные создатели действий (в нашем случае все возможные):

```
import {createNotice, fetchNotice, setCurrentPage} from
"../../store/actions/noticeActions";
```

И теперь использовать их:

```
const mapDispatchToProps = dispatch => ({
    logoutUser: () => dispatch (logoutUser ()),
    createdNotice: (noticeData, pageNumber, noticePerPage) => dispatch
(createNotice (noticeData, pageNumber, noticePerPage)),
    fetchNotice: (pageNumber, noticePerPage) => dispatch (fetchNotice
(pageNumber, noticePerPage)),
    setCurrentPage: pageNumber => dispatch (setCurrentPage(pageNumber)),})
```

Теперь компонентам даже и не нужно знать о том, что за логика происходит при создании и отправки действий, мы всего лишь используем готовые создатели действий. Отметьте, что мы вызываем эти создатели, и таким образом передаем то, что они

возвращают (в нашем случае, JS-объекты действий) в метод dispatch. Также, мы можем использовать константы с типами экшнов в редьюсере:

Файл src/store/reducers/noticeReducers.js

```
import {
    CREATE_NOTICE_FAILURE,
    FETCH_NOTICE_SUCCESS,
    SET_CURRENT_PAGE,
    SET_TOTAL_NOTICES
} from "../actions/noticeActions";

const initialState = {
    notices: [],
    noticeError: null,
    noticePerPage: 5,
    currentPage: 1,
    totalNotices: 0
};

const noticeReducers = (state = initialState, action) => {
    switch (action.type) {
        case FETCH_NOTICE_SUCCESS:
            return {
                ...state,
                notices: action.notices};
        case CREATE_NOTICE_FAILURE:
            return {...state, noticeError: action.error};
        case SET_CURRENT_PAGE:
            return {...state, currentPage: action.currentPage};
        case SET_TOTAL_NOTICES:
            return {...state, totalNotices: action.totalNotices};
        default:
            return state;
    }
};

export default noticeReducers;
```

В этом случае, технически мы можем запрашивать данные с API в компоненте, также как мы это делали раньше и затем отправлять соответственные действия (одно перед запросом, затем при получении данных или ошибках). Однако, также можно рассмотреть особый подход, который часто используется при разработке с Redux.

Redux поддерживает некую расширяемость, благодаря которой мы можем использовать middleware (специальная библиотека-посредник), который встраивается в Redux и позволяет например обрабатывать действия особых типов. К примеру, библиотека redux-thunk позволяет нам определять действия не только в виде объектов, но и с помощью функций. Таким образом, мы можем определить специальный "создатель действий", который будет возвращать нам уже целую функцию, внутри которой мы теперь будем вольны делать что захотим, даже делать запросы. Сначала проведем некоторые подготовления - установим axios и redux-thunk

$ npm install --save axios redux-thunk

Начнем с того, что определим при создании хранилища установленный middleware:

```
import {applyMiddleware, combineReducers, compose, createStore} from
"redux";
import thunkMiddleware from 'redux-thunk';
import {createBrowserHistory} from 'history';
import {connectRouter, routerMiddleware} from 'connected-react-router';
import {loadFromLocalStorage, saveToLocalStorage} from "./localStorage";
import axios from '../axiosBase';
import noticeReducers from "./reducers/noticeReducers";
import usersReducers from «. /reducers/usersReducers";

export const history = createBrowserHistory();

const rootReducer = combineReducers({
    router: connectRouter(history),
    users: usersReducers,
    notices: noticeReducers,
});

const composeEnhancers = window.__REDUX_DEVTOOLS_EXTENSION_COMPOSE__ ||
compose;

const middleware = [
    thunkMiddleware,
    routerMiddleware(history)
];

const enhancers = composeEnhancers(applyMiddleware(...middleware));

const persistedState = loadFromLocalStorage();

const store = createStore(rootReducer, persistedState, enhancers);

store.subscribe(() => {
    saveToLocalStorage({
        users: {
            user: store.getState().users.user
        }
    });
});

axios.interceptors.request.use(config => {
    try {
        config.headers['Authorization']= store.getState().users.user.token;
    } catch(e) {
        // do nothing, user is not logged in
    }

    return config
});

export default store;
```

Теперь настроим instance для axios:

Файл src/axiosBase.js

```
import axios from "axios";
import {apiURL} from "./constants";

const instance = axios.create({
    baseURL: apiURL
});

export default instance;
```

Теперь мы можем использовать и axios и функции-экшны, поэтому напишем такой создатель экшнов в actions.js:

```
import axios from '../../axiosBase';

export const createNotice = (noticeData, pageNumber, noticePerPage) => {
    return (dispatch, getState) => {
        const token = getState().users.user.token;
        return axios.post('/notice', noticeData, {headers: {'Authorization':
token}}).then(
            () => {
                dispatch(fetchNotice(pageNumber, noticePerPage));
                NotificationManager.success('Объявление создано');
                dispatch(createNoticeSuccess());
                dispatch(push('/main'));
            }, error => {
                if (error.response) {
                    dispatch(createNoticeFailure(error.response.data))
                } else {
                    dispatch(createNoticeFailure({global: 'Нет
соединения'}))
                }
            }
        )
    };
};
```

Как видите, данный создатель возвращает функцию, которая принимает аргументы, который мы назвали dispatch и getState. Как только передать такую функцию в dispatch, ее обработает redux-thunk и вызовет, передав dispatch в качестве первого параметра. Теперь мы внутри этой функции можем вызывать действия асинхронно.

2.7 Подключение редьюсеров и действий к основным файлам проекта

Итак, редьюсер для действий, связанных с построением объявления, готов, давайте использовать его в деле и свяжем наш контейнер Board с Redux-ом. Для начала, перейдем в Board и заимпортируем в него функцию connect из библиотеки react-redux:
Файл src/containers/Board/Board.js

```
import {connect} from 'react-redux';
```

Теперь мы можем написать две функции для connect-а, чтобы связать "глобальный" стейт с props нашего компонента, а также создать функции, вызывающие

нужные действия. Перед этим не забудем также импортировать создатели действий, чтобы мы могли эти действия отправлять.

```
const mapStateToProps = state => ({
    user: state.users.user,
    error: state.notices.noticeError,
    notices: state.notices.notices,
    noticePerPage: state.notices.noticePerPage,
    currentPage: state.notices.currentPage,
    totalNotices: state.notices.totalNotices
});

const mapDispatchToProps = dispatch => ({
    logoutUser: () => dispatch(logoutUser()),
    createdNotice:     (noticeData,     pageNumber,     noticePerPage)     =>
dispatch(createNotice(noticeData, pageNumber, noticePerPage)),
    fetchNotice:           (pageNumber,           noticePerPage)           =>
dispatch(fetchNotice(pageNumber, noticePerPage)),
    setCurrentPage: pageNumber => dispatch (setCurrentPage(pageNumber)),
});
```

Подключаем функции и компонент через connect:

```
export default withRouter(connect(mapStateToProps, mapDispatchToProps)(Board));
```

Теперь для отображения добавленных объявлений необходимо лишь создать функцию, которая пройдет по массиву и отобразит каждое объявление.

```
let noticeCard = null;
        if (this.props.notices) {
            noticeCard = this.props.notices.map(notice => {
                return (
                    <div className="board-post" key={notice._id}>
                        <div className="board-post__top">
                            <span>{notice.user.secondName}
{notice.user.firstName}</span>
                            <span><Moment format='LLL'
locale='ru'>{notice.dateTime}</Moment></span>
                        </div>
                        <h4 className="board-
post__title">{notice.title}</h4>
                        <p className="board-
post__desc">{notice.description}</p>
                        <div className="board-post-file">
                            {
                                notice.noticeFile.map(file => {
                                    return (
                                        <a href={apiURL +
'/uploads/notice/' + file} key={file}>{file}</a>
                                    )
                                })
                            }
                        </div>
                    </div>
                )
            })
        }
```

На подобие данного функционала были созданы файлы для других разделов приложения, таких так:

Регистрация/ Авторизация

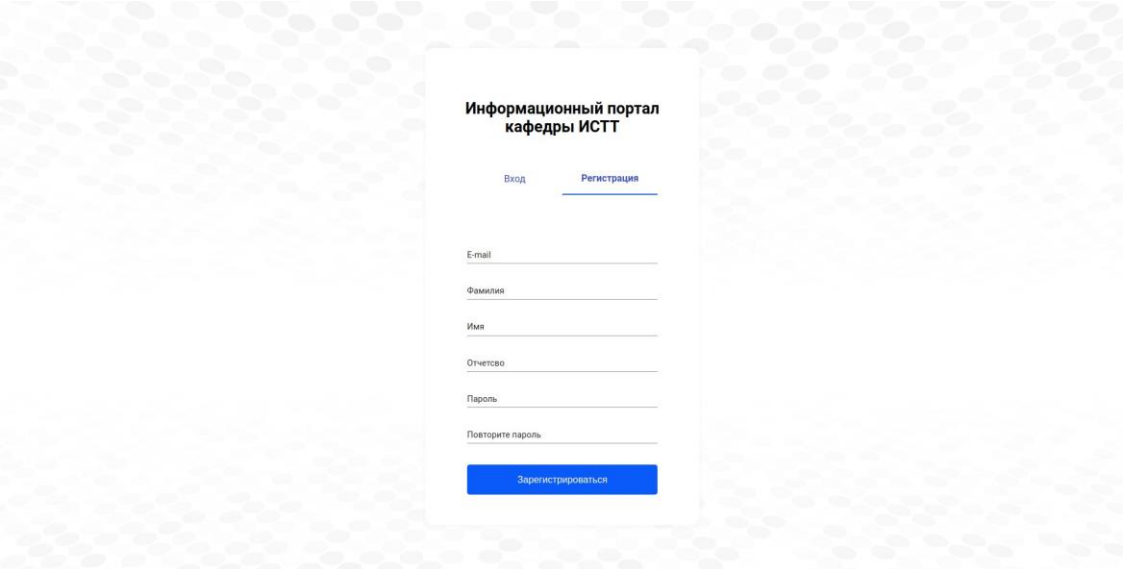

Рисунок 12 – Страница регистрации

Добавление новых пользователей администратором

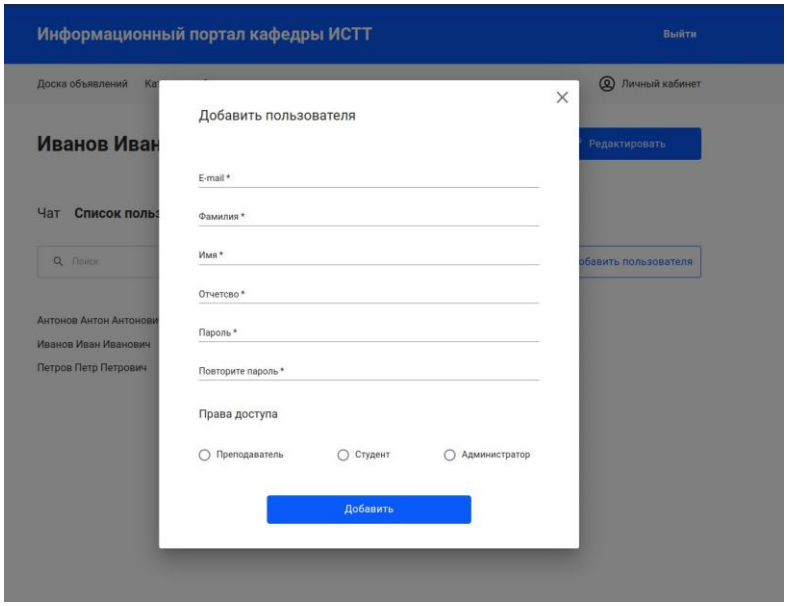

Рисунок 13 – Форма добавления новых пользователей

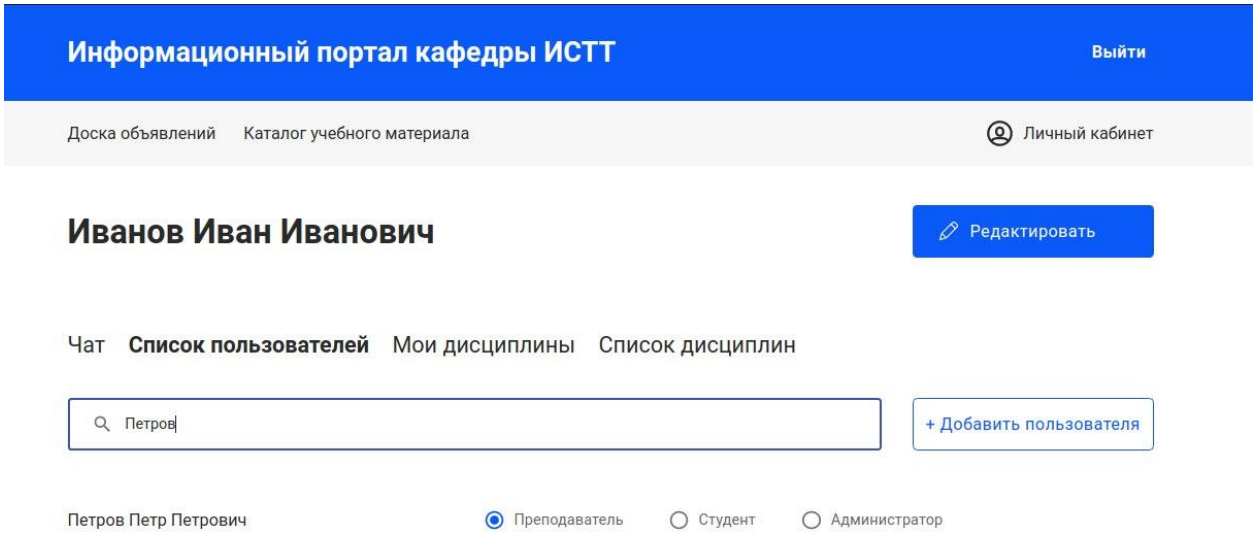

Рисунок 14 – Страница всех пользователей (доступна администратору)

Список дисциплин

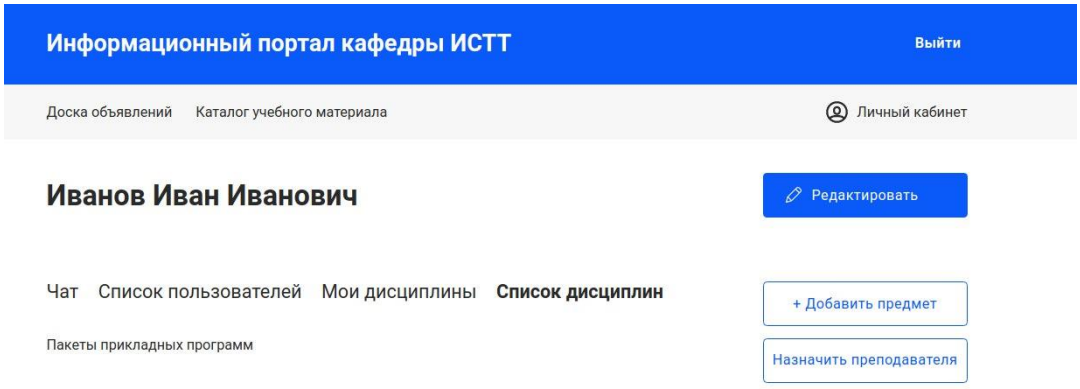

Рисунок 15 – Страница всех дисциплин (доступна администратору)

Глава 3. Разработка серверной части и базы данных

3.1 Установка библиотек для сервера и базы данных

Node.js — это программная платформа, основанная все на том же движке V8, о котором мы уже с вами говорили, которая позволяет с помощью языка JavaScript писать любые приложения, а не только те, которые строго привязаны к браузеру. Можно сказать, что Node.js превращает язык JavaScript из узкоспециализированного языка (для браузера) в язык общего назначения.

Таким образом, с помощью Node.js можно создать на JavaScript все от командной строки до HTTP сервера. Для разработки нужно установить Node.js. Для этого необходимо зайти на официальный сайт https://nodejs.org/ и скачать установщик LTS версию.

Запустить сам установщик и дождаться завершения установки. После этого в проекте была создана отдельная папка, для серверной части которую назвал api. И создадим еще файл, который назовем server.js, его можно создать вручную или при помощи команды *touch server.js.*

При добавление каких-то новых библиотек в данную папку добавиться так же папка node_modules. Данная директория всегда должна игнорироваться версионным контролем, так как содержит «зависимости», которые можно и нужно получать из хранилища. Но в данном случае возникает следующий вопрос, как другой разработчик или кто-то еще поймет, какие зависимости нужны для работы приложения? Для этого используется специальный файл package.json, который должен быть в node.js проекте, и быть под версионным контролем. Именно в этом файле можно указать, какие зависимости нам нужны, а также некоторая другая информация.

К счастью, npm содержит в себе инструмент для автоматического создания такого файла. Для этого наберем в нашей директории через консоль команду:
npm init

Она спросит ряд вопросов, такие как названия проекта, текущей версии, автора и прочего. Многие из этих полей можно пропускать, если на данный момент у нас нет информации по этому поводу. После этого эта команда покажет будущий package.json файл, который будет создан. Если все хорошо, отвечаем "yes" и файл создается.

Теперь нужно добавить этот файл в репозиторий. Таким образом наш репозиторий сейчас будет содержать три файла:

- .gitignore - здесь должна быть строка, указывающая на игнорирование node_modules
- package.json - содержит информацию о зависимостях
- server.js - ваша программа, использующая эти зависимости

Теперь, если кто-либо другой "склонирует" ваш репозиторий, он сможет установить зависимости командой:

npm install

Без передачи команде npm install аргумента, содержащего название зависимости, она попытается прочитать файл package.json и установить зависимости оттуда. Если у нас уже есть package.json файл, можно добавлять туда зависимости таким образом:
npm install express --save

При этом библиотека express помимо того что будет скачана и установлена, также будет добавлена в свойство dependencies файла package.json.

3.2 Дополнительные библиотеки для разработки серверной части и базы данных

Для разработки серверной части нужны дополнительные библиотеки, которые позволят обрабатывать запросы, загружать файлы и шифровать данные. Для этого их нужно установить и при помощи команды npm install название библиотеки --save.
Список библиотек:

- Express
- Cors
- Bcrypt
- Date-fns
- Mongoose
- Multer
- Nanoid
- Socket.io

После установки всех библиотек в файл package.json в зависимости должны добавиться вышеперечисленные библиотеки.

```
api > {} package.json > {} devDependencies
{
  "name": "backend",
  "version": "1.0.0",
  "description": "",
  "main": "server.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1",
    "start": "node server.js",
    "dev": "nodemon server.js",
    "seed": "node fixtures.js"
  },
  "author": "",
  "license": "ISC",
  "dependencies": {
    "bcrypt": "^4.0.1",
    "cors": "^2.8.5",
    "date-fns": "^1.3.0",
    "express": "^4.17.1",
    "mongoose": "^5.9.3",
    "multer": "^1.4.2",
    "nanoid": "^2.1.11",
    "socket.io": "^2.3.0"
  },
  "devDependencies": {
    "nodemon": "^2.0.2"
  }
}
```

Рисунок 16 – файл package.json для серверной части

Express.js – это облегченная платформа веб-приложений, помогающая организовать веб-приложение в архитектуру MVC на стороне сервера. Эта библиотека нужна для того чтобы наше серверное приложение могло работать с MongoDB(для моделирования). Express.js в основном помогает управлять всем, от маршрутов до обработки запросов.

Cors – расшифровывается как (cross origin resource sharing) это механизм браузера, который обеспечивает контролируемый доступ к ресурсам, расположенным за пределами данного домена. Это расширяет и добавляет гибкость в политику одного и того же происхождения (SOP).

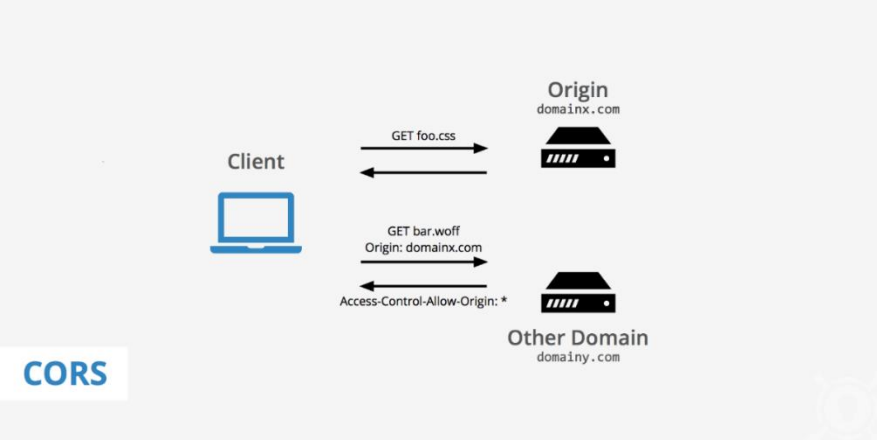

Рисунок 17 – Работа библиотеки CORS

Библиотека bcrypt позволяет шифровать данные. Например, при регистрации нового пользователя его пароль будет зашифрован и будет отображаться набором каких-

то символов, но при этом сам пользователь сможет авторизоваться с тем паролем, который он указал при регистрации. Данная шифровка прописана в модели базы данных User.js.

```
UserSchema.methods.checkPassword = function (password) {
  return bcrypt.compare(password, this.password);
};
UserSchema.pre('save', async function (next) {
  if (!this.isModified('password')) return next();
  const salt = await bcrypt.genSalt(SALT_WORK_FACTOR);
  const hash = await bcrypt.hash(this.password, salt);
  this.password = hash;
  next();
});
```

Date-fns это библиотека которая позволяет преображать дату или время в определенный формат. В данном проекте она используется для того, чтобы могли посчитать время последнего визита пользователя и записать в модель новое значение для поля isOnline.

Mongoose – служит для создания моделей и самой базы данных MongoDB.

Multer – при помощи данной библиотеки можно загружать файлы.

Nanoid – нужна для генерации случайных чисел, букв и символов. В данном проекте эта библиотека используется для генерации токена пользователя.

```
UserSchema.methods.generateToken = function () {
  this.token = nanoid();
};
```

Socket.io – Socket.IO позволяет двунаправленную связь между клиентом и сервером. Двунаправленная связь включается, когда клиент имеет Socket.IO в браузере, а сервер также интегрировал пакет Socket.IO. Хотя данные могут быть отправлены в нескольких формах, JSON является самым простым.

Для установления соединения и обмена данными между клиентом и сервером Socket.IO использует Engine.IO. Это реализация более низкого уровня, используемая под капотом. Engine.IO используется для реализации сервера, а Engine.IO-клиент используется для клиента.

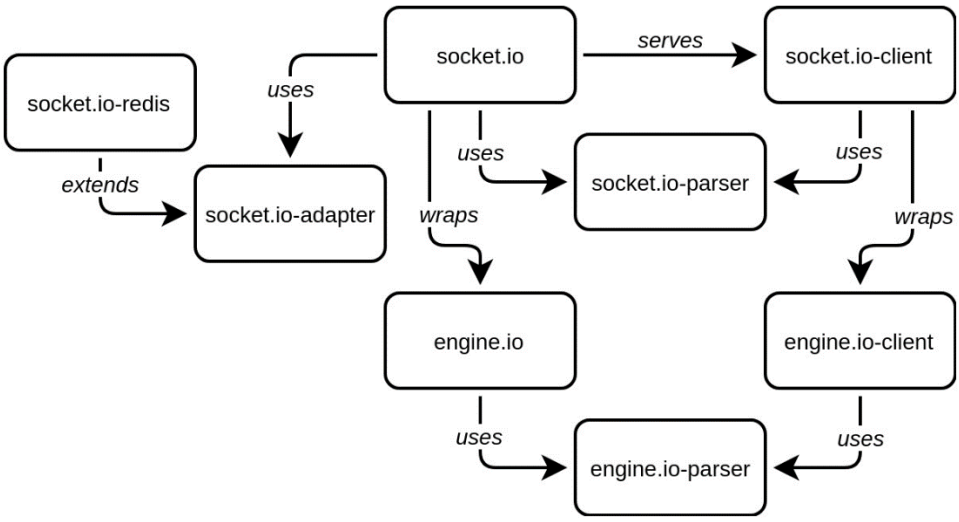

Рисунок 18 – Socket.io

3.3 База данных. MongoDB.

Мы рассмотрели, что базы данных бывают реляционные. В этом случае данные хранятся в отдельных таблицах, каждая таблица содержит в себе информацию о какой-то конкретной сущности. Мы должны определить структуру этих таблиц вручную перед тем, как использовать.

Также существуют другие варианты хранения данных. Например, мы можем хранить информацию в файлах, но тогда мы теряем возможность удобно находить, сортировать данные, а также теряем оптимизацию, которую нам дает система управления базами данных. Нам придется проверять целостность данных вручную, заботиться о безопасности и о том, как хранить данные, читать и записывать их обратно.

Нечто среднее между хранением данных в файлах и реляционными базами нам дают другие базы данных - нереляционные, или их еще называют NoSQL, поскольку в них не используется язык SQL для доступа к данным. Одним из примеров такой базы данных является MongoDB.

В современной разработке очень часто любят относиться к записям, хранящимся в таблицах, как к объектам модели данных. То есть, существует некая модель данных в приложении, которая описывает, какие существуют "сущности", какие свойства они

имеют, как относятся друг к другу. То есть, иными словами, для каждой "таблицы" в базе мы создаем некий класс, который содержит все свойства, соответствующие данным из таблицы. И каждый экземпляр этого класса является "сущностью" нашей системы, связанный с соответствующей записью из таблицы.

Существуют специальные библиотеки, которые помогают нам организовывать такую модель данных, и даже сами умеют подключаться к нужным базам данных. Такие библиотеки часто называются: ORM, что означает "object-relational mapping", то есть занимаются, по сути, преобразованием реляционных данных из БД в объекты языка программирования, согласно принципам ООП. Для MySQL и прочих реляционных баз данных название ORM оправдано, тогда как для Mongo, которая не является реляционной базой, больше подходит название ODM (object-data mapping). Именно такой ODM является библиотека mongoose, позволяющая упрощать запросы к базе MongoDB и обращаться с каждым документом, полученным из этой базы, как с неким объектом, который может иметь собственные дополнительные методы, в том числе и такие, которые позволяют сохранять данные в базу.
Официальный сайт библиотеки:

http://mongoosejs.com/

После установки mongoose нам нужно настроить подключение к БД. Для этого уже в созданном файле server.js должны подключить данную библиотеку.
const mongoose = require ('mongoose');

Отметьте, что для соединения мы должны передать URL к серверу MongoDB, а также через слэш должно идти название базы данных, к которой мы подключаемся. То есть, что-то вроде:
mongodb://localhost/portal данный Url можно вынести в отдельный файл который назовем config.js. В данном файле мы так же будем хранить Url для загрузки картинок и файлов в определенные папки.

```
const path = require('path');

const rootPath = __dirname;

module.exports = {

    rootPath,
```

```
  uploadPathNotice: path.join(rootPath, 'public/uploads/notice'),
  uploadPathSubject: path.join(rootPath, 'public/uploads/subject'),
  uploadPathMessageFile: path.join(rootPath, 'public/uploads/messageFile'),
  dbUrl: 'mongodb://localhost/portal',
  mongoOption: { useNewUrlParser: true, useCreateIndex: true, useUnifiedTopology: true, useFindAndModify: false},
};
```

Вернемся к файлу server.js тут мы должны подключить соединение к базе данных через функцию, к которую будем записывать все роуты наших разделов приложения.

```
const port = 8000;
mongoose.connect(config.dbUrl, config.mongoOption).then(() => {
  http.listen(port, () => {
    app.use('/admin/users', adminUsers);
    app.use('/users', users);
    app.use('/notice', notice);
    app.use('/subject', subject);
    app.use('/dialog', dialog);
    app.use('/messages', message);
    console.log(`Server started on ${port} port`);
  });
});
```

Передаем вторым аргументом функцию, которая должна выполниться, когда произойдет успешное соединение. Заметьте, что мы не получаем ничего из этой функции, и не передаем никакого объекта подключения в роутеры (admin/users, users, notice, subject, dialog, messages). Дело в том, что мы будем настраивать отдельный объект - так называемую "модель" для произведения непосредственных запросов к базе. Тогда как подключение mongoose будет для всех этих моделей одно и то же, т.е. глобальное.

Теперь давайте создадим модели. Как было сказано ранее, модель должна отражать все свойства, которые есть в нашей "сущности". Для начала, создадим файлы:

- /models/User/User.js

- /models/Dialog/Dialog.js
- /models/Messages/Messages.js
- /models/Notice/Notice.js
- /models/Subjects/Subjects.js

В каждом модели мы должны добавить библиотеку mongoose.

```
const mongoose = require('mongoose');
```

Теперь нам нужно создать "схемы", которой будут подчиняться наши модели. Схема отражает все поля, которые есть в нашей модели. Например, для нашего объявления (notice) мы можем перечислить поля, которые будем использовать:

```
const mongoose = require('mongoose');

const Schema = mongoose.Schema;

const NoticeSchema = new Schema ({
   user: {
      type: Schema.Types.ObjectId,
      ref: 'User',
      required: true
   },
   title: {
      type: String,
      required: [true, 'Поле обязательно для заполнения']
   },
   description: String,
   noticeFile: [String],
   dateTime: String
});

const Notice = mongoose.model('Notice', NoticeSchema);
```

module.exports = Notice;

Заметьте, что в объекте, передаваемом в конструктор «схемы», мы указываем свойства с теми же названиями, что и свойства, которые мы используем. Значения этих свойств - конфигурация поля. Получается, что мы можем указать либо просто тип, которым должно быть это поле (dateTime: String), так и объект с дополнительной конфигурацией. Здесь мы настраиваем поля user, title чтобы они были обязательными для модели. Поле noticeFile будет являться массивом строк, куда будут записываться названия файлов.

Итак, в конце мы экспортируем созданную модель и можем теперь ее использовать в роутере.

3.4 Роутеры

Но чтобы использовать модели и делать какие-то запросы необходимо создать файлы (роутеры) где мы и будем создать определенные запросы за добавление, получение или редактирование. Роутеры:

- /app/admin/users.js
- /app/dialog.js
- /app/messages.js
- /app/notice.js
- /app/subject.js
- /app/user.js

Для примера возьмем роутер notice.js и за экспортируем модель /models/Notice.js и другие необходимые библиотеки и файлы для дальнейшей работы.

```
const express = require('express');

const multer = require('multer');

const config = require('../config');

const auth = require('../middleware/auth');

const tryAuth = require('../middleware/tryAuth');

const permit = require('../middleware/permit');
```

```
const Notice = require('../models/Notice');
```

Далее создадим отдельные роутеры на получение и добавления объявлений.

```
router.get('/', tryAuth, async (req, res) => {
    try {
        const notice = await Notice.find().populate('user', '_id firstName secondName').sort({dateTime: -1});
        const page = parseInt(req.query.page);
        const limit = parseInt(req.query.limit);
        const startIndex = (page - 1) * limit;
        const endIndex = page * limit;
        const notices = {};
        if (endIndex < notice.length) {
            notices.next = {
                page: page + 1,
                limit: limit
            };
        }
        if (startIndex > 0) {
            notices.previous = {
                page: page - 1,
                limit: limit
            };
        }
        notices.notices = notice.slice(startIndex, endIndex);
        notices.totalNotices = notice.length;
```

```
        res.send(notices);
    } catch (e) {
        res.sendStatus(500);
    }
});
router.post('/', [auth, permit('admin')], upload.array('noticeFile', 5), (req, res) => {
    const noticeData = {
        user: req.user._id,
        title: req.body.title,
        description: req.body.description,
        noticeFile: [],
        dateTime: new Date().toISOString()
    };
    if (req.files) {
        noticeData.noticeFile = req.files.map(file => file.filename)
    }
    const notice = new Notice (noticeData);
    notice.save()
        .then(results => res.send(results))
        .catch(error => {
            res.status(400).send(error)
        } );
});
```

При помощи первого роутера router.get мы будем получать все объявления, которые создаст администратор. Так же в данные роутер мы передадим middleware

tryAuth который будет делать проверку авторизован пользователь или нет. Если пользователь не авторизован, то наше приложение выдаст ошибку. Для выборки мы используем статические методы класса Notice, типа find.

```
const notice = await Notice.find().populate('user', '_id firstName
secondName').sort({dateTime: -1});
```

При помощи populate мы можем получить также данные о пользователе, который создал объявление. Метод sort служит для сортировки созданных объявлений по дате. Для создания новой записи будем использовать роутер router.post в котором мы сначала создаем экземпляр этого класса. При этом мы можем в конструктор сразу передать начальные данные. Также можно затем менять свойства в этом объекте напрямую. После вызова метода save() документ сохраняется в базу данных. Если что-то пошло не так, например не прошли правила валидации данных, т.е. не пришли данные, то произойдет ошибка валидации.

```
const notice = new Notice (noticeData);

  notice.save()

    .then(results => res.send(results))

    .catch(error => {

      res.status(400).send(error)

    });
```

3.5 Загрузка файлов на сервер

Для отправки файлов существует определенный стандарт создания запроса. Отправка бинарных данных имеет определенную сложность, в том смысле, что это не текст, и поэтому должен восприниматься сервером именно побайтово как файл. Поэтому в случае формата "multipart/form-data" для каждого файла создается собственный "кусок" запроса, который содержит дополнительную информацию о файле, а также само содержимое файла. Браузеры умеют также определять, какого типа данный файл и какие указать заголовки, чтобы файл воспринялся сервером.

Как правило, для отправки файлов на сервер раньше использовался такой способ: создается форма (HTML-тег "<form>") в которую указывается, что тип кодирования

данных должен быть multipart/form-data, и в этой форме можно создавать <input>-ы с типом "file":

```
<form enctype="multipart/form-data" method="POST" action="/notice ">
 …
 <input type="file" name="image" />
 <button type="submit">Save</button>
</form>
```

Если нажать на "Save" в такой форме, то браузер сформирует запрос методом POST, с использованием кодировки multipart/form-data, и отправит данные в запросе, с перезагрузкой страницы:

```
POST /notice HTTP/1.1
Host: localhost:8000
Content-Type: multipart/form-data; boundary=----
WebKitFormBoundary7MA4YWxkTrZu0gW
Cache-Control: no-cache
Postman-Token: 5fa14b7e-d12f-b28c-7dc6-7fbf1e9ab947
------WebKitFormBoundary7MA4YWxkTrZu0gW
Content-Disposition: form-data; name="image"; filename="1031013673.jpg"
Content-Type: image/jpeg
------WebKitFormBoundary7MA4YWxkTrZu0gW--
```

Итак, нам здесь интересно следующее: в заголовке запроса есть такой тип данных: Content-Type: multipart/form-data, а также указан boundary:

boundary=----WebKitFormBoundary7MA4YWxkTrZu0gW

boundary указывает, как запрос разделяется на части, и по этой "отрезной линии" сервер должен будет разобрать запрос на части. К каждому кусочку добавляется заголовок Content-Disposition: form-data, указывающий на то что каждая часть является отдельным "элементом" данных формы, а также дополнительные данные. Например, для файлов указывается filename, в который записывается оригинальное название файла.

Также сам файл отправляется в этой части запроса, но его не так просто увидеть, т.к. браузеры и Postman не показывают бинарные данные в запросе.

3.6 Обработка multipart/form-data на сервере

Однако, теперь сервер не понимает наши данные и т.к. у нас нет никакой валидации, то в базу, по сути, записываются пустые объекты, без данных вообще. Давайте добавим специальный middleware, который поможет нам получить файл и затем что-то с ним сделать. По умолчанию в express нет такого middleware и нам надо будет найти и установить таковой.

Очень часто для этих целей используется middleware, который называется multer. Эта библиотека позволяет сразу же сохранять загруженные файлы туда, куда мы хотим их положить. Для этого немного подготовимся.

Для начала, создадим директорию для файлов, в которую будут загружаться изображения. Создадим директорию /public/uploads. В этой директории создадим файл .gitignore.

Сам этот файл добавим в версионный контроль. Это нужно, чтобы директория public/uploads зафиксировалась в версионном контроле, ибо git не поддерживает пустые директории (он не хранит их у себя в индексе). В директории обязательно должен быть хотя бы один файл, и именно такой мы сейчас и создали - .gitignore.

Т.к. мы не хотим, чтобы загруженные файлы добавлялись в версионный контроль (также как и данные в базе данных), мы в файле .gitignore добавим "звездочку" - * (то есть игнорировать все файлы. Звездочка гарантирует, что никакие файлы кроме уже добавленного .gitignore не добавятся в версионный контроль. Теперь создадим в корне проекта файл config.js, в котором у нас будут храниться различные конфигурационные значения.

```
const path = require('path');

const rootPath = __dirname;

module.exports = {

 rootPath,

 uploadPath: path.join(rootPath, 'public/uploads')
```

```
};
```

Здесь следует отметить специальную глобальную переменную __dirname, которая в момент выполнения JS-файла содержит абсолютный путь к директории, в которой этот JS-файл лежит. Также мы импортировали специальный модуль path из стандартных модулей Node.js, который содержит различные функции для облегчения работы с путями файловой системы. В частности, функция join позволяет "объединять" пути файловой системы. Того же можно добиться через обычную конкатенацию со "слэшами", но этот метод позволяет более умно работать с этими путями, поэтому будем использовать его. Получается, мы экспортируем объект с двумя свойствами - первое (rootPath) позволяет получить "корневой путь", то есть путь к директории с проектом, а второе (uploadPath) позволяет получить путь к директории с загружаемыми пользователями файлами. Теперь, давайте настроим multer. Для этого перейдем в файл notice.js и сконфигурируем multer для работы:

```
const multer  = require('multer');

const path = require('path');

const config = require('../config');

const nanoid = require('nanoid');


const storage = multer.diskStorage({

 destination: function (req, file, cb) {

   cb(null, config.uploadPath);

 },

 filename: function (req, file, cb) {

   cb(null, nanoid() + path.extname(file.originalname));

 }

});
```

Здесь мы настраиваем "хранилище" для multer, таким образом:

- destination - функция, которая определяет, "куда" загружать наш файл. В нашем случае мы передаем config.uploadPath для того, чтобы сохранять загруженные файлы именно там.
- filename - функция, которая определяет, как назвать наш файл. Здесь мы генерируем для файла случайное название с помощью nanoid и добавляем оригинальное расширение файла с помощью функции path.extname, которая умеет вырезать из пути только часть с расширением. При этом мы берем свойство file.originalname, которое содержит оригинальное название файла. Например, мы загружаем изображение my_picture.png. В этом случае функция path.extname('my_picture.png') вернет нам просто '.png'

После того как мы настроили хранилище, мы создаем middleware upload, который принимает эту настройку как опцию:

```
const upload = multer({storage});
```

Теперь мы можем передать middleware в роут, который занимается приемом файлов, а это роут, который отвечает за создание нового продукта:

```
router.post('/', [auth, permit('admin')], upload.array('noticeFile', 5), (req, res) => {
  const noticeData = {
    user: req.user._id,
    title: req.body.title,
    description: req.body.description,
    noticeFile: [],
    dateTime: new Date().toISOString()
  };
  if (req.files) {
    noticeData.noticeFile = req.files.map(file => file.filename)
  }
  const notice = new Notice (noticeData);
  notice.save()
```

```
    .then(results => res.send(results))
    .catch(error => {
        res.status(400).send(error)
    } );
});
```

Вторым аргументом (и далее) мы можем передавать необходимые middleware, в нашем случае мы передали upload.array('noticeFile', 5), который обеспечивает сохранение поля noticeFile в файл, и добавляет в объект запроса (req) свойство file, которое содержит в себе информацию о загруженном файле. Мы хотим сохранить имя файла, которое было сгенерировано, поэтому берем свойство req.file.filename, и сохраняем его как свойство будущего товара. Теперь все товары будут иметь это свойство, если мы загружаем файл. Если файл не загружен, то свойство req.file будет undefined. Отлично, теперь у нас в базе сохраняются названия загруженных изображений. Например, это выглядит так:

```
{
  "title": "Собрвние студентов",
  "description": "Описание",
  "image": "_fmq~vRpD96zMpZjswOvd.jpg",
  "id": "jV7P6eVj3XStkXEgzkWFf"
 }
```

Заметьте, что название изображение не обязательно должно совпадать с ID объекта, к которому мы его загружаем и даже наоборот, сейчас мы его вовсе не сделали совпадающим.

3.7 Раздача статических файлов

Теперь мы можем отобразить загруженные картинки во frontend-части нашего приложения. Для этого мы должны как-то получать доступ к этому изображению. Сейчас, если мы запросим все объявления через API, то мы получим список объявлений

с названиями файлов, но мы не сможем получить сами файлы. Для того, чтобы сервер мог "отдать" файлы, мы должны всего лишь добавить соответствующий middleware:
app.use(express.static('public'));

Так мы говорим express-у, что мы хотим отдавать файлы, находящиеся в директории public относительно файла сервера, как статические. Теперь мы сможем получить указанной выше файл по такому пути:
http://localhost:8000/uploads/Статья.docx

3.8 Авторизация

Под авторизацией обычно понимают способность системы распознавать пользователей различных видов (ролей) или пользователей с разными "разрешениями" и давать или запрещать доступ к тем или иным функциям системы. Как и в случае с аутентификацией, для обеспечения безопасности доступа к чужим данным, авторизация как правило производится на сервере (в backend-части). Некоторые функции дублируются на клиенте, чтобы "не показывать" пользователю части системы, в которые он не имеет доступа. Когда мы говорим про авторизацию, то это может работать различными способами, в зависимости от вашей задачи. Например, для информационного портала в системе есть различные роли пользователей:

- Администратор - имеет доступ ко всей системе. Может свободно редактировать, удалять и создавать материалы, настраивать систему, пользователей и т.п.
- Преподаватель - умеет добавлять материалы для студентов, не имеет доступа к каким-то другим администраторским функциям
- Студент - может просматривать информацию, вести диалог в чате

Либо, у каждого пользователя могут быть определенные разрешения, на основании которых этот конкретный пользователь может или не может производить действия. Этот подход гибче, но часто является более сложным в реализации. Если брать пример выше, то можно представить себе, что в системе есть разные пользователи:

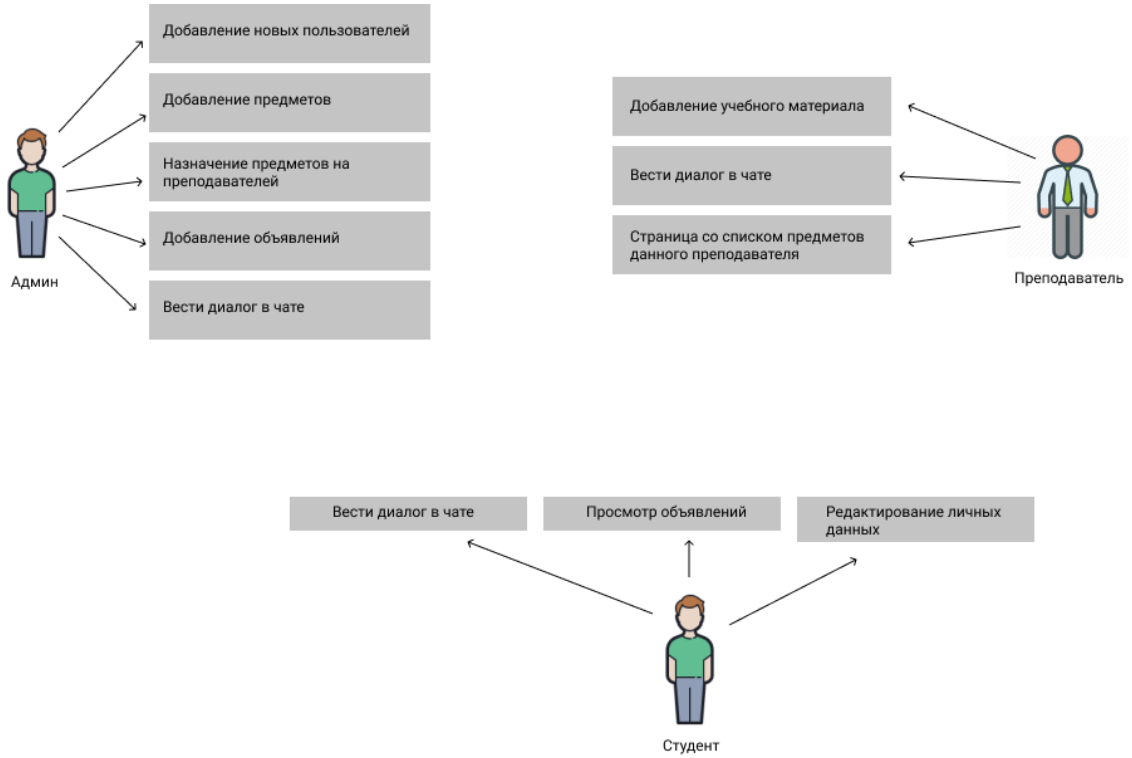

Рисунок 19 - Концептуальная модель

И так далее, то есть некий заранее описанный в системе набор разрешений, каждое разрешение дает право на одно определенное действие. У пользователя соответственно есть некий список этих разрешений. Всякий раз система проверяет наличие того или иного разрешения в списке пользователя при попытке произвести действие и запрещает либо разрешает доступ к этому действию. С точки зрения интерфейса приложения также есть несколько подходов к тому, как организовывать доступы с фронтенд- и бэкэнд- части. Два наиболее популярных:

Мы организовываем нечто вроде "админ-панели", то есть, по сути, второй дополнительный интерфейс, который выглядит совершенно иначе, чем основной (пользовательский), доступ к которому обеспечивается через авторизацию. В этом случае можно сказать, что в приложении есть два разных интерфейса - для администрации и для пользователей.

Прямо в пользовательском приложении добавляется возможность создавать и редактировать сущности, при наличии соответствующих прав. В этом случае можно сказать, что в приложении есть один лишь интерфейс, в котором добавляются элементы и страницы при наличии соответствующих прав.

Прежде чем принимать подобное решение со стороны интерфейса, мы можем проработать эту систему со стороны API. Здесь тоже есть несколько подходов - когда

мы создаем отдельные роуты для доступа пользователей и отдельные роуты для доступов администратора. Т.к. бывает иногда нужно разделять роуты, например, одной сущности, когда администратор может видеть всю информацию о данной сущности, а обычный пользователь - только некоторую. Чтобы не переусложнять API, у нас есть только 3 роли - простой пользователь (студент), преподаватель и администратор. Поэтому, давайте добавим в модель поле role:

```
const UserSchema = new Schema({
  email: {
    type: String,
    required: [true, 'Поле обязательно для заполения'],
    unique: true,
    validate: {
      validator: async  function (value) {
        if (!this.isModified('email')) return;
        const user = await User.findOne({email: value});
        if (user) throw new Error(`Email ${value} уже используется!`);
      }
    }
  },
  role: {
    type: String,
    required: true,
    default: 'student',
    enum: ['admin', 'student', 'teacher']
  },
```

То есть, мы создаем новое поле, которое по-умолчанию будет равно 'student', и может быть либо 'student' либо 'admin' либо 'teacher'. Теперь мы сталкиваемся с некоей проблемой безопасности, если мы обратим внимание на наш роут создания (регистрации) пользователя, то увидим, что мы создаем пользователя из объекта, приходящего нам в req.body:
const user = new User(req.body);

Таким образом возможно в запросе передать нечто вроде:

{"username": "hacker", "password": "123", "role": "admin"} и пользователь создастся с ролью "admin", поскольку нигде мы не проверяем что это поле не может приходить со стороны клиента.

Поэтому давайте будем использовать "белый список" ключей объекта req.body вместо того, чтобы передавать его напрямую в создание экземпляра модели:

```
const user = new User({
  email: req.body.email,
  password: req.body.password
});
```

Таким образом, мы точно уверены, что кроме полей email и password в модель из запроса пользователя ничего другого не попадет. Мы можем создать несколько пользователей в фикстурах, чтобы при очистке базы сразу можно было бы логиниться под известными логинами/паролями. Для этого добавим в fixtures.js:

```
...
  await User.create({
    email: 'user@user.com',
    password: '12345678'
  }, {
    email: 'admin@admin.com',
    password: '12345678',
    role: 'admin'
```

```
});
```

Для реализации авторизации начнем с того, что просто "закроем" некоторые роуты, которые у нас есть — это создание товара, создание категории. Для этого нам понадобится несколько middleware, включая тот, что мы уже использовали - auth.js:

```
const User = require('../models/User');

const auth = async (req, res, next) => {
  const token = req.get('Token');
  if (!token) {
    return res.status(401).send({'message': 'No token present'});
  }
  const user = await User.findOne({token});
  if (!user) {
    return res.status(401).send({'message': 'Token incorrect'});
  }
  req.user = user;
  next();
};

module.exports = auth;
```

Здесь мы проверяем, что в запросе есть Token и существует пользователь с таким токен-ом. Теперь нам нужна дополнительная функция-middleware, которая будет проверять наличие определенной роли у пользователя. Создадим отдельный файл middleware/permit.js:

```
const permit = (...roles) => {
  return (req, res, next) => {
    if (!req.user) {
      return res.status(401).send({'message': 'Unauthenticated'});
```

```
    }
    if (!roles.includes(req.user.role)) {
      return res.status(403).send({'message': 'Unauthorized'});
    }
    next();
  }
};
module.exports = permit;
```

Здесь мы на всякий случай проверяем наличие пользователя в объекте request, т.к. данный middleware необходимо использовать в связке с предыдущим (auth.js). Далее мы проверяем наличие пользовательской роли в массиве roles, который представляет из себя все аргументы в функцию permit. Получается, что если мы передадим один аргумент, то данный массив будет содержать один элемент, если несколько - то несколько. Таким образом мы можем использовать данный middleware так. Добавим проверку на создание объявления:

```
// Notice create
router.post('/', [auth, permit('admin'), upload.single('image')], (req, res) => {
  ...
});
```

Вызывая функцию permit, мы создаем middleware, который проверяет всех пользователей на наличие роли 'admin'. Также мы сделаем для создания новых пользователей:

```
const auth = require('../middleware/auth');
const permit = require('../middleware/permit');
router.post('/', [auth, permit('admin')] (req, res) => {
  ...
});
```

Теперь мы имеем два "защищенных" роута, и пользователи без роли "admin" добавлять туда записи через API не смогут.

3.9 Руководство пользователя

3.9.1 Список пользователей

Данным приложением могут пользоваться следующие пользователи:

- Администратор – добавляет новых пользователей, объявления, предметы и назначает предметы на преподавателя.
- Преподаватель – добавляет учебный материал в раздел «Каталог учебного материала».
- Студент – может просматривать объявления, каталог учебных материалов, скачивать файл, ввести переписку с другими пользователями

3.10.1 Запуск программы

Для запуска приложения необходимо открыть любой браузер и в адресной строке указать ссылку к приложению. После загрузки откроется первая страница авторизации.

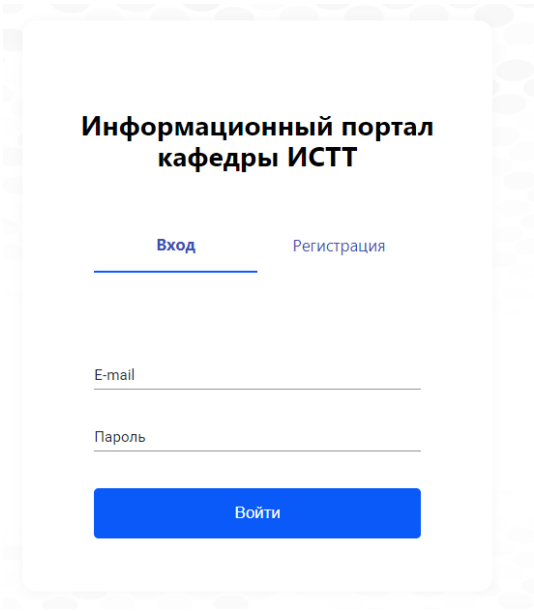

Рисунок 20 – страница авторизации

Если у пользователя нету логина и пароля, то ему необходимо будет зарегистрироваться. Где необходимо указать почту, фамилию, имя, отчество и указать пароль.

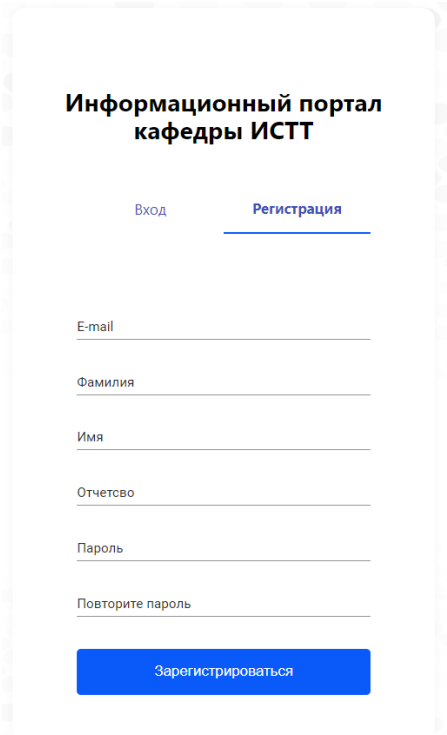

Рисунок 21 – раздел регистрации

После успешной авторизации пользователь попадет на страницу с объявлениями, которые может создать только администратор.

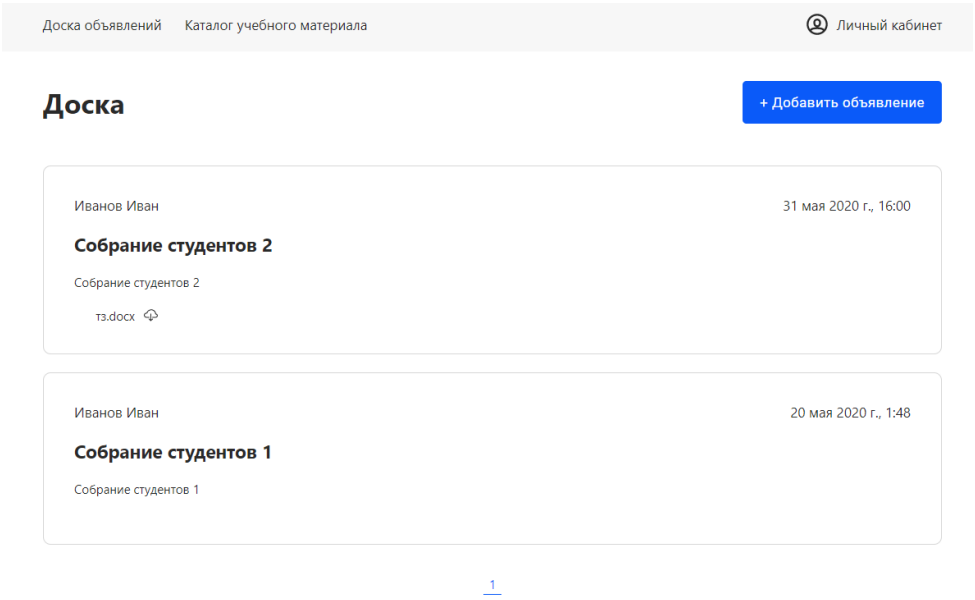

Рисунок 22 – страница объявлений

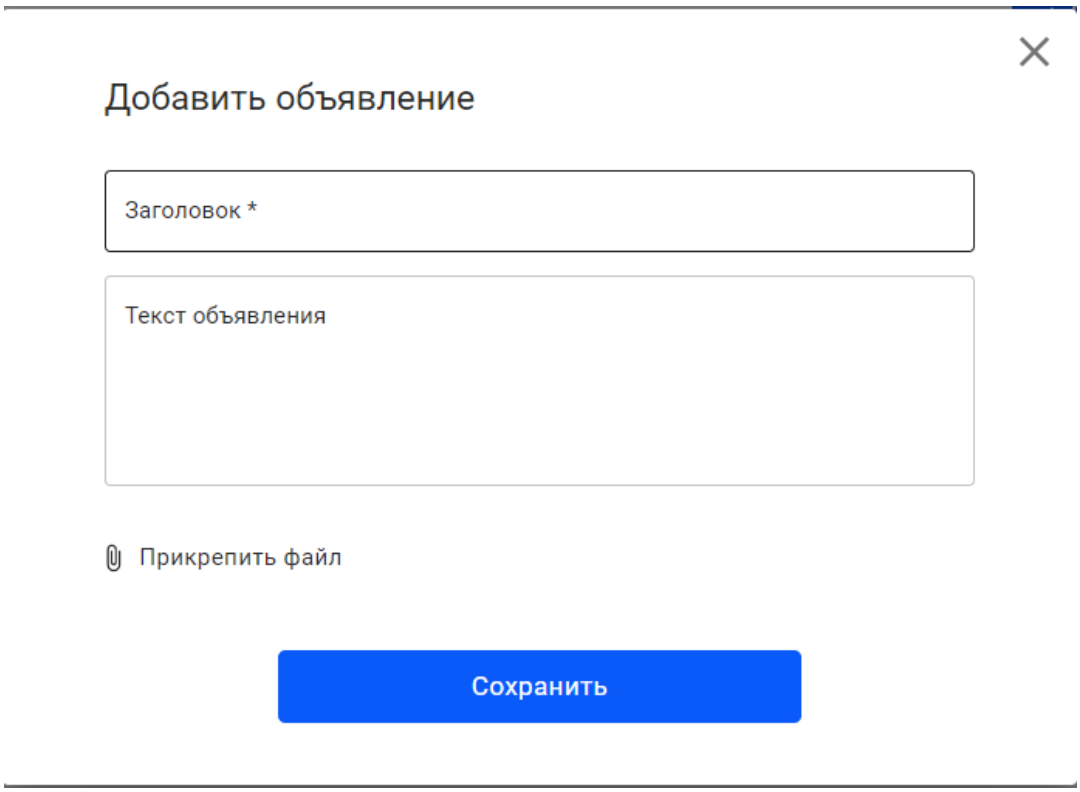

Рисунок 23 – форма создания объявлений

Следующий раздел, который сможет посетить пользователь это каталог учебных материалов, куда будут загружаться файлы преподавателями. И если студент выберет определенного преподавателя, то в списке отобразятся файлы, которые загрузил этот преподаватель.

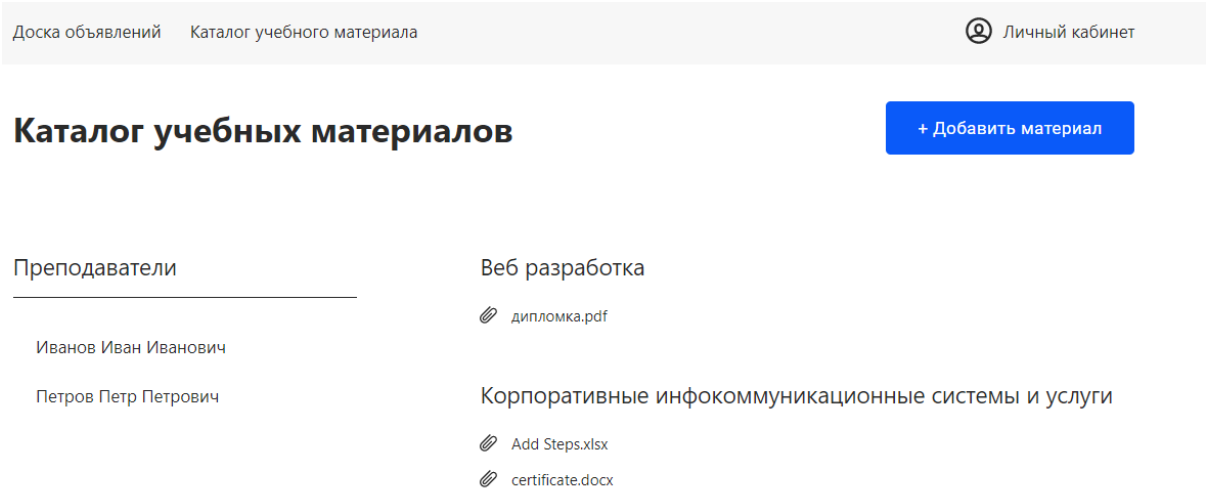

Рисунок 24 – страница учебных материалов

Так же в данном приложение реализован чат, в котором пользователи могут создавать новые диалоги писать сообщения и обмениваться файлами. Чтобы попасть в чат необходимо перейти в личный кабинет.

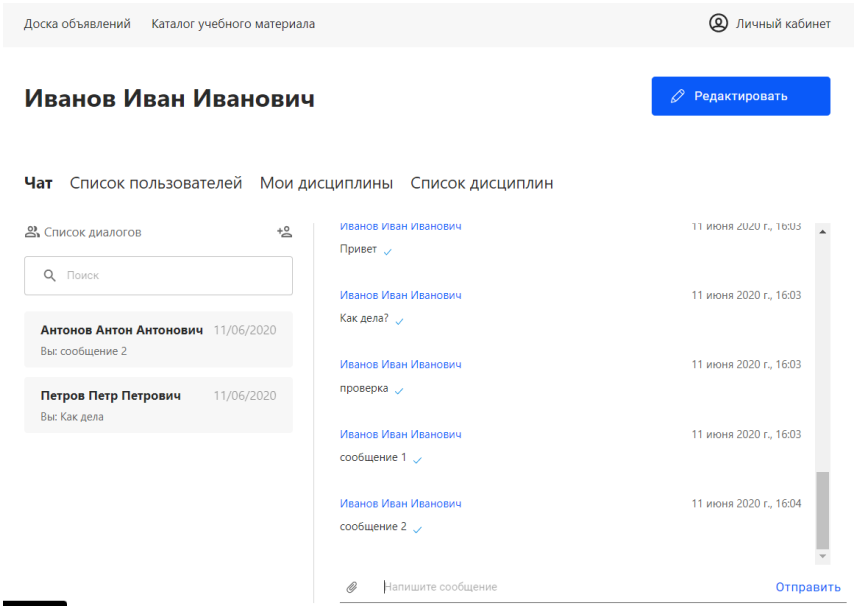

Рисунок 25 – чат

Если пользователь зашел, используя права админа, то ему будет доступен раздел «Список пользователей» в личном кабинете, в котором администратор может посмотреть список всех пользователей, поменять права доступа пользователю или добавить нового пользователя с определенным правом доступа.

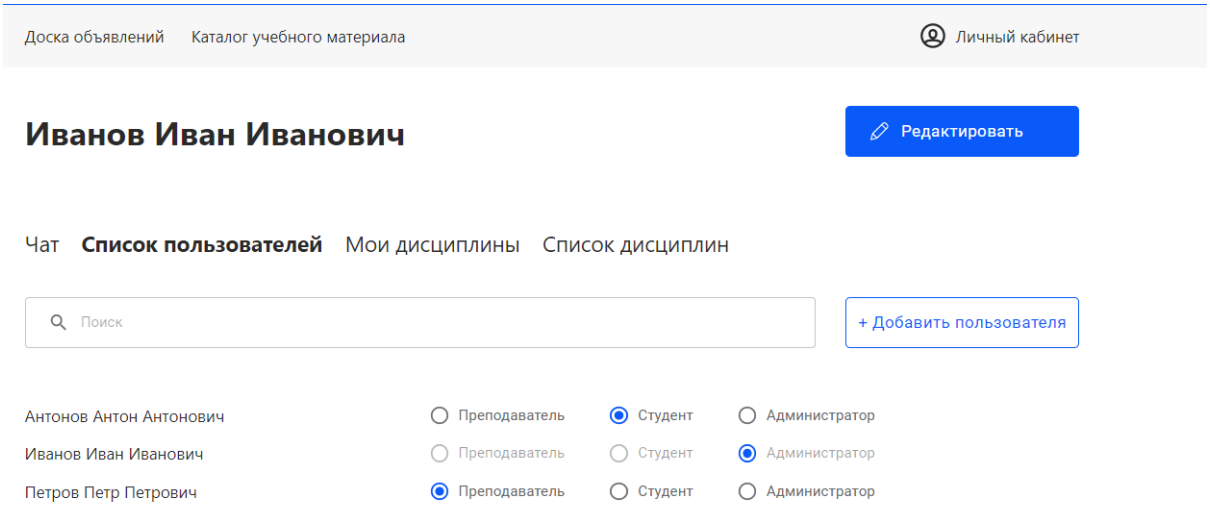

Рисунок 26 – список пользователей

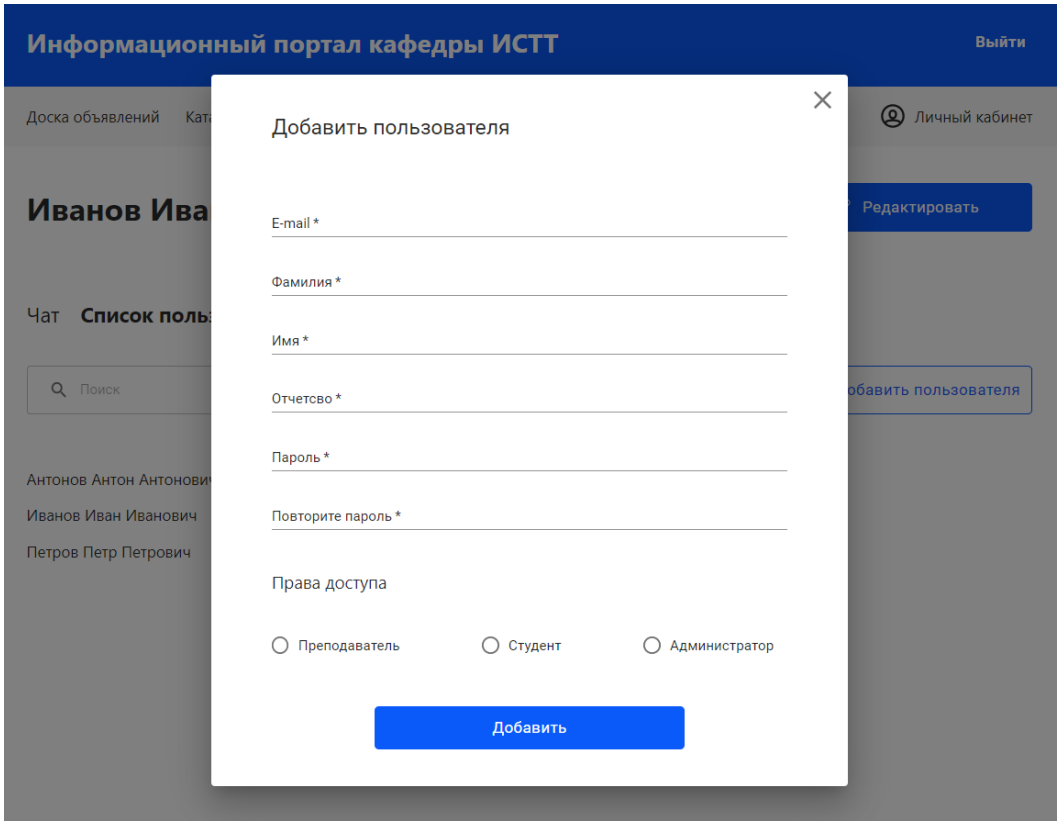

Рисунок 27 – форма добавления нового пользователя

Глава 4. Экономическое обоснование

4.1 Организационный план работы по реализации проекта и расчет сметы

Для определения трудоемкости выполнения данной работы необходимо составить перечень всех этапов, определить трудоёмкость на каждом этапе работы (человеко-дни). Трудоемкость выполнения всей работы складывается из трудоемкости отдельных этапов работы.

Наименование работ	**Трудоемкость, чел./дни**	
	Главный специалист	**Инженер (веб-программист)**
Разработка технического задания	1	3
Разработка методов решения задачи. Выбор языка программирования.	–	3
Изучение литературы	–	2

Разработка системы синхронизации	–	15
Проверка данных на изменение или обновление	–	10
Определение пути найденных файлов	–	10
Поддержка любого типа файлов	–	5
Тестирование и отладка всей системы в целом	1	13
Составление технической документации	1	10
Сдача проекта	1	1
ИТОГО:	4	72

Таблица 1 - Трудоемкость работ по разработке проекта

На основе трудоемкости выполнения работ по разработке системы рассчитываются издержки на оплату труда ее исполнителей, являющиеся одной из основных статей калькуляции себестоимости разработки.

Себестоимость определяется по фактическим затратам, произведенным за счет собственных финансовых средств предприятия. В основе определения лежит перечень выполненных работ и трудоемкость их выполнения.

Калькуляция себестоимости системы осуществляется по следующим статьям: материалы с учетом транспортно-заготовительных расходов, основная и дополнительная заработная плата основных исполнителей работы; отчисления на социальные нужды; расходы на служебные командировки; оплата услуг сторонних организаций, привлекаемых для выполнения данной разработки; прочие прямые затраты; накладные расходы.

Стоимость основных и вспомогательных расходных материалов, необходимых для выполнения разработки, определяется, исходя из величины их расхода, действующих цен и транспортно-заготовительных расходов. Величина транспортно-заготовительных расходов принимается равной 10% стоимости материалов, полуфабрикатов и комплектующих изделий.

Калькуляция расходов по статье "Материалы" приведена в таблице «Материалы».

Материалы	Единица измерения	Количество	Цена, сом	Сумма, сом
Бумага писчая	пачка	1	150	150
Записываемый компакт-диск	шт.	5	20	100
ИТОГО:				250
Транспортно-заготовительные расходы, 10%				25
ВСЕГО:				275

Таблица 2 - Калькуляция расходов по статье "Материалы"

4.2 Расчет сметы затрат на систему

При составлении сметы затрат на систему учитываются:

- стоимость материалов,
- основная заработная плата,
- дополнительная заработная плата,
- отчисления на социальные нужды,
- накладные расходы,
- затраты на машинное время.

Стоимость материалов 275сом.

Основная и дополнительная заработная плата непосредственных исполнителей рассчитывается на основании следующих данных:

- трудоемкость выполнения работ $Т_{СП}$ и $Т_{ИНЖ}$ (по таблице 5.2);
- дневная ставка главного специалиста $Д_{СП}$ = 500 сом;
- дневная ставка инженера $Д_{ИНЖ}$ = 380 сом;
- процент дополнительной заработной платы - 10%;
- процент отчисления на социальные нужды - 27 %**;**
- процент накладных расходов (по данным предприятия) - 15%.

Основная заработная плата исполнителей ($С_{ЗО}$) рассчитывается по формуле:

$С_{ЗО} = Т_{СП} * Д_{СП} + Т_{инж} * Д_{инж}$

где $Т_{СП}$ и $Т_{инж}$ - соответственно, трудоемкость выполнения работ по реализации данной разработки главным специалистом и инженером, чел/дн.

$С_{ЗО} = 4 * 500 + 72 * 380 = 29360$ сом

Дополнительная заработная плата рассчитывается по формуле:

$С_{ЗД} = 29360 * 0,10 = 2936$ сом

Отчисления на социальные нужды: $С_{СН} = (30296) * 0,27 = 8179$ сом

Расходов на служебные командировки нет. Накладные расходы рассчитываются по формуле: $С_{НР} = (30296) * 0,15 = 4544$ сом

На основании вышеприведенных данных, составлена калькуляция себестоимости разработки и сведена в таблицу.

Затраты	***Сумма, сом***
Материалы	275
Основная заработная плата	29360
Дополнительная заработная плата	2936
Отчисления на социальные нужды	8179
Накладные расходы	4544
ИТОГО себестоимость:	45294

Таблица 3 - Калькуляция себестоимости разработки

Заключение

При выполнении дипломного проекта были достигнуты следующие результаты:

- ✓ Составлено техническое задание
- ✓ Разработан дизайн
- ✓ Разработана функциональная модель
- ✓ Разработана клиентская часть веб приложения:
  - Авторизация
  - Страница объявлений
  - Каталог учебных материалов
  - Личный кабинет
  - Чат
  - Список пользователей
  - Добавление новых пользователей
  - Добавление предметов
- ✓ Разработана серверная часть
- ✓ Разработана база данных

Данное приложение пока еще не размещено на хостинге, приложение пока можно запустить на локальной машине.

Список литературы

1. JavaScript-библиотека для создания пользовательских интерфейсов https://ru.reactjs.org/
2. Redux https://redux.js.org/
3. Figma https://ru.wikipedia.org/wiki/Figma
4. Node.js - https://nodejs.org/en/docs/
5. Socket.io - https://socket.io/

Frontend

Index.html

```
<!DOCTYPE html>
<html lang="ru">
  <head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1" />
    <meta name="theme-color" content="#000000" />
    <meta
      name="description"
      content="Web site created using create-react-app"
    />

    <link
href="https://fonts.googleapis.com/css?family=Roboto:400,700&display=swap&subs
et=cyrillic" rel="stylesheet">
    <link rel="manifest" href="%PUBLIC_URL%/manifest.json" />

    <title>Портал кафедры ИСТТ</title>
  </head>
  <body>
    <noscript>You need to enable JavaScript to run this app.</noscript>
    <div id="portal"></div>
  </body>
</html>
```

Index.js

```
import React from 'react';
import ReactDOM from 'react-dom';
import {Provider} from 'react-redux';
import {ConnectedRouter} from 'connected-react-router';
import store, {history} from './store/configureStore';
import App from './App';
import * as serviceWorker from './serviceWorker';

import 'react-notifications/lib/notifications.css';
import './index.css';


const app = (
   <Provider store={store}>
      <ConnectedRouter history={history}>
         <App/>
      </ConnectedRouter>
   </Provider>
);
```

```
    ReactDOM.render(app, document.getElementById('portal'));
serviceWorker.unregister();
```

Routes.js

```
    import React from 'react';
    import {Route, Switch} from 'react-router-dom';
    import Login from './containers/Login/Login';
    import Register from './containers/Register/Register';
    import Board from './containers/Board/Board';
    import Catalog from './containers/Catalog/Catalog';
    import PersonalAccount from './containers/PersonalAccount/PersonalAccount';

    const Routes = () => {
      return (
        <Switch>
          <Route path='/' exact component={Login} />
          <Route path='/register' exact component={Register} />
          <Route path='/main' exact component={Board} />
          <Route path='/catalog' exact component={Catalog} />
          <Route path='/personal-account' component={PersonalAccount} />
          <Route path='/catalog/user/:userId' exact component={Catalog} />
        </Switch>
      );
    };

export default Routes;
```

App.js

```
    import React, {Component, Fragment} from 'react';
    import Routes from './Routes';
    import {NotificationContainer} from 'react-notifications';

    import './App.css';

    class App extends Component {
      render() {
        return (
          <Fragment>
            <NotificationContainer/>
            <Routes user={this.props.user}/>
          </Fragment>
        );
      }
    }
```

```
export default App;
```

configureStore.js

```
import {applyMiddleware, combineReducers, compose, createStore} from 'redux';
import thunkMiddleware from 'redux-thunk';
import {createBrowserHistory} from 'history';
import {connectRouter, routerMiddleware} from 'connected-react-router';
import {loadFromLocalStorage, saveToLocalStorage} from './localStorage';
import axios from '../axiosBase';
import noticeReducers from './reducers/noticeReducers';
import usersReducers from './reducers/usersReducers';
import adminUsersReducers from './reducers/adminUsersReducers';
import subjectsReducers from './reducers/subjectsReducers';
import dialogReducers from './reducers/dialogReducers';
import messageReducers from './reducers/messageReducers';

export const history = createBrowserHistory();

const rootReducer = combineReducers({
  router: connectRouter(history),
  users: usersReducers,
  notices: noticeReducers,
  adminUsers: adminUsersReducers,
  subjects: subjectsReducers,
  dialogs: dialogReducers,
  messages: messageReducers
});

const composeEnhancers =
window.__REDUX_DEVTOOLS_EXTENSION_COMPOSE__ || compose;

const middleware = [
  thunkMiddleware,
  routerMiddleware(history)
];

const enhancers = composeEnhancers(applyMiddleware(...middleware));

const persistedState = loadFromLocalStorage();

const store = createStore(rootReducer, persistedState, enhancers);

store.subscribe(() => {
  saveToLocalStorage({
    users: {
```

```
            user: store.getState().users.user
        }
    });
});

axios.interceptors.request.use(config => {
    try {
        config.headers['Authorization'] = store.getState().users.user.token;
    } catch(e) {
        // do nothing, user is not logged in
    }

    return config
});

export default store;
```

localStorage.js

```
export const saveToLocalStorage = state => {
    try {
        const serializedState = JSON.stringify(state);
        localStorage.setItem('state', serializedState);
    } catch (e) {
        console.log('Could not save state');
    }
};

export const loadFromLocalStorage = () => {
    try {
        const serializedState = localStorage.getItem('state');
        if (serializedState === null) {
            return undefined;
        }

        return JSON.parse(serializedState);
    } catch (e) {
        return undefined;
    }
};
```

adminUsersActions.js

```
import axios from '../../axiosBase';
import {NotificationManager} from 'react-notifications';

export const FETCH_USERS_SUCCESS = 'FETCH_USERS_SUCCESS';
export const FETCH_USER_SUCCESS = 'FETCH_USER_SUCCESS';
```

```
export const PATCH_USER_SUCCESS = 'PATCH_USER_SUCCESS';
export const CREATE_USER_SUCCESS = 'CREATE_USER_SUCCESS';
export const CREATE_USER_FAILURE = 'CREATE_USER_FAILURE';

export const fetchUsersSuccess = adminUsers => ({type:
FETCH_USERS_SUCCESS, adminUsers});
export const fetchUserSuccess = adminUser => ({type: FETCH_USER_SUCCESS,
adminUser});
export const patchUserSuccess = user => ({type: PATCH_USER_SUCCESS, user});
export const createUserSuccess = user => ({type: CREATE_USER_SUCCESS,
user});
export const createUserFailure = error => ({type: CREATE_USER_FAILURE,
error});

export const fetchAdminUsers = () => {
  return dispatch => {
    return axios.get('/admin/users').then(
      response => {
        dispatch(fetchUsersSuccess(response.data));
      }
    );
  };
};

export const fetchAdminUser = (userId) => {
  return dispatch => {
    return axios.get(`/admin/users/${userId}`).then(
      response => {
        dispatch(fetchUserSuccess(response.data))
      }
    )
  }
};

export const createUser = (userData) => {
  return (dispatch) => {
    return axios.post('/admin/users', userData).then(
      () => {
        dispatch(fetchAdminUsers());
        NotificationManager.success('Пользователь создан');
        dispatch(createUserSuccess());
      }, error => {
        if (error.response) {
          dispatch(createUserFailure(error.response.data))
        } else {
          dispatch(createUserFailure({global: 'Нет соединения'}))
```

```
                }
            }
        )
    };
};

export const patchUser = (id, role) => {
    return (dispatch) => {
        return axios.patch(`/admin/users/role_change/${id}`, {id, role: role}).then(
            () => {
                dispatch(patchUserSuccess());
                dispatch(fetchAdminUsers());
                NotificationManager.success('Роль пользователя изменена');
            }
        );
    };
};

export const  assignSubjectToTeacher = (userId, subjectId) =>{
    return dispatch => {
        axios.patch(`/admin/users/${userId}`, { appointSubjects: subjectId}).then(
            () => {
                dispatch(patchUserSuccess());
                NotificationManager.success('Предмет назначен на преподавателя');
            },
            error => {
                if (error.response) {
                    NotificationManager.warning('Выберите преподавателя и предмет');
                }
            }
        )
    }
};
```

dialogActions.js

```
import axios from '../../axiosBase';

export const FETCH_DIALOGS_SUCCESS = 'FETCH_DIALOGS_SUCCESS';
export const FETCH_DIALOG_SUCCESS = 'FETCH_DIALOG_SUCCESS';
export const CREATE_DIALOG_SUCCESS = 'CREATE_DIALOG_SUCCESS';
export const CREATE_DIALOG_FAILURE = 'CREATE_DIALOG_FAILURE';
export const DIALOG_READ_STATUS = 'DIALOG_READ_STATUS';

export const fetchDialogsSuccess = dialogs => ({ type:
FETCH_DIALOGS_SUCCESS, dialogs });
```

```
export const fetchDialogSuccess = currentDialogId => ({ type:
FETCH_DIALOG_SUCCESS, currentDialogId });
export const createDialogSuccess = dialog => ({ type:
CREATE_DIALOG_SUCCESS, dialog });
export const createDialogFailure = error => ({ type: CREATE_DIALOG_FAILURE,
error });

export const updateReadStatus = (userId, dialogId) => ({
  type: DIALOG_READ_STATUS,
  userId,
  dialogId
});

export const fetchDialogs = () => {
  return dispatch => {
    return axios.get(`/dialog`).then(
      response => {
        dispatch(fetchDialogsSuccess(response.data));
      }
    )
  }
};

export const createDialog = dialogData => {
  return dispatch => {
    return axios.post('/dialog', dialogData).then(
      () => {
        dispatch(createDialogSuccess());
      },
      error => {
        if (error.response) {
          dispatch(createDialogFailure(error.response.data));
        } else {
          dispatch(createDialogFailure({ global: 'Нет соединения' }));
        }
      }
    )
  }
};
```

messageActions.js

```
import axios from '../../axiosBase';

export const FETCH_MESSAGES_SUCCESS = 'FETCH_MESSAGES_SUCCESS';
export const CREATE_MESSAGE_SUCCESS =
'CREATE_MESSAGE_SUCCESS';
```

```
export const ADD_MESSAGE = 'ADD_MESSAGE';

export const fetchMessagesSuccess = messages => ({ type:
FETCH_MESSAGES_SUCCESS, messages });
export const createMessageSuccess = message => ({ type:
CREATE_MESSAGE_SUCCESS, message });
export const addMessage = message => (dispatch, getState) => {
  const currentDialogId = getState().dialogs.currentDialogId;
  if (currentDialogId === message.dialog._id) {
    dispatch({
      type: ADD_MESSAGE,
      message
    })
  }
};

export const fetchMessagesByDialogId = currentDialogId => {
  return dispatch => {
    return axios.get(`/messages?dialog=${currentDialogId}`).then(
      response => {
        dispatch(fetchMessagesSuccess(response.data));
      }
    )
  }
};

export const createNewMessage = (messageData) => {
  return dispatch => {
    return axios.post('/messages', messageData).then(() => {
      dispatch(createMessageSuccess);
    })
  }
};
```

noticeActions.js

```
import axios from '../../axiosBase';
import {NotificationManager} from "react-notifications";

export const FETCH_NOTICE_SUCCESS = 'FETCH_NOTICE_SUCCESS';
export const CREATE_NOTICE_SUCCESS = 'CREATE_NOTICE_SUCCESS';
export const CREATE_NOTICE_FAILURE = 'CREATE_NOTICE_FAILURE';
export const SET_CURRENT_PAGE = 'SET_CURRENT_PAGE';
export const SET_TOTAL_NOTICES = 'SET_TOTAL_NOTICES';

export const fetchNoticeSuccess = notices => ({type: FETCH_NOTICE_SUCCESS,
notices});
```

```
export const createNoticeSuccess = notice => ({type:
CREATE_NOTICE_SUCCESS, notice});
export const createNoticeFailure = error => ({type: CREATE_NOTICE_FAILURE,
error});
export const setCurrentPage = currentPage => ({type: SET_CURRENT_PAGE,
currentPage});
export const setTotalNotices = totalNotices => ({type: SET_TOTAL_NOTICES,
totalNotices});

export const fetchNotice = (pageNumber, noticePerPage) => {
  return dispatch => {
    return axios.get(`/notice?page=${pageNumber}&limit=${noticePerPage}`).then(
      response => {
        dispatch(fetchNoticeSuccess(response.data.notices));
        dispatch(setTotalNotices(response.data.totalNotices));
      }
    );
  };
};

export const createNotice = (noticeData, pageNumber, noticePerPage) => {
  return (dispatch, getState) => {
    const token = getState().users.user.token;
    return axios.post('/notice', noticeData, {headers: {'Authorization': token}}).then(
      () => {
        dispatch(fetchNotice(pageNumber, noticePerPage));
        NotificationManager.success('Объявление создано');
        dispatch(createNoticeSuccess());
      }, error => {
        if (error.response) {
          console.log('Ошибка', error.response);
          dispatch(createNoticeFailure(error.response.data));
        } else {
          dispatch(createNoticeFailure({global: 'Нет соединения'}));
        }
      }
    )
  };
};
```

subjectActions.js

```
import axios from '../../axiosBase';
import {push} from 'connected-react-router';
import {NotificationManager} from 'react-notifications';

export const FETCH_SUBJECTS_SUCCESS = 'FETCH_SUBJECTS_SUCCESS';
```

```
export const CREATE_SUBJECT_SUCCESS = 'CREATE_SUBJECT_SUCCESS';
export const CREATE_SUBJECT_FAILURE = 'CREATE_SUBJECT_FAILURE';
export const PATCH_SUBJECT_SUCCESS = 'PATCH_SUBJECT_SUCCESS';

export const fetchSubjectsSuccess = subjects => ({type:
FETCH_SUBJECTS_SUCCESS, subjects});
export const createSubjectSuccess = subject => ({type:
CREATE_SUBJECT_SUCCESS, subject});
export const createSubjectFailure = error => ({type:
CREATE_SUBJECT_FAILURE, error});
export const patchSubjectSuccess = subject => ({type:
PATCH_SUBJECT_SUCCESS, subject});

export const fetchSubjects = () => {
   return dispatch => {
      return axios.get('/subject').then(
         response => {
            dispatch(fetchSubjectsSuccess(response.data));
         }
      )
   }
};

export const createSubject = subjectData => {
   return dispatch => {
      return axios.post('/subject', subjectData).then(
         () => {
            dispatch(fetchSubjects());
            dispatch(createSubjectSuccess());
            NotificationManager.success('Новая дисциплина создана');
         },
         error => {
            if (error.response) {
               dispatch(createSubjectFailure(error.response.data));
            } else {
               dispatch(createSubjectFailure({global: 'Нет соединения'}));
            }
         }
      )
   }
};

export const  addFileToSubject = (subjectId, subjectData) =>{
   return dispatch => {
      axios.patch(`/subject/${subjectId}`, subjectData).then(
         () => {
```

```
                dispatch(fetchSubjects());
                dispatch(patchSubjectSuccess());
                NotificationManager.success('Файл добавлен');
                dispatch(push('/catalog'));
            },
            error => {
                if (error.response) {
                    NotificationManager.warning('Файлов должно быть от 1 до 5');
                }
            }
        )
    }
};
```

userActions.js

```
import {push} from 'connected-react-router';
import {NotificationManager}  from 'react-notifications';
import axios from '../../axiosBase';

export const REGISTER_USER_SUCCESS = 'REGISTER_USER_SUCCESS';
export const REGISTER_USER_FAILURE = 'REGISTER_USER_FAILURE';

export const LOGIN_USER_SUCCESS = 'LOGIN_USER_SUCCESS';
export const LOGIN_USER_FAILURE = 'LOGIN_USER_FAILURE';

export const USER_INFO = 'USER_INFO';

export const LOGOUT_USER = 'LOGOUT_USER';

export const EDIT_USER_SUCCESS = 'EDIT_USER';
export const EDIT_USER_FAILURE = 'EDIT_USER_FAILURE';


const registerUserSuccess = (user) => ({type: REGISTER_USER_SUCCESS, user});
const registerUserFailure = error => ({type: REGISTER_USER_FAILURE, error});

const loginUserSuccess = user => ({type: LOGIN_USER_SUCCESS, user});
const loginUserFailure = error => ({type: LOGIN_USER_FAILURE, error});

const getUserInfo = user => ({type: USER_INFO, user});

export const editUserSuccess = user => ({type: EDIT_USER_SUCCESS, user});
export const editUserFailure = error => ({type: EDIT_USER_FAILURE, error});

export const fetchUserInfo = () => {
    return dispatch => {
```

```
        return axios.get('/users').then(
            response => {
                dispatch(getUserInfo(response.data));
            }
        )
    }
};

export const logoutUser = () => {
    return (dispatch, getState) => {
        const token = getState().users.user.token;
        const config = {headers: {'Authorization': token}};

        return axios.delete('/users/sessions', config).then(
            () => {
                dispatch({type: LOGOUT_USER});
                NotificationManager.success('Вы вышли из системы!');
                dispatch(push('/'));
            },
            error => {
                NotificationManager.error('Не возможно авторизоваться, попробуйте
позже!')
            }
        )
    }
};

export const registerUser = userData => {
    return dispatch => {
        return axios.post('/users', userData.user).then(
            response => {
                dispatch(registerUserSuccess(response.data.user));
                NotificationManager.success('Регистрация завершена');
                dispatch(push('/main'))
            },
            error => {
                if (error.response && error.response.data) {
                    dispatch(registerUserFailure(error.response.data))
                } else {
                    dispatch(registerUserFailure({global: 'Нет соединения'}))
                }
            }
        )
    }
};
```

```
export const loginUser = userData => {
  return dispatch => {
    return axios.post('/users/sessions', userData).then(
      response => {
        dispatch(loginUserSuccess(response.data.user));
        NotificationManager.success('Вы вошли в систему!');
        dispatch(push('/main'));
      }, error => {
        if (error.response) {
          dispatch(loginUserFailure(error.response.data))
        } else {
          dispatch(loginUserFailure({global: 'Нет соединения'}))
        }
      }
    )
  }
};

export const editUser = (userData) => {
  return (dispatch, getState) => {
    const token = getState().users.user.token;
    const config = {headers: {'Authorization': token}};
    return axios.put('/users/sessions', userData, config).then(
      () => {
        dispatch(editUserSuccess());
        NotificationManager.success('Данные пользователя изменены');
        dispatch(fetchUserInfo());
      }, error => {
        if (error.response) {
          dispatch(editUserFailure(error.response.data))
        } else {
          dispatch(editUserFailure({global: 'Нет соединения'}))
        }
      }
    );
  };
};
```

adminUsersReducers.js

```
import {
  CREATE_USER_FAILURE, FETCH_USER_SUCCESS,
  FETCH_USERS_SUCCESS
} from '../actions/adminUsersActions';

const initialState = {
  adminUsers: [],
```

```
    adminUser: [],
    addUserError: null
};

const adminReducer = (state = initialState, action) => {
    switch (action.type) {
        case FETCH_USERS_SUCCESS:
            return {...state, adminUsers: action.adminUsers};
        case FETCH_USER_SUCCESS:
            return {...state, adminUser: action.adminUser};
        case CREATE_USER_FAILURE:
            return {...state, addUserError: action.error};
        default:
            return state;
    }
};

export default adminReducer;
```

dialogReducers.js

```
import {
    CREATE_DIALOG_FAILURE,
    FETCH_DIALOG_SUCCESS,
    FETCH_DIALOGS_SUCCESS,
    DIALOG_READ_STATUS
} from '../actions/dialogActions';

const initialState = {
    dialogs: [],
    dialogError: null,
    currentDialogId: window.location.pathname.split('personal-account/dialog/')[1],
};

const dialogReducer = (state = initialState, action) => {
    switch (action.type) {
        case FETCH_DIALOGS_SUCCESS:
            return { ...state, dialogs: action.dialogs };
        case FETCH_DIALOG_SUCCESS:
            return { ...state, currentDialogId: action.currentDialogId };
        case DIALOG_READ_STATUS:
            return {
                ...state,
                dialogs: state.dialogs.map(dialog => {
                    if (dialog._id === action.dialogId) {
                        dialog.lastMessage.read = true;
                    }
```

```
                return dialog;
            })
        };
    case CREATE_DIALOG_FAILURE:
        return { ...state, dialogError: action.error };
    default:
        return state;
  }
};

export default dialogReducer;
```

messageReducers.js

```
import {FETCH_MESSAGES_SUCCESS, ADD_MESSAGE} from
'../actions/messageActions';
import {DIALOG_READ_STATUS} from '../actions/dialogActions';

const initialState = {
  messages: []
};

const messageReducer = (state = initialState, action) => {
  switch (action.type) {
    case DIALOG_READ_STATUS:
      return {
        ...state,
        messages: state.messages.map(message => {
          if (message._id === action.dialogId) {
            message.read = true;
          }
          return message;
        })
      };
    case ADD_MESSAGE:
      return { ...state, messages: [...state.messages, action.message] };
    case FETCH_MESSAGES_SUCCESS:
      return { ...state, messages: action.messages };
    default:
      return state;
  }
};

export default messageReducer;
```

noticeReducers.js

```
import {
   CREATE_NOTICE_FAILURE,
   FETCH_NOTICE_SUCCESS,
   SET_CURRENT_PAGE,
   SET_TOTAL_NOTICES
} from '../actions/noticeActions';

const initialState = {
   notices: [],
   noticeError: null,
   noticePerPage: 5,
   currentPage: 1,
   totalNotices: 0
};

const noticeReducers = (state = initialState, action) => {
   switch (action.type) {
      case FETCH_NOTICE_SUCCESS:
         return {
            ...state,
            notices: action.notices};
      case CREATE_NOTICE_FAILURE:
         return {...state, noticeError: action.error};
      case SET_CURRENT_PAGE:
         return {...state, currentPage: action.currentPage};
      case SET_TOTAL_NOTICES:
         return {...state, totalNotices: action.totalNotices};
      default:
         return state;
   }
};

export default noticeReducers;
```

subjectsReducers.js

```
import {CREATE_SUBJECT_FAILURE, FETCH_SUBJECTS_SUCCESS} from
'../actions/subjectActions';

const initialState = {
   subjects: [],
   subjectError: null
};

const subjectsReducers = (state = initialState, action) => {
   switch (action.type) {
```

```
            case FETCH_SUBJECTS_SUCCESS:
                return {...state, subjects: action.subjects};
            case CREATE_SUBJECT_FAILURE:
                return {...state, subjectError: action.error};
            default:
                return state;
        }
    };

export default  subjectsReducers;
```

usersReducers.js

```
    import {
        EDIT_USER_FAILURE,
        LOGIN_USER_FAILURE,
        LOGIN_USER_SUCCESS, LOGOUT_USER,
        REGISTER_USER_FAILURE,
        REGISTER_USER_SUCCESS, USER_INFO
    } from '../actions/userActions';

    const initialState = {
        registerError: null,
        loginError: null,
        editUserError: null,
        user: null
    };

    const usersReducers = (state = initialState, action) => {
        switch (action.type) {
            case REGISTER_USER_SUCCESS:
                return {...state, registerError: null, user: action.user};
            case REGISTER_USER_FAILURE:
                return {...state, registerError: action.error};
            case LOGIN_USER_SUCCESS:
                return {...state, user: action.user, loginError: null};
            case LOGIN_USER_FAILURE:
                return {...state, loginError: action.error};
            case USER_INFO:
                return {...state, user: action.user};
            case EDIT_USER_FAILURE:
                return {...state, editUserError: action.error};
            case LOGOUT_USER:
                return {...state, user: null};
            default:
                return state;
        }
```

```
};

export default usersReducers;
```

Board.js

```
import React, {Component, Fragment} from 'react';
import Moment from 'react-moment';
import 'moment/locale/ru';
import {connect} from 'react-redux';
import {apiURL} from '../../constants';
import {withRouter} from 'react-router-dom';

import Button from '@material-ui/core/Button';
import DialogTitle from '@material-ui/core/DialogTitle';
import Dialog from '@material-ui/core/Dialog';
import IconButton from '@material-ui/core/IconButton';
import DialogContent from '@material-ui/core/DialogContent';
import TextField from '@material-ui/core/TextField';
import CloseIcon from '@material-ui/icons/Close';
import AttachFileIcon from '@material-ui/icons/AttachFile';

import Toolbar from '../../components/UI/Toolbar';
import {createNotice, fetchNotice, setCurrentPage} from
'../../store/actions/noticeActions';
import {logoutUser} from '../../store/actions/userActions';
import PaginationBlock from '../../components/Pagination/Pagination';

import './Board.css';

class Board extends Component {
  state = {
    title: '',
    description: '',
    noticeFile: [],
    fileName: [],
    addNoticeModal: false,
  };

  componentDidMount() {
    this.props.fetchNotice(this.props.currentPage, this.props.noticePerPage);
  }

  inputChangeHandler = event => {
    this.setState({
      [event.target.name]: event.target.value
    });
```

```
    };

    openNoticeModal = () => {
        this.setState({addNoticeModal: true});
    };
    closeNoticeModal = () => {
        this.setState({addNoticeModal: false, title: '', description: '', noticeFile: [],
fileName: []});
    };

    noticeChangeHandler = event => {
        event.preventDefault();

        let files = Array.from(event.target.files);

        files.forEach((file) => {
            let reader = new FileReader();
            reader.onloadend = () => {
                this.setState({
                    noticeFile: [...this.state.noticeFile, file],
                    fileName: [...this.state.fileName, file.name]
                });
            };
            reader.readAsDataURL(file);
        });
    };

    submitFormHandler = event => {
        event.preventDefault();
        const formData = new FormData();

        for (let i = 0; i < this.state.noticeFile.length; i++) {
            formData.append('noticeFile', this.state.noticeFile[i]);
        }

        Object.keys(this.state).forEach(key => {
            if (!['noticeFile'].includes(key)) {
                formData.append(key, this.state[key]);
            }
        });

        this.props.createdNotice(formData, this.props.currentPage,
this.props.noticePerPage);

        if (this.state.title !== '') {
            this.closeNoticeModal();
```

```
    }
  };

  onPageChanged = pageNumber => {
    this.props.setCurrentPage(pageNumber);
    this.props.fetchNotice(pageNumber, this.props.noticePerPage);
  };

  getFieldError = fieldName => {
    return this.props.error && this.props.error.errors &&
this.props.error.errors[fieldName]
      && (this.props.error.errors[fieldName].message)
  };

  render() {
    let noticeCard = null;
    if (this.props.notices) {
      noticeCard = this.props.notices.map(notice => {
        return (
          <div className='board-post' key={notice._id}>
            <div className='board-post__top'>
              <span>{notice.user.secondName} {notice.user.firstName}</span>
              <span><Moment format='LLL'
locale='ru'>{notice.dateTime}</Moment></span>
            </div>
            <h4 className='board-post__title'>{notice.title}</h4>
            <p className='board-post__desc'>{notice.description}</p>
            <div className='board-post-file'>
              {
                notice.noticeFile.map(file => {
                  return (
                    <a href={apiURL + '/uploads/notice/' + file}
key={file}>{file}</a>
                  )
                })
              }
            </div>
          </div>
        )
      })
    }
    return (
      <Fragment>
        <Toolbar
          user={this.props.user}
          logout={this.props.logoutUser}
```

```
            />
            <div className='main-board'>
               <div className='container'>
                  <div className='top'>
                     <h4 className='top__title'>Доска</h4>
                     {
                        this.props.user && this.props.user.role === 'admin' &&
                        <Button className='top__btn common-btn'
onClick={this.openNoticeModal}>+ Добавить объявление</Button>
                     }
                  </div>
                  <Dialog onClose={this.closeNoticeModal}
open={this.state.addNoticeModal} className='add-user-modal'>
                     <DialogTitle onClose={this.closeNoticeModal} className='header-
modal'>
                        Добавить объявление
                        <IconButton aria-label='delete' className='close-modal'
onClick={this.closeNoticeModal}>
                           <CloseIcon fontSize='large' />
                        </IconButton>
                     </DialogTitle>
                     <DialogContent>
                        <form className='add-notice-form'
onSubmit={this.submitFormHandler}>
                           <TextField
                              label='Заголовок *'
                              type='text'
                              value={this.state.title}
                              name="title"
                              onChange={this.inputChangeHandler}
                              helperText={this.getFieldError('title')}
                              variant='outlined'
                           />
                           <TextField
                              label='Текст объявления'
                              type='textarea'
                              value={this.state.description}
                              name='description'
                              onChange={this.inputChangeHandler}
                              helperText={this.getFieldError('description')}
                              variant='outlined'
                              multiline
                              rows={6}
                              rowsMax={6}
                           />
                           <div className='file-preview'>
```

```
                        {
                            this.state.fileName.map((file) => {
                                return (
                                    <span key={file}>{file}</span>
                                )
                            })
                        }
                    </div>
                    <input
                        accept='noticeFile/*'
                        type='file'
                        multiple
                        id='upload-file'
                        onChange={this.noticeChangeHandler}
                        style={{ display: 'none', }}
                    />
                    {
                        this.state.fileName.length === 0 ?
                            <label htmlFor='upload-file' className='upload-file'>
                                <Button
                                    component='span'
                                    startIcon={<AttachFileIcon />}
                                >
                                    Прикрепить файл
                                </Button>
                            </label> :
                            <label htmlFor='upload-file' className='upload-file
catalog-upload-file add-new-file'>
                                <Button
                                    component='span'
                                >
                                    + Добавить файл
                                </Button>
                            </label>
                    }

                    <div className='form-button-wrap'>
                        <Button autoFocus type='submit' color='primary'>
                            Сохранить
                        </Button>
                    </div>
                </form>
            </DialogContent>
        </Dialog>
        {
            this.props.notices.length !== 0 ?
```

```
                        <div className='board-posts'>
                            {noticeCard}
                        </div> : <div className='board-posts-empty'>Список
объявлейний пуст</div>
                    }
                    <PaginationBlock
                        totalNotices = {this.props.totalNotices}
                        noticePerPage = {this.props.noticePerPage}
                        currentPage = {this.props.currentPage}
                        onPageChanged = {this.onPageChanged}
                    />
                </div>
            </div>
        </Fragment>
    );
  }
}

const mapStateToProps = state => ({
  user: state.users.user,
  error: state.notices.noticeError,
  notices: state.notices.notices,
  noticePerPage: state.notices.noticePerPage,
  currentPage: state.notices.currentPage,
  totalNotices: state.notices.totalNotices
});

const mapDispatchToProps = dispatch => ({
  logoutUser: () => dispatch(logoutUser()),
  createdNotice: (noticeData, pageNumber, noticePerPage) =>
dispatch(createNotice(noticeData, pageNumber, noticePerPage)),
  fetchNotice: (pageNumber, noticePerPage) => dispatch(fetchNotice(pageNumber,
noticePerPage)),
  setCurrentPage: pageNumber => dispatch (setCurrentPage(pageNumber)),
});

export default withRouter(connect(mapStateToProps, mapDispatchToProps)(Board));
```

Catalog.js

```
import React, {Component, Fragment} from 'react';

import {connect} from 'react-redux';
import {withRouter} from 'react-router-dom';

import Button from '@material-ui/core/Button';
import Dialog from '@material-ui/core/Dialog';
```

```
import DialogTitle from '@material-ui/core/DialogTitle';
import IconButton from '@material-ui/core/IconButton';
import CloseIcon from '@material-ui/icons/Close';
import DialogContent from '@material-ui/core/DialogContent';
import FormControl from '@material-ui/core/FormControl';
import Select from '@material-ui/core/Select';
import AttachFileIcon from '@material-ui/icons/AttachFile';

import Toolbar from '../../components/UI/Toolbar';
import {logoutUser} from '../../store/actions/userActions';
import {addFileToSubject, fetchSubjects} from '../../store/actions/subjectActions';
import {fetchAdminUser, fetchAdminUsers} from
'../../store/actions/adminUsersActions';

import './Catalog.css';
import '../Board/Board.css';
import {apiURL} from '../../constants';

class Catalog extends Component {
    state = {
        subject: '',
        materials: [],
        fileName: [],
        addCatalogModal: false,
        userId: '',
    };

    componentDidMount() {
        this.props.onFetchSubjects();
        this.props.onFetchUsers();
    }

    inputChangeHandler = event => {
        this.setState({
            [event.target.name]: event.target.value
        })
    };

    subjectFilesHandler = event => {
        event.preventDefault();

        let files = Array.from(event.target.files);

        files.forEach((file) => {
            let reader = new FileReader();
            reader.onloadend = () => {
```

```
            this.setState({
                materials: [...this.state.materials, file],
                fileName: [...this.state.fileName, file.name]
            });
        };
        reader.readAsDataURL(file);
    });
};

openCatalogModal = () => {
    this.setState({addCatalogModal: true});
};

closeCatalogModal = () => {
    this.setState({addCatalogModal: false, subject: '', materials: [], fileName: []});
};

getUserDataById = async (event) => {
    event.preventDefault();
     await this.setState({userId: event.target.id});

    await this.props.onFetchUser(this.state.userId);
};

submitFormHandler = event => {
    event.preventDefault();
    const formData = new FormData();

    for (let i = 0; i < this.state.materials.length; i++) {
        formData.append('materials', this.state.materials[i]);
    }

    this.props.addFileToSubject(this.state.subject, formData);

    if (this.state.fileName.length !== 0) {
        this.closeCatalogModal();
    }
};

render() {
    let subjectItem = null;
    if (this.props.adminUser.appointSubjects) {
        subjectItem = this.props.adminUser.appointSubjects.map(subject => {
            return (
                <div className='right-column-subject' key={subject._id}>
                    <h5 className='right-column-subject__title'>{subject.subject}</h5>
```

```
                {
                    subject.materials.map(file => {
                        return (
                            <a className='right-column-subject__name' href={apiURL +
'/uploads/subject/' + file} key={file}>{file}</a>
                        )
                    })
                }
            </div>
        )
    })
}
return (
    <Fragment>
        <Toolbar
            user={this.props.user}
            logout={this.props.logoutUser}
        />
        <div className='main-catalog'>
            <div className='container'>
                <div className='top'>
                    <h4 className='top__title'>Каталог учебных материалов</h4>
                    <Button className='top__btn common-btn'
onClick={this.openCatalogModal}>+ Добавить материал</Button>
                </div>
                <Dialog onClose={this.closeCatalogModal}
open={this.state.addCatalogModal} className='add-user-modal'>
                <DialogTitle onClose={this.closeCatalogModal} className='header-
modal'>
                    Добавить материал
                    <IconButton aria-label='delete' className='close-modal'
onClick={this.closeCatalogModal}>
                        <CloseIcon fontSize='large' />
                    </IconButton>
                </DialogTitle>
                <DialogContent>
                    <form className='add-notice-form'
onSubmit={this.submitFormHandler}>
                        <FormControl variant='outlined'>
                            <Select
                                native
                                value={this.state.subject}
                                onChange={this.inputChangeHandler}
                                inputProps={{
                                    name: 'subject',
                                }}
```

```
                        >
                            <option value='' disabled>Предмет</option>
                            {
                                this.props.subjects.map(subject => {
                                    return (
                                        <option value={subject._id}
key={subject._id}>{subject.subject}</option>
                                    )
                                })
                            }
                        </Select>
                    </FormControl>
                    <div className='file-preview'>
                        {
                            this.state.fileName.map((file) => {
                                return (
                                    <span key={file}>{file}</span>
                                )
                            })
                        }
                    </div>
                    <input
                        accept='file/*'
                        type='file'
                        id='upload-file'
                        onChange={this.subjectFilesHandler}
                        style={{ display: 'none', }}
                        multiple
                    />
                    {
                        this.state.fileName.length === 0 ?
                            <label htmlFor='upload-file' className='upload-file catalog-
upload-file'>
                                <Button
                                    component='span'
                                    startIcon={<AttachFileIcon />}
                                >
                                    Прикрепить файл
                                </Button>
                            </label> :
                            <label htmlFor='upload-file' className='upload-file catalog-
upload-file add-new-file'>
                                <Button
                                    component='span'
                                >
                                    + Добавить файл
```

```
                                </Button>
                            </label>
                        }
                        <span className='notification-text'>Максимальное число
файлов для загрузки равно 5</span>
                        <div className='form-button-wrap'>
                            <Button autoFocus color='primary' type='submit'>
                                Сохранить
                            </Button>
                        </div>
                    </form>
                </DialogContent>
            </Dialog>
                <div className='catalog-block'>
                    <div className='left-column'>
                        <h5 className='left-column__title'>Преподаватели</h5>
                        <ul className='teacher-list'>
                            {
                                this.props.adminUsers
                                    .filter(user => user.role === 'teacher' || user.role ===
'admin')
                                    .map(user => {
                                        return (
                                            <li className='teacher-list__item' key={user._id}>
                                                <span id={user._id} className='teacher-
list__link' onClick={this.getUserDataById}>{user.secondName} {user.firstName}
{user.lastName}</span>
                                            </li>
                                        )
                                    })
                            }
                        </ul>
                    </div>
                    <div className='right-column'>
                        {subjectItem}
                    </div>
                </div>
            </div>
        </Fragment>
    );
  }
}

const mapStateToProps = state => ({
  user: state.users.user,
```

```
    subjects: state.subjects.subjects,
    adminUsers: state.adminUsers.adminUsers,
    adminUser: state.adminUsers.adminUser,
});

const mapDispatchToProps = dispatch => ({
    logoutUser: () => dispatch(logoutUser()),
    onFetchSubjects: () => dispatch(fetchSubjects()),
    onFetchUsers: () => dispatch(fetchAdminUsers()),
    onFetchUser: userId => dispatch(fetchAdminUser(userId)),
    addFileToSubject: (subjectId, subjectData) =>
dispatch(addFileToSubject(subjectId, subjectData)),
});

export default withRouter(connect(mapStateToProps, mapDispatchToProps)(Catalog));
```

Chat.js

```
import React, { useEffect } from 'react';
import { withRouter } from 'react-router';
import { connect } from 'react-redux';

import Messages from '../Messages/Messages';
import Dialogs from '../Dialogs/Dialogs';
import ChatInput from '../ChatInput/ChatInput';
import SideBar from '../Sidebar/SideBar';
import { fetchDialogSuccess } from '../../store/actions/dialogActions';

import './Chat.css'

const Chat = props => {
  const { fetchDialogSuccess, user } = props;

  useEffect(() => {
    const { pathname } = props.location;
    const currentDialogId = pathname.split('/personal-account/dialog/')[1];
    fetchDialogSuccess(currentDialogId);
  }, [props.location, fetchDialogSuccess]);

  return (
    <section className='personal-chat'>
      <div className='chat'>
        <div className='chat__sidebar'>
          <SideBar />
          <Dialogs />
        </div>
        {user && (
```

```
            <div className='chat__dialog'>
              <Messages />
              <ChatInput />
            </div>
          )}
        </div>
      </section>
    );
  };

  const mapStateToProps = state => ({
    user: state.users.user,
  });

  const mapDispatchToProps = dispatch => ({
    fetchDialogSuccess: currentDialogId =>
  dispatch(fetchDialogSuccess(currentDialogId))
  });

export default withRouter(connect(mapStateToProps, mapDispatchToProps)(Chat));
```

Dialogs.js

```
  import React, { useState, useEffect } from 'react';
  import { connect } from 'react-redux';
  import { withRouter } from 'react-router-dom';

  import {fetchDialogs, updateReadStatus} from '../../store/actions/dialogActions';
  import DialogItems from '../../components/DialogItems/DialogItems';
  import socket from '../../core/socket';

  const Dialogs = ({ fetchDialogs, updateReadStatus, currentDialogId, dialogs, user })
  => {
    const [inputValue, setValue] = useState('');
    const [filteredItems, setFilteredItems] = useState(Array.from(dialogs));

    const onChangeInput = (value = '') => {
      setFilteredItems(
        dialogs.filter(
          dialog =>
            dialog.author.fullName.toLowerCase().indexOf(value.toLowerCase()) >= 0 ||
            dialog.partner.fullName.toLowerCase().indexOf(value.toLowerCase()) >= 0,
        ),
      );
      setValue(value);
    };
```

```
  useEffect(() => {
    if (dialogs.length) {
      onChangeInput();
    }
  }, [dialogs]); // eslint-disable-line react-hooks/exhaustive-deps

  useEffect(() => {
    fetchDialogs();

    socket.on('SERVER:DIALOG_CREATED', fetchDialogs);
    socket.on('SERVER:NEW_MESSAGE', fetchDialogs);
    socket.on('SERVER:MESSAGES_READ', updateReadStatus);
    return () => {
      socket.removeListener('SERVER:DIALOG_CREATED', fetchDialogs);
      socket.removeListener('SERVER:NEW_MESSAGE', fetchDialogs);
    };
  }, [fetchDialogs, updateReadStatus]);

  return (
    <DialogItems
      user={user}
      dialogs={filteredItems}
      onSearch={onChangeInput}
      inputValue={inputValue}
      currentDialogId={currentDialogId}
    />
  );
};

const mapStateToProps = state => ({
  user: state.users.user,
  dialogs: state.dialogs.dialogs,
  currentDialogId: state.dialogs.currentDialogId
});

const mapDispatchToProps = dispatch => ({
  fetchDialogs: () => dispatch(fetchDialogs()),
  updateReadStatus : (userId, dialogId) => dispatch(updateReadStatus(userId,
dialogId))
});

export default withRouter(connect(mapStateToProps, mapDispatchToProps)(Dialogs));
```

Login.js

```
import React, {Component} from 'react';
import {Link as RouterLink} from 'react-router-dom';
import TextField from '@material-ui/core/TextField';
import Link from '@material-ui/core/Link';
import {connect} from 'react-redux';
import {loginUser} from '../../store/actions/userActions';
import Alert from '@material-ui/lab/Alert';

import './Login.css'

class Login extends Component {
   state = {
      email: '',
      password: '',
   };

   inputChangeHandler = event => {
      this.setState({
         [event.target.name]: event.target.value
      })
   };

   submitFormHandler = event => {
      event.preventDefault();
      this.props.loginUser({...this.state})
   };

   render() {
      return (
         <div className='main-backdrop'>
            <div className='auth-wrap'>
               <div className='auth-title'>
                  <h4 className='title'>Информационный портал кафедры
ИСТТ</h4>
               </div>
               {this.props.error &&  (
                  <Alert severity='error'>{this.props.error.error ||
this.props.error.global}</Alert>
               )}
               <div className='auth-nav'>
                  <nav className='auth-menu'>
                     <Link component={RouterLink} to='/' className='auth-
menu__link auth-menu__link-active'>Вход</Link>
                     <Link component={RouterLink} to='/register' className='auth-
menu__link'>Регистрация</Link>
```

```
                    </nav>
                </div>
                <form className='auth-form' onSubmit={this.submitFormHandler}>
                    <TextField
                        label='E-mail'
                        type='email'
                        name='email'
                        value={this.state.email}
                        onChange={this.inputChangeHandler}
                        className='auth-form__input'
                    />
                    <TextField
                        label='Пароль'
                        type='password'
                        name='password'
                        value={this.state.password}
                        onChange={this.inputChangeHandler}
                        className='auth-form__input'
                    />
                    <button className='btn auth-login'>Войти</button>
                </form>
            </div>
        </div>
    );
  }
}

const mapStateToProps = state => ({
  error: state.users.loginError
});

const mapDispatchToProps = dispatch => ({
  loginUser: userData => dispatch(loginUser(userData))
});

export default connect(mapStateToProps, mapDispatchToProps)(Login);
```

Messages.js

```
import React, { useEffect, useRef } from 'react';
import { connect } from 'react-redux';
import { withRouter } from 'react-router-dom';

import { fetchMessagesByDialogId, addMessage } from
'../../store/actions/messageActions';
import MessagesItems from '../../components/MessageItems/MessageItems';
import socket from '../../core/socket';
```

```
import find from 'lodash/find';

const Messages = ({
  fetchMessagesByDialogId,
  currentDialog,
  updateReadStatus,
  currentDialogId,
  messages,
  userId,
  addMessage,
  user }) => {

  const messagesRef = useRef(null);

  const onNewMessage = (message) => {
    addMessage(message);
  };

  useEffect(() => {
    if (currentDialog) {
      fetchMessagesByDialogId(currentDialog._id);
    }

    socket.on('SERVER:NEW_MESSAGE', onNewMessage);

    return () => socket.removeListener('SERVER:NEW_MESSAGE',
onNewMessage);
  }, [fetchMessagesByDialogId, currentDialog]); // eslint-disable-line react-
hooks/exhaustive-deps

  useEffect(() => {
    if (messagesRef.current !== null) {
      messagesRef.current.scrollTo(0, 999999)
    }
  }, [messages]);

  if (!currentDialog) {
    return <div>Откройте диалог</div>
  }

  return (
    <MessagesItems
      user={user}
      blockRef={messagesRef}
      messages={messages}
    />
```

```
  );
};

const mapStateToProps = state => ({
  user: state.users.user,
  currentDialog: find(state.dialogs.dialogs, { _id: state.dialogs.currentDialogId }),
  messages: state.messages.messages,
});

const mapDispatchToProps = dispatch => ({
  fetchMessagesByDialogId: (currentDialogId) =>
dispatch(fetchMessagesByDialogId(currentDialogId)),
  addMessage: (message) => dispatch(addMessage(message)),

});

export default withRouter(connect(mapStateToProps, mapDispatchToProps)(Messages));
```

PersonalAccount.js

```
import React, {Component, Fragment} from 'react';
import {connect} from 'react-redux';
import {withRouter} from 'react-router-dom';
import {NotificationManager} from 'react-notifications';

import {Tab, TabList, TabPanel, Tabs} from 'react-tabs';
import Button from '@material-ui/core/Button';
import Dialog from '@material-ui/core/Dialog';
import DialogTitle from '@material-ui/core/DialogTitle';
import IconButton from '@material-ui/core/IconButton';
import DialogContent from '@material-ui/core/DialogContent';
import TextField from '@material-ui/core/TextField';
import CloseIcon from '@material-ui/icons/Close';

import Toolbar from '../../components/UI/Toolbar';
import Chat from '../Chat/Chat';
import Users from '../../components/Users/Users';
import Subjects from '../../components/Subjects/Subjects';
import SubjectsList from '../../components/Subjects/SubjectsList';
import {editUser, logoutUser} from '../../store/actions/userActions';

import './PersonalAccount.css';

class PersonalAccount extends Component {
   state = {
      email: this.props.user.email,
      secondName: this.props.user.secondName,
```

```
    firstName: this.props.user.firstName,
    lastName: this.props.user.lastName,
    password: '',
    secondPassword: '',
    editProfileModal: false,
};

inputChangeHandler = event => {
    this.setState({
        [event.target.name]: event.target.value
    })
};

openEditProfileModal = () => {
    this.setState({editProfileModal: true});
};

closeEditProfileModal = () => {
    this.setState({
        editProfileModal: false,
        email: this.props.user.email,
        firstName: this.props.user.firstName,
        lastName: this.props.user.lastName,
        secondName: this.props.user.secondName,
        password: '',
        secondPassword: ''
    });
};

submitEditProfileModalHandler = event => {
    event.preventDefault();

    if (this.state.password !== this.state.secondPassword) {
        NotificationManager.error('Пароли не совподают!');
    } else {
        const editProfileData = {
            email: this.state.email,
            secondName: this.state.secondName,
            firstName: this.state.firstName,
            lastName: this.state.lastName,
            password: this.state.password,
        };

        this.props.editUser(editProfileData);
        this.closeEditProfileModal();
    }
```

```
  };

  getFieldError = fieldName => {
    return this.props.error && this.props.error.errors &&
this.props.error.errors[fieldName]
      && (this.props.error.errors[fieldName].message)
  };

  render() {
    return (
      <Fragment>
        <Toolbar
          user={this.props.user}
          logout={this.props.logoutUser}
        />
        <div className='personal-block'>
          <div className='container'>
            <div className='top'>
              {
                this.props.user &&
                <h4 className='top__title'>{this.props.user.fullName}</h4>
              }
              <Button className='top__btn common-btn edit-btn'
onClick={this.openEditProfileModal}>
                Редактировать
              </Button>
              <Dialog onClose={this.closeEditProfileModal}
open={this.state.editProfileModal} className='add-user-modal'>
                <DialogTitle onClose={this.closeEditProfileModal}
className='header-modal'>
                  Редактировать пользователя
                  <IconButton aria-label='delete' className='close-modal'
onClick={this.closeEditProfileModal}>
                    <CloseIcon fontSize='large' />
                  </IconButton>
                </DialogTitle>
                <DialogContent>
                  <form className='add-user-form'
onSubmit={this.submitEditProfileModalHandler}>
                    <TextField
                      label='E-mail'
                      type='email'
                      value={this.state.email}
                      name='email'
                      onChange={this.inputChangeHandler}
                      helperText={this.getFieldError('email')}
```

```
    className='add-user-form__input'
  />
  <TextField
    label='Фамилия'
    type='text'
    name='secondName'
    value={this.state.secondName}
    onChange={this.inputChangeHandler}
    helperText={this.getFieldError('secondName')}
    className='add-user-form__input'
  />
  <TextField
    label='Имя'
    type='text'
    name='firstName'
    value={this.state.firstName}
    onChange={this.inputChangeHandler}
    helperText={this.getFieldError('firstName')}
    className='add-user-form__input'
  />
  <TextField
    label='Отчетсво'
    type='text'
    name='lastName'
    value={this.state.lastName}
    onChange={this.inputChangeHandler}
    helperText={this.getFieldError('lastName')}
    className='add-user-form__input'
  />
  <TextField
    label='Пароль'
    type='password'
    name='password'
    value={this.state.password}
    onChange={this.inputChangeHandler}
    helperText={this.getFieldError('password')}
    className='add-user-form__input'
    required
  />
  <TextField
    label='Повторите пароль'
    type='password'
    name='secondPassword'
    value={this.state.secondPassword}
    onChange={this.inputChangeHandler}
    className='add-user-form__input'
```

```
                        required
                      />
                      <Button autoFocus color='primary' type='submit'>
                        Сохранить
                      </Button>
                    </form>
                  </DialogContent>
                </Dialog>
              </div>
              <div className='personal-menu'>
                <Tabs>
                  <TabList className='personal-menu__list'>
                    <Tab className='personal-menu__item'>Чат</Tab>
                    {
                      this.props.user && this.props.user.role === 'admin' &&
                      <Tab className='personal-menu__item'>Список
пользователей</Tab>
                    }
                    {
                      this.props.user && (this.props.user.role === 'admin' ||
this.props.user.role === 'teacher') &&
                      <Tab className='personal-menu__item'>Мои
дисциплины</Tab>
                    }
                    {
                      this.props.user && this.props.user.role === 'admin' &&
                      <Tab className='personal-menu__item'>Список
дисциплин</Tab>
                    }
                  </TabList>
                  <TabPanel>
                    <Chat/>
                  </TabPanel>
                  {
                    this.props.user && this.props.user.role === 'admin' &&
                    <TabPanel>
                      <Users/>
                    </TabPanel>
                  }
                  {
                    this.props.user && (this.props.user.role === 'admin' ||
this.props.user.role === 'teacher') &&
                    <TabPanel>
                      <Subjects/>
                    </TabPanel>
                  }
```

```
                        {
                            this.props.user && this.props.user.role === 'admin' &&
                            <TabPanel>
                                <SubjectsList/>
                            </TabPanel>
                        }
                    </Tabs>
                </div>
            </div>
        </div>
    </Fragment>
    );
  }
}

const mapStateToProps = state => ({
  user: state.users.user,
  error: state.users.editUserError,
});

const mapDispatchToProps = dispatch => ({
  logoutUser: () => dispatch(logoutUser()),
  editUser: userData => dispatch(editUser(userData))
});

export default withRouter(connect(mapStateToProps,
mapDispatchToProps)(PersonalAccount));
```

Backend

server.js

```
const express = require('express');
const mongoose = require ('mongoose');
const config = require('./config');
const cors = require ('cors');
const {createServer} = require('http');

const users = require('./app/user');
const notice = require('./app/notice');
const adminUsers = require('./app/admin/users');
const subject = require('./app/subject');
const dialog = require('./app/dialog');
const message = require('./app/message');
const tryAuth = require('./middleware/tryAuth');
const updateLastSeen = require('./middleware/updateLastSeen');
const createSocket = require('./core/socket');
```

```
const app = express();
const http = createServer(app);
const io = createSocket(http);

app.use(cors());
app.use(express.json());
app.use(express.static('public'));
app.use(tryAuth);
app.use(updateLastSeen);
app.use(function(req, res, next) {
   req.io = io;
   next();
});

const port = 8000;

mongoose.connect(config.dbUrl, config.mongoOption).then(() => {
   http.listen(port, () => {
      app.use('/admin/users', adminUsers);
      app.use('/users', users);
      app.use('/notice', notice);
      app.use('/subject', subject);
      app.use('/dialog', dialog);
      app.use('/messages', message);
      console.log(`Server started on ${port} port`);
   });
});
```

Models/ Dialog.js

```
const mongoose = require('mongoose');

const Schema = mongoose.Schema;

const DialogSchema = new Schema ({
   author: {
      type: Schema.Types.ObjectId,
      ref: 'User',
      required: true
   },
   partner: {
      type: Schema.Types.ObjectId,
      ref: 'User',
      required: true
   },
   lastMessage: {
      type: Schema.Types.ObjectId,
```

```
            ref: 'Message',
        }
    }, {timestamps: true});

    const Dialog = mongoose.model('Dialog', DialogSchema);

module.exports = Dialog;
```

models/ Messages.js

```
    const mongoose = require('mongoose');

    const Schema = mongoose.Schema;

    const MessagesSchema = new Schema ({
        dialog: {
            type: Schema.Types.ObjectID,
            ref: 'Dialog',
            required: true
        },
        user: {
            type: Schema.Types.ObjectID,
            ref: 'User',
            required: true
        },
        text: {
            type: String
        },
        read: {
            type: Boolean,
            default: false
        },
        attachments: {type: String},
    }, {
        timestamps: true,
    });

    const Messages = mongoose.model('Message', MessagesSchema);

module.exports = Messages;
```

models/ Notice.js

```
    const mongoose = require('mongoose');

    const Schema = mongoose.Schema;

    const NoticeSchema = new Schema ({
```

```
    user: {
      type: Schema.Types.ObjectId,
      ref: 'User',
      required: true
    },
    title: {
      type: String,
      required: [true, 'Поле обязательно для заполнения']
    },
    description: String,
    noticeFile: [String],
    dateTime: String
  });

  const Notice = mongoose.model('Notice', NoticeSchema);

module.exports = Notice;
```

models/ Subjects.js

```
  const mongoose = require('mongoose');

  const Schema = mongoose.Schema;

  const SubjectSchema = new Schema ({
    subject: {
      type: String,
      required: [true, 'Поле обязательно для заполнения']
    },
    materials: {
      type: [String],
      default: []
    },
  });

  const Subject = mongoose.model('Subject', SubjectSchema);

module.exports = Subject;
```

models/ User.js

```
  const mongoose = require('mongoose');
  const bcrypt = require('bcrypt');
  const nanoid = require("nanoid");
  const differenceInMinutes = require('date-fns/difference_in_minutes');

  const SALT_WORK_FACTOR = 10;
```

```
const Schema = mongoose.Schema;

const UserSchema = new Schema({
   email: {
      type: String,
      required: [true, 'Поле обязательно для заполения'],
      unique: true,
      validate: {
         validator: async  function (value) {
            if (!this.isModified('email')) return;
            const user = await User.findOne({email: value});
            if (user) throw new Error(`Email ${value} уже используется!`);
         }
      }
   },
   role: {
      type: String,
      required: true,
      default: 'student',
      enum: ['admin', 'student', 'teacher']
   },
   password: {
      type: String,
      required: [true, 'Поле обязательно для заполения'],
      minlength: [8, 'Должно быть не меньше 8 символов.'],
   },
   firstName: {
      type: String,
      required: [true, 'Поле обязательно для заполения']
   },
   secondName: {
      type: String,
      required: [true, 'Поле обязательно для заполения']
   },
   lastName: {
      type: String,
      required: [true, 'Поле обязательно для заполения']
   },
   fullName: {
      type: String,
   },
   appointSubjects: {
      type: [{type: Schema.ObjectId, ref: 'Subject'}],
      default: [],
   },
   token: {
```

```
            type: String,
            required: true
        },
        last_seen: {
            type: Date,
            default: new Date
        },
    }, {timestamps: true});

    UserSchema.methods.checkPassword = function (password) {
        return bcrypt.compare(password, this.password);
    };

    UserSchema.methods.generateToken = function () {
        this.token = nanoid();
    };

    UserSchema.virtual('isOnline').get(function() {
        return differenceInMinutes(new Date().toISOString(), this.last_seen) < 5;
    });

    UserSchema.pre('save', async function (next) {
        if (!this.isModified('password')) return next();

        const salt = await bcrypt.genSalt(SALT_WORK_FACTOR);
        const hash = await bcrypt.hash(this.password, salt);

        this.password = hash;

        next();
    });

    UserSchema.set('toJSON', {
        virtuals: true,
        transform: (doc, ret, options) => {
            delete ret.password;
            return ret;
        }
    });

    const User = mongoose.model('User', UserSchema);

module.exports = User;
```

auth.js

```
const User = require('../models/User');

const auth = async (req, res, next) => {
    const token = req.get('Authorization');

    if (!token) {
        return res.status(401).send({error: 'Token not provided'});
    }

    const user = await User.findOne({token});

    if (!user) {
        return res.status(401).send({error: 'Token incorrect'});
    }

    req.user = user;

    next();
};

module.exports = auth;
```

permit.js

```
const permit = (...roles) => {
    return (req, res, next) => {
        if (!req.user) {
            return res.status(401).send({message: 'Unauthenticated'});
        }

        if (!roles.includes(req.user.role)) {
            return res.status(403).send({message: 'Unauthorized'})
        }

        next();
    };
};

module.exports = permit;
```

tryAuth.js

```
const User = require('../models/User');

const tryAuth = async (req, res, next) => {
    const token = req.get('Authorization');
```

```
    if (!token) {
      return next();
    }

    const user = await User.findOne({token});

    if (!user) {
      return next();
    }

    req.user = user;

    next();
  };

module.exports = tryAuth;
```

updateLastSeen.js

```
  const User = require('../models/User');

  const updateLastSeen = async (req, res, next) => {
    if (req.user) {
      User.findOneAndUpdate(
        { _id: req.user.id },
        {
          last_seen: new Date()
        },
        { new: true },
        () => {}
      );
    }
    next();
  };

module.exports = updateLastSeen;
```

users.js

```
  const express = require('express');
  const auth = require('../../middleware/auth');
  // const updateLastSeen = require('../../middleware/updateLastSeen');
  const permit = require('../../middleware/permit');
  const nanoid = require("nanoid");
  const User = require('../../models/User');

  const router = express.Router();
```

```
router.get('/', auth, async (req, res) => {
   try {
      const users = await User.find({}, {
         email: true,
         firstName: true,
         secondName: true,
         lastName: true,
         fullName: true,
         role: true,
         appointSubjects: true,
         last_seen:
true}).populate('appointSubjects').sort('secondName').sort('firstName');

      return res.send(users)
   } catch (e) {
      return res.status(500).send(e)
   }
});

router.get('/:id', async (req, res) => {
   const userId = req.params.id;
   try {
      const user = await User.findById(userId).populate('user', 'id
fullName').populate('appointSubjects');
      res.send(user);
   } catch (e) {
      if (e.name === 'ValidationError') {
         return res.status(400).send(e)
      }
      return res.status(500).send(e)
   }
});

router.post('/', [auth, permit('admin')], async (req, res) => {
   const userData = {
      email: req.body.email,
      role: req.body.role,
      password: req.body.password,
      firstName: req.body.firstName,
      secondName: req.body.secondName,
      lastName: req.body.lastName,
      fullName: `${req.body.secondName} ${req.body.firstName}
${req.body.lastName}`,
      token: nanoid(),
   };
```

```
    const user = new User (userData);

    user.save()
        .then(results => res.send(results))
        .catch(error => {
            res.status(400).send(error)
        } );
});

router.patch('/role_change/:id', [auth, permit('admin')], async (req, res) => {
    try {
        const user = await User.findById(req.params.id);

        user.role = req.body.role;

        await user.save();

        res.sendStatus(200)

    } catch (e) {
        return res.status(500).send(e)
    }
});

router.patch('/:id', [auth, permit('admin')], async (req, res) => {
    const singleUser = await User.findByIdAndUpdate({_id: req.params.id}, {$push:
{appointSubjects: req.body.appointSubjects}});

    singleUser.save()
        .then(result => res.send(result))
        .catch(error => res.status(400).send(error));
});

module.exports = router;
```

dialog.js

```
const express = require('express');
const auth = require('../middleware/auth');
const Dialog = require('../models/Dialog');
const Message = require('../models/Messages');

const router = express.Router();

router.get('/', auth, async (req, res) => {
    const userId = req.user._id;
    Dialog.find()
```

```
    .or([{ author: userId }, { partner: userId }])
    .populate(['author', 'partner'])
    .populate({
      path: 'lastMessage',
      populate: {
        path: 'user',
      },
    })
    .exec(function(err, dialogs) {
      if (err) {
        return res.status(404).send({
          message: 'Dialogs not found',
        });
      }
      return res.json(dialogs);
    });
});

router.post('/', auth, async (req, res) => {
    const dialogData = {
        author: req.user._id,
        partner: req.body.partner
    };

    Dialog.findOne(
        {
          author: req.user._id,
          partner: req.body.partner,
        },
        (err, user) => {
          if (err) {
            return res.status(500).send({
              status: 'error',
              message: err,
            });
          }
          if (user) {
            return res.status(403).send({
              status: 'error',
              message: 'Такой диалог уже есть',
            });
          } else {
            const dialog = new Dialog(dialogData);

            dialog
              .save()
```

```
                .then((dialogObj) => {
                  const message = new Message({
                    text: req.body.text,
                    user: req.user._id,
                    dialog: dialogObj._id,
                  });

                  message
                    .save()
                    .then(() => {
                      dialogObj.lastMessage = message._id;
                      dialogObj.save().then(() => {
                        res.send(dialogObj);
                        req.io.emit('SERVER:DIALOG_CREATED', {
                          ...dialogData,
                          dialog: dialogObj,
                        });
                      });
                    })
                    .catch(reason => {
                      res.send(reason);
                    });
                })
                .catch(err => {
                  res.send({
                    status: 'error',
                    message: err,
                  });
                });
            }
          },
        );
    });

module.exports = router;
```

message.js

```
    const express = require('express');
    const auth = require('../middleware/auth');
    const multer = require('multer');
    const config = require('../config');
    const Message = require('../models/Messages');
    const Dialog = require('../models/Dialog');

    const storage = multer.diskStorage({
       destination: (req, file, cb) => {
```

```
        cb(null, config.uploadPathMessageFile);
    },
    filename: (req, file, cb) => {
        cb(null, file.originalname);
    }
});

const upload = multer({storage});

const router = express.Router();

const updateReadStatus = (req, res, userId, dialogId) => {
    Message.updateMany(
        { dialog: dialogId, user: { $ne: userId } },
        { $set: { read: true } },
        (err) => {
            if (err) {
                return res.status(500).json({
                    status: 'error',
                    message: err,
                });
            }
            req.io.emit('SERVER:MESSAGES_READ', {
                userId,
                dialogId,
            });
        },
    );
};

router.get('/', auth, async (req, res) => {
    const dialogId = req.query.dialog;
    const userId = req.user._id;

    updateReadStatus(req, res, userId, dialogId);

    try {
        const messages = await Message.find({dialog:
dialogId}).populate('dialog').populate('user', '_id fullName');

        res.send(messages);
    } catch (e) {
        res.sendStatus(500);
    }
});
```

```
router.post('/', auth, upload.single('attachments'), async (req, res) => {
  const userId = req.user._id;

  const messageData = {
    dialog: req.body.dialog,
    text: req.body.text,
    attachments: req.body.attachments,
    user: userId
  };

  if (req.file) {
    messageData.attachments = req.file.filename;
  }

  const message = new Message (messageData);

  updateReadStatus(req, res, userId, req.body.dialog._id);

  message.save()
    .then(results => {
      results.populate(['dialog', 'user'], (err, message) => {
        if (err) {
          return res.status(500).send({
            status: 'error',
            message: err,
          });
        }
        Dialog.findOneAndUpdate(
          { _id: messageData.dialog },
          { lastMessage: message._id },
          { upsert: true },
          function(err) {
            if (err) {
              return res.status(500).send({
                status: 'error',
                message: err,
              });
            }
          },
        );
        res.send(message);

        req.io.emit('SERVER:NEW_MESSAGE', message);
      });
    })
    .catch(error => {
```

```
            res.status(400).send(error)
        });
    });

module.exports = router;
```

notice.js

```
    const express = require('express');
    const multer = require('multer');
    const config = require('../config');
    const auth = require('../middleware/auth');
    const tryAuth = require('../middleware/tryAuth');
    const permit = require('../middleware/permit');
    const Notice = require('../models/Notice');

    const storage = multer.diskStorage({
      destination: (req, file, cb) => {
        cb(null, config.uploadPathNotice);
      },
      filename: (req, file, cb) => {
        cb(null, file.originalname);
      }
    });

    const upload = multer({storage});

    const router = express.Router();

    router.get('/', tryAuth, async (req, res) => {
      try {
        const notice = await Notice.find().populate('user', '_id firstName
    secondName').sort({dateTime: -1});

        const page = parseInt(req.query.page);
        const limit = parseInt(req.query.limit);

        const startIndex = (page - 1) * limit;
        const endIndex = page * limit;

        const notices = {};

        if (endIndex < notice.length) {
          notices.next = {
            page: page + 1,
            limit: limit
          };
```

```
            }

            if (startIndex > 0) {
                notices.previous = {
                    page: page - 1,
                    limit: limit
                };
            }

            notices.notices = notice.slice(startIndex, endIndex);
            notices.totalNotices = notice.length;
            res.send(notices);
        } catch (e) {
            res.sendStatus(500);
        }
    });

    router.post('/', [auth, permit('admin')], upload.array('noticeFile', 5), (req, res) => {
        const noticeData = {
            user: req.user._id,
            title: req.body.title,
            description: req.body.description,
            noticeFile: [],
            dateTime: new Date().toISOString()
        };

        if (req.files) {
            noticeData.noticeFile = req.files.map(file => file.filename)
        }

        const notice = new Notice (noticeData);

        notice.save()
            .then(results => res.send(results))
            .catch(error => {
                res.status(400).send(error)
            } );
    });

    router.delete('/:id', [auth, permit('admin')], (req, res) => {
        Notice.deleteOne({_id: req.params.id})
            .then(result => res.send(result))
            .catch(error => res.status(403).send(error))
    });

module.exports = router;
```

subject.js

```
const express = require('express');
const multer = require('multer');
const config = require('../config');
const auth = require('../middleware/auth');
const permit = require('../middleware/permit');
const Subject = require('../models/Subjects');

const storage = multer.diskStorage({
   destination: (req, file, cb) => {
      cb(null, config.uploadPathSubject);
   },
   filename: (req, file, cb) => {
      cb(null, file.originalname);
   }
});

const upload = multer({storage});

const router = express.Router();

router.get('/', async (req, res) => {
   try {
      const subject = await Subject.find().sort('subject');

      res.send(subject);
   } catch (e) {
      res.sendStatus(500);
   }
});

router.post('/', [auth, permit('admin')], async (req, res) => {
   const subjectData = {
      subject: req.body.subject
   };

   const subject = new Subject (subjectData);

   subject.save()
      .then(results => res.send(results))
      .catch(error => {
         res.status(400).send(error)
      });
});
```

```
router.patch('/:id', [auth, permit('admin', 'teacher')], upload.array('materials', 5), async
(req, res) => {
  const singleSubjectData = req.body;

  if (req.files) {
    singleSubjectData.materials = req.files.map(file => file.filename)
  }

  const singleSubject = await Subject.findByIdAndUpdate({_id: req.params.id},
{$push: {materials: {$each: singleSubjectData.materials}}});

  singleSubject.save()
    .then(result => res.send(result))
    .catch(error => res.status(400).send(error));
});

module.exports = router;
```

user.js

```
const express = require('express');
const tryAuth = require('../middleware/tryAuth');
const User = require('../models/User');
const updateLastSeen = require('../middleware/updateLastSeen');

const router = express.Router();

router.get('/', tryAuth, async (req, res) => {
  try {
    const token = req.get('Authorization');

    if (!token) {
      return res.status(401).send({error: 'Authorization headers not present'});
    }

    const user = await User.findOne({token});

    if (!user) {
      return res.status(401).send({error: 'Неверный токен'});
    }

    res.send(user);
  } catch (e) {
    res.sendStatus(500);
  }
});
```

```
router.post('/', async (req, res) => {
    const user = new User({
        email: req.body.email,
        password: req.body.password,
        firstName: req.body.firstName,
        secondName: req.body.secondName,
        lastName: req.body.lastName,
        fullName: `${req.body.secondName} ${req.body.firstName} ${req.body.last}`,
    });
    user.generateToken();

    try {
        await user.save();
        return res.send({message: 'Пользователь зарегистрирован', user});
    } catch (error) {
        return res.status(400).send(error)
    }
});

router.post('/sessions', updateLastSeen, async (req, res) => {
    const user = await User.findOneAndUpdate({email: req.body.email}, {$set:
{last_seen: new Date()}}).populate('user');

    if (!user) {
        return res.status(400).send({error: 'Такого пользователя нет'});
    }

    const isMatch = await user.checkPassword(req.body.password);

    if (!isMatch) {
        return res.status(400).send({error: 'Неверный email или пароль'});
    }

    user.generateToken();

    await user.save();

    res.send({message: 'Успешная авторизация', user});
});


router.delete('/sessions', async (req, res) => {
    const token = req.get('Authorization');
    const success = {message: 'Logged out'};

    if (!token) {
```

```
            return res.send(success);
        }
        const user = await User.findOne({token});

        if (!user) {
            return res.send(success);
        }

        user.generateToken();
        await user.save();

        return res.send(success);
    });

    router.put('/sessions', tryAuth, async (req, res) => {
        const token = req.get('Authorization');

        if (!token) {
            return res.status(401).send({error: 'Authorization headers not present'});
        }

        const user = await User.findOne({token});

        if (!user) {
            return res.status(401).send({error: 'Неверный токен'});
        }

        user.email = req.body.email;
        user.secondName = req.body.secondName;
        user.firstName = req.body.firstName;
        user.lastName = req.body.lastName;
        user.fullName = `${req.body.secondName} ${req.body.firstName}
    ${req.body.lastName}`;
        user.password = req.body.password;

        await user.save()
            .then(results => res.send(results))
            .catch(error => {
                res.status(400).send(error)
            });
    });

module.exports = router;
```