МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования

«Гродненский государственный университет имени Янки Купалы»

Факультет математики и информатики

Кафедра современных технологий программирования

ХУДОБА ДАНИИЛ МИХАЙЛОВИЧ

**Разработка клиентской части веб-сайта Гродненского областного клинического центра “Психиатрия-Наркология”**

Дипломная работа

студента 4 курса дневной формы получения образования

Специальность: 1-40 01 01 «Программное обеспечение информационных технологий»

Руководитель:  
Урбан О.И.

старший преподаватель кафедры СТП

\_\_\_\_\_\_\_\_\_\_\_\_\_

Студент: \_\_\_\_\_\_\_\_\_\_\_\_\_

**Допущен к защите** 15 мая 2024 г.

Заведующий кафедрой  
кандидат технических наук, доцент:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ / Карканица А.В.

**Гродно, 2024**

#### РЕФЕРАТ

ХУДОБА ДАНИИЛ МИХАЙЛОВИЧ

**Разработка клиентской части веб-сайта Гродненского областного клинического центра “Психиатрия-Наркология”**

Дипломная работа содержит 47 страниц, 28 рисунков, 11 использованных источников литературы.

Ключевые слова: пагинация, хранилище, запрос, база данных, пользователь.

Цель дипломной работы – разработка клиентской части веб-сайта Гродненского областного клинического центра “Психиатрия-Наркология”.

Объектом исследования выступают функциональные возможности и технические реализации клиентской части веб-сайтов, реализующих взаимодействие с пользователем.

Предметом исследования настоящей работы является архитектура клиент-серверных приложений, методы и технологии взаимодействия с пользователем.

Методы исследования – анализ рынка и конкурентов, анализ предметной области, техническое исследование, разработка пользовательских сценариев, архитектурное проектирование.

Полученные результаты и их апробация. В ходе выполнения дипломной работы была разработана клиентская часть веб-сайта Гродненского областного клинического центра “Психиатрия-Наркология”, реализующая механизмы авторизации, регистрации, изменения пользовательской информации, распределения ролей, добавления на веб-сайт различной отображаемой информации.

Область возможного практического применения результатов исследования. Разработанный веб-сайт является примером для разработки подобных информационных сайтов, ориентированных на отображение большого количества данных. Он может быть использован как в коммерческих, так и в некоммерческих целях.

ОГЛАВЛЕНИЕ

[ПЕРЕЧЕНЬ СОКРАЩЕНИЙ И ОБОЗНАЧЕНИЙ 4](#_Toc169564775)

[ВВЕДЕНИЕ 5](#_Toc169564776)

[ГЛАВА 1 6](#_Toc169564777)

[ВВЕДЕНИЕ В ОБЛАСТЬ ИНФОРМАЦИОННЫХ САЙТОВ 6](#_Toc169564778)

[1.1 Обзор существующих решений 6](#_Toc169564779)

[1.1.1 Сайт Гомельской областной клинической психиатрической больницы 6](#_Toc169564780)

[1.1.2 Психиатрическая клиника Василия Шурова 7](#_Toc169564781)

[1.1.3 УЗ “Областная психоневрологическая больница “Островля” Лидского района” 7](#_Toc169564782)

[1.2 Описание предметной области 8](#_Toc169564783)

[1.3 Вывод по главе 1 9](#_Toc169564784)

[ГЛАВА 2 10](#_Toc169564785)

[ПРОЕКТИРОВАНИЕ КЛИЕНТСКОЙ ЧАСТИ ВЕБ-САЙТА 10](#_Toc169564786)

[2.1 Выбор необходимых технологий разработки 10](#_Toc169564787)

[2.2 Проектирование внешнего вида веб-сайта 10](#_Toc169564788)

[2.3 Проектирование взаимодействия с авторизацией 12](#_Toc169564789)

[2.3.1 Проектирование авторизации 12](#_Toc169564790)

[2.3.2 Проектирование регистрации 13](#_Toc169564791)

[2.4 Проектирование изменения информаций пользователя 15](#_Toc169564792)

[2.5 Проектирование публикации отзывов, благодарностей и вопросов 16](#_Toc169564793)

[2.5.1 Проектирование публикации отзывов 16](#_Toc169564794)

[2.5.2 Проектирование публикации благодарностей 16](#_Toc169564795)

[2.5.3 Проектирование публикации вопросов 17](#_Toc169564796)

[2.6 Проектирование пагинации 18](#_Toc169564797)

[2.6.1 Проектирование создания данных при помощи админ-панели 18](#_Toc169564798)

[2.6.2 Проектирование отображения элементов пагинации 20](#_Toc169564799)

[2.7 Вывод по главе 2 21](#_Toc169564800)

[ГЛАВА 3 22](#_Toc169564801)

[ПРОГРАММНАЯ РЕАЛИЗАЦИЯ КЛИЕНТСКОЙ ЧАСТИ ВЕБ-САЙТА 22](#_Toc169564802)

[3.1 Создание общей структуры приложения 22](#_Toc169564803)

[3.1.1 Навигация и роутинг 22](#_Toc169564804)

[3.1.2 Глобальное хранилище 24](#_Toc169564805)

[3.1.3 Настройка axios 27](#_Toc169564806)

[3.2 Реализация авторизации, регистрации, изменения и проверки пользователя 28](#_Toc169564807)

[3.2.1 Реализация авторизации 28](#_Toc169564808)

[3.2.2 Реализация регистрации 29](#_Toc169564809)

[3.2.3 Реализация проверки пользователя 30](#_Toc169564810)

[3.2.4 Реализация изменения пользователя 31](#_Toc169564811)

[3.3 Реализация добавления данных в БД при помощи админ-панели 33](#_Toc169564812)

[3.4 Реализация пагинации 35](#_Toc169564813)

[3.4.1 Реализация функции получения данных для пагинации 35](#_Toc169564814)

[3.4.2 Реализация отображения элементов пагинации 37](#_Toc169564815)

[3.4.3 Реализация отображения страниц и модальных окон пагинации 38](#_Toc169564816)

[3.5 Вывод по главе 3 39](#_Toc169564817)

[ЗАКЛЮЧЕНИЕ 40](#_Toc169564818)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 41](#_Toc169564819)

**ПЕРЕЧЕНЬ СОКРАЩЕНИЙ И ОБОЗНАЧЕНИЙ**

**БД** – база данных

**Backend** - серверная часть приложения, реализует бизнес логику и обработку данных приложения

**Frontend** - клиентская часть приложения, с которой взаимодействует пользователь

**Пагинация** – постраничное отображение данных с БД и их нумерация в определённом блоке.

**Роутинг(маршрутизация)** - процесс, который отвечает за определение обработчика для конкретной запрашиваемой страницы.

**Стор(хранилище)** - централизованный объект состояния, который управляет всеми данными приложения.

**Промис** — асинхронный оператор, предназначенный для выполнения задач, которые могут занять некоторое время, например, запросы к серверу, чтение файлов или другие асинхронные операции.

# **ВВЕДЕНИЕ**

Психиатрическая больница - место, где врачи занимаются обследованием, лечением и реабилитацией при психических расстройствах: от легких до тяжелых, включая неврозы, реактивные состояния, психозы, алкогольную и наркотическую зависимость.

В психиатрических больницах оказывается медицинская помощь и проводятся консультации специалистов при подозрении на психическое заболевание, чтобы определить наличие диагноза и предрасположенность к нарушениям поведения.

К сожалению, не у всех есть возможность лично посетить такие учреждения для записи на консультацию или получения информации. Поэтому сегодня создаются веб-сайты, где можно найти все нужные данные.

**Цель дипломной работы** – разработать клиентскую часть веб-сайта Гродненского областного клинического центра “Психиатрия-Наркология”. Веб-сайт состоит из трех компонентов:

1) Backend. Данное приложение получает и обрабатывает клиентские запросы после чего отправляет ответ пользователю.

2) Database (база данных). Данная часть приложения отвечает за хранение, изменение и предоставление данных для backend части приложения.

3) Frontend. Эта часть приложения предоставляет пользователю интерфейс для взаимодействия с веб-сайтом. Оно должно обрабатывать события пользовательского взаимодействия и при необходимости отправлять запросы на сервер (backend), после чего обрабатывать его ответ.

**Предусмотрено решение следующих задач:**

1) Выявление требований к разрабатываемому веб-сайту, а также анализ предметной области.

2) Разработка моделей данных на основе анализа предметной области.

3) Выбор технологий и средств разработки для клиентской части веб-сайта.

4) Разработать архитектуру клиент-серверного приложения.

5) Проектирование структуры хранилища базы данных, а также его дальнейшая реализация.

6) Проектирование прототипа и дальнейшее создание макета внешнего вида веб-сайта.

7) Проектирование и дальнейшая реализация клиентской части веб приложения.

# **ГЛАВА 1**

# **ВВЕДЕНИЕ В ОБЛАСТЬ ИНФОРМАЦИОННЫХ САЙТОВ**

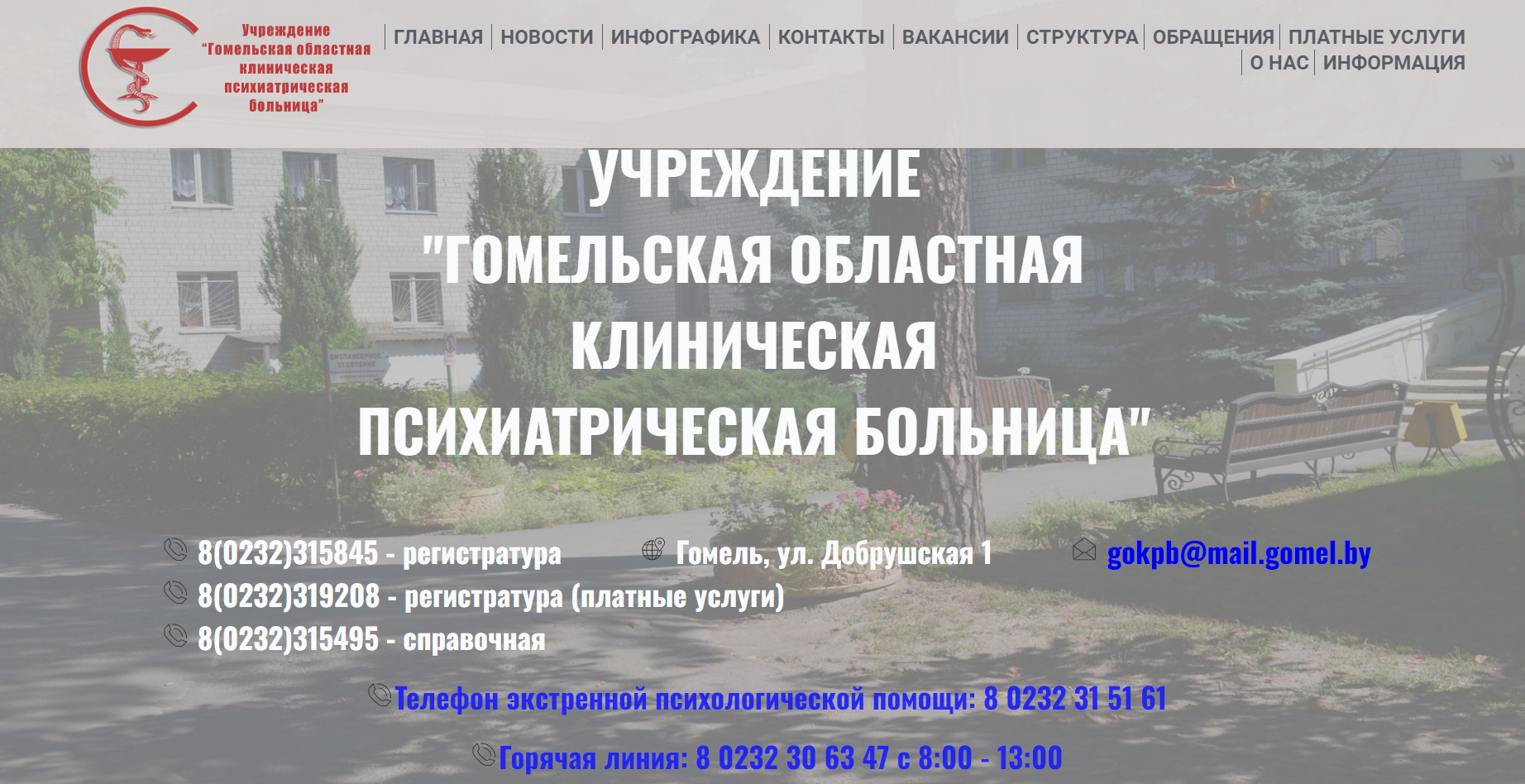
## **1.1 Обзор существующих решений**

Чтобы выявить требования, которые нужны для реализации клиентской части веб-сайта Гродненского областного клинического центра “Психиатрия-Наркология”, необходимо рассмотреть уже существующие сайты данного направления.

### **1.1.1 Сайт Гомельской областной клинической психиатрической больницы**

На первый взгляд сайт выглядит вполне нормально, но опустившись ниже на главной странице можно увидеть максимально простое оформление, не имеющее нормально разделение элементов, а также маленький и не самый приятный на вид текст.

Но также можно подметить, что все кнопки работают так, как надо, а некоторые даже имеют нестандартный вид и изменяют свою внешность при наведении на них курсором.

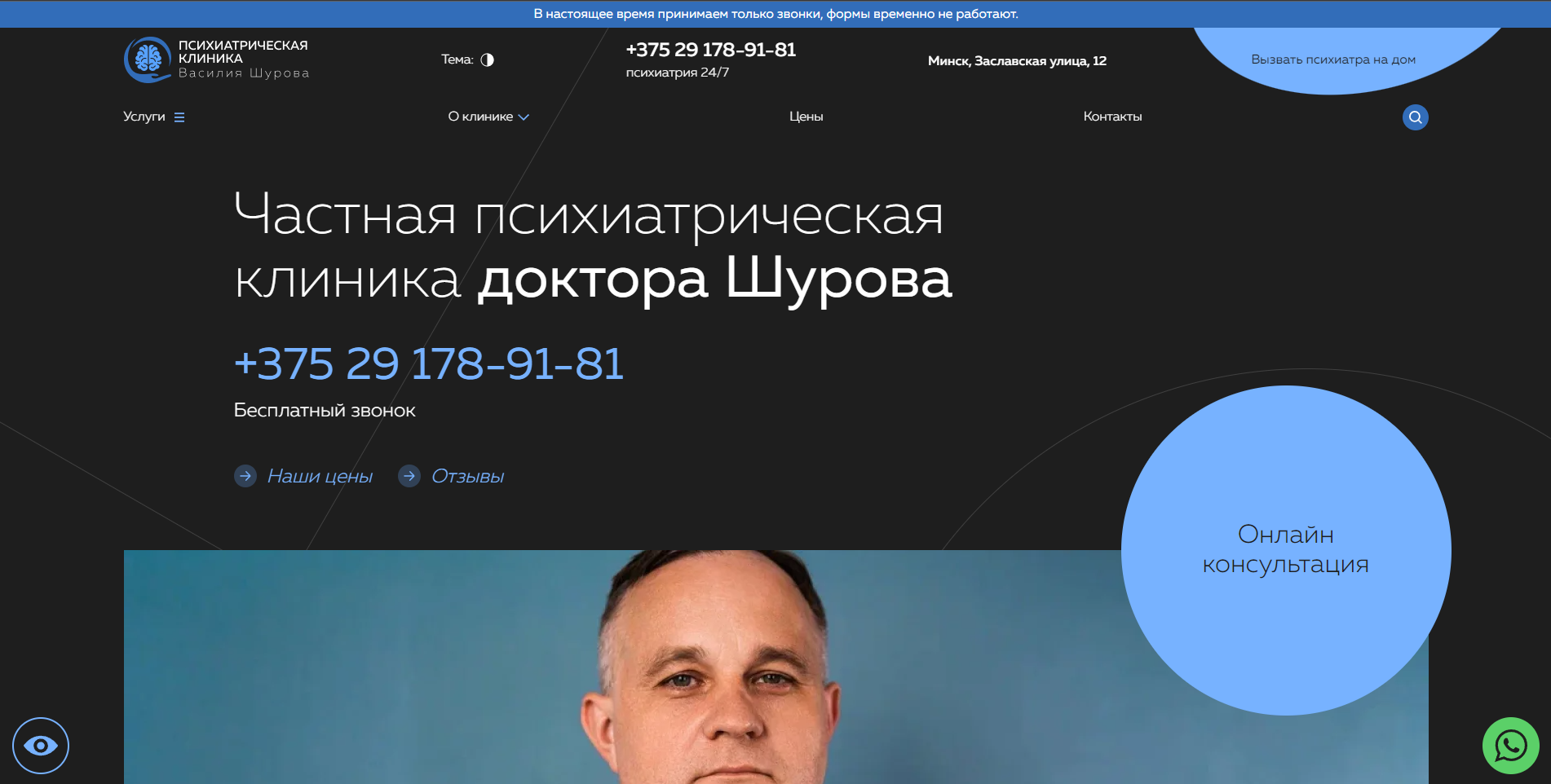


**Рисунок 1.1 - Главная страница сайта учреждение “Гомельская областная клиническая психиатрическая больница”**

### **1.1.2 Психиатрическая клиника Василия Шурова**

Совершенно другой случай по сравнению с ранее описанным сайтом. Очень приятное оформление, каждая кнопка кликабельна и меняет внешний вид при наведении на неё курсором. Информация также избыточна, но подано всё это в куда более приятном виде. Ещё можно отметить возможность изменять главную тему сайта на тёмную или светлую соответственно, маленькая вещь, но тоже приятно. Также у сайта есть редкая, но очень нужная для некоторых людей опция, а именно возможность менять шрифт, цвет и другие различные элементы сайта для людей со слабым зрением.

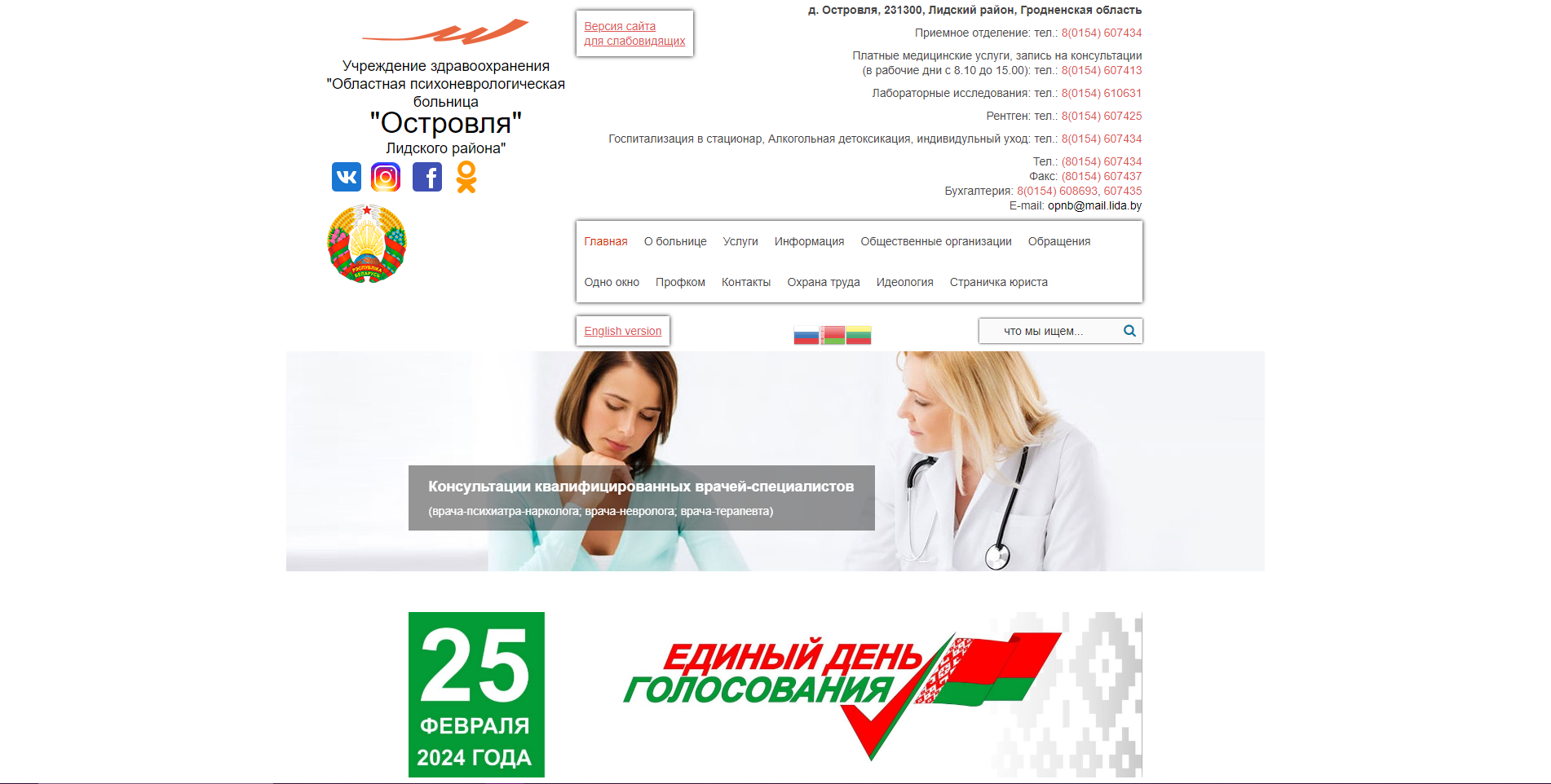
Справедливости ради стоит еще уточнить, что это сайт частной клиники, а не государственной, соответственно и бюджеты у подобных сайтов будут гораздо выше.



**Рисунок 1.2 - Главная страница сайта психиатрической клиники Василия Шурова**

### **1.1.3 УЗ “Областная психоневрологическая больница “Островля” Лидского района”**

Сразу бросается в глаза простое, но при этом хорошее оформление, без лишних элементов и текста, а также меняющиеся изображения на главной странице. Все кнопки кликабельны и выполняют нужный функционал, еще некоторые из них имеют плавное появление окошка с дополнительной информацией. У сайта имеется смена локализации на разные языки, а именно на русский, английский, белорусский и литовский. Также имеется переключение версии сайта на версию для слабовидящих, что определенно является плюсом. Вся опубликованная информация избыточна и понятна для чтения.



**Рисунок 1.3 - Главная страница сайта УЗ “Областная психоневрологическая больница “Островля” Лидского района”**

## **1.2 Описание предметной области**

Информационный сайт представляет собой веб-ресурс, который содержит информацию о конкретной теме, продукте, услуге или организации. Основная цель информационного сайта — предоставить посетителям необходимую информацию, ответить на их вопросы и помочь им понять предлагаемые продукты или услуги.

**Внешняя архитектура.** Главной нашей целью является подготовить front-end часть для информационного сайта, чтобы она корректно отображала нужную информацию и взаимодействовала с сервером, но для начала нужно определить, какую внешнюю архитектуру имеет наш веб-сайт.

Веб-сайт имеет перечисленные страницы:

**• Главная страница**

**• Структура**

**• Новости коллектива**

**• Вакансии**

**• Услуги**

**• Обращения граждан**

**• Контакты**

Также веб-сайт имеет личный кабинет пользователя и различные роли, а именно:

**• Незарегистрированный пользователь**

**• Пользователь**

**• Модератор**

**• Администратор**

Исходя от роли пользователя ему будет доступен определенный функционал, а также отображение определенных элементов веб-сайта.

Администратор и модератор будут иметь возможность добавлять различные элементы на сайт (новости, публикации, фотографии в галерею и т.д.) и их редактировать при помощи админ-панели. С её помощью также модератор и администратор имеют возможность назначать роли для пользователей.

**Внешний вид.** Для начала создаётся прототип внешнего вида веб-сайта, чтобы иметь представление, какой вид приложение будет иметь в будущем. После создания прототипа мы приступаем к созданию его макета. Макет отличается от прототипа тем, что является завершающий этапом представления внешнего вида веб-сайта, после создания которого можно приступать к созданию уже самого приложения.

По началу разработки устанавливаем все необходимые технологии в нашу среду разработки, после чего создаём структуру приложения.

## **1.3 Вывод по главе 1**

Информационные сайты являются неотъемлемой частью сегодняшнего интернета. Имея доступ в интернет, они помогаю просматривать необходимую информацию об интересующих вещах, без необходимости выходить из дома.

Веб-сайты, рассмотренные в данной главе, позволили выявить основные аспекты, связанные с реализацией приложения, а именно:

1. Реализация пагинации;
2. Реализация роутинга и навигации по страницам.
3. Реализация публикации отзывов, благодарностей и вопросов пользователем, а также их просмотр модератором и администратором;
4. Реализация личного кабинета;
5. Реализация пагинации в админ-панели для добавления информации в БД;
6. Реализация регистрации и авторизации пользователя;

Целью создания клиентской части веб-сайта должен стать программный продукт, удовлетворяющий перечисленным ранее требованиям.

# **ГЛАВА 2**

# **ПРОЕКТИРОВАНИЕ КЛИЕНТСКОЙ ЧАСТИ ВЕБ-САЙТА**

## **Выбор необходимых технологий разработки**

Так как основным языком для разработки нашего приложения является JavaScript, то будет правильным использовать его библиотеку React для упрощённого создания пользовательского интерфейса. После установки и разворачивания React приложения нам дополнительно нужно установить следующие технологии:

* **Axios**
* **React-router-dom**
* **Mobx**
* **Mobx-react-lite**
* **Bootstrap**

Рассмотрим перечисленные технологии подробно.

**Axios** — это один из самых популярных HTTP клиентов для браузеров и node.js, основанный на промисах. В Axios есть поддержка запросов, получение ответов от сервера, их трансформация и автоматическая конвертация в JSON.

Именно для взаимодействия с сервером и была выбрана данная технология.

**React-router-dom** — решение для переключения и маршрутизации страниц в нашем приложении.

**Mobx** — это библиотека, дающая разработчикам инструмент для глобального использования переменных и методов между разными компонентами. Глобальный стор упрощает передачу параметров между компонентами, изменение которых вызывает ререндер активного интерфейса.

Также в сторах удобно хранить общие методы — будь то api‑запросы или парсеры данных — которые используются в разных компонентах, но которые сложно передавать между ними по дереву компонентов.

**Mobx-react-lite** — для связи Mobx с функциональными компонентами React, а также для динамического обновления компонентов.

**Bootstrap** — это мощный, многофункциональный интерфейсный инструментарий. Этот фреймворк сильно упростит для нас вёрстку сайта.

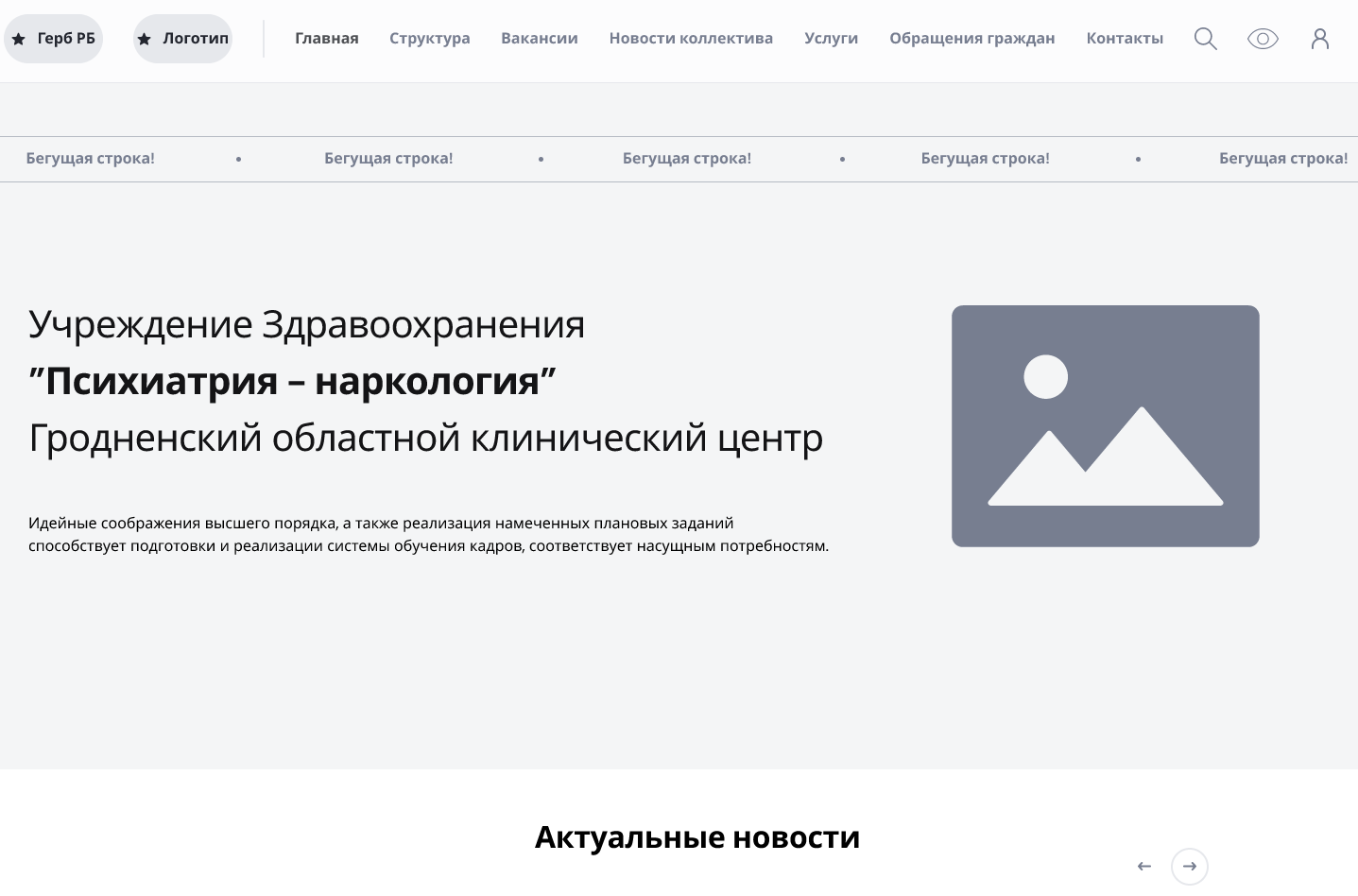
## **Проектирование внешнего вида веб-сайта**

Чтобы начать работу над самим приложением нам необходимо для начала представить, как будет выглядеть наш сайт. В этом нам помогут различные графические редакторы. Для создания прототипа и макета соответственно, наш выбор пал на Figma.

**Figma (Фигма)** — это графический онлайн-редактор для совместной работы. В нём можно создать прототип сайта, интерфейс приложения и обсудить правки с коллегами в реальном времени.

**Прототип.**  Прототип — динамическая модель взаимодействия, которая позволяет показать, как пользователи будут перемещаться по интерфейсу и взаимодействовать с различными элементами. Прототипы в Figma помогают продемонстрировать логику работы приложения или сайта, а также оценить удобство использования и взаимодействие элементов.

Используя Фигму мы создадим прототип сайта, на котором будет находиться примерное расположение элементов страницы, их размер и т.д.

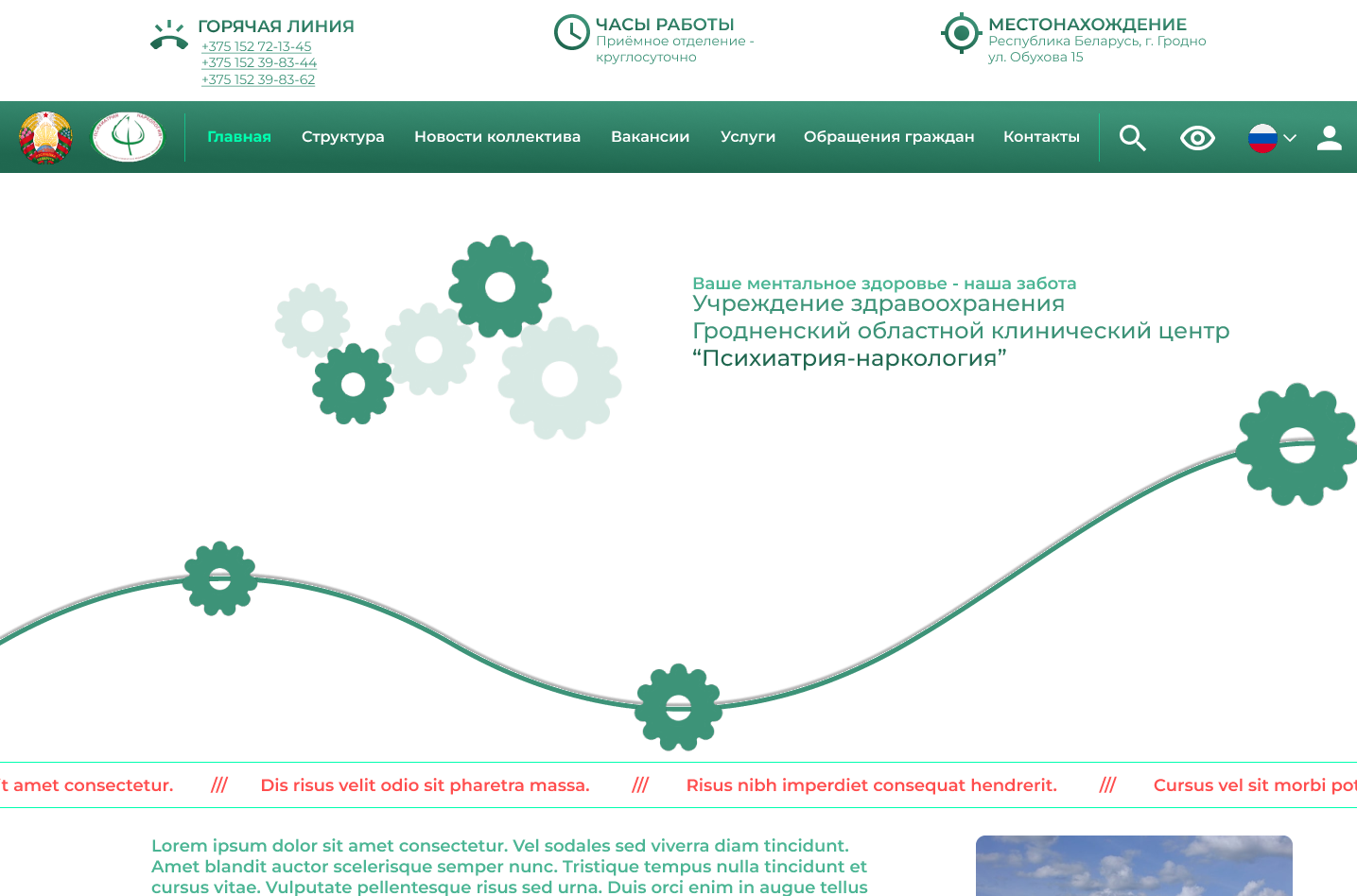


**Рисунок 2.1 - Прототип главной страницы**

Так как прототип выступает в роли примера для внешнего вида сайта, то уже сам макет может отличаться от него, но при этом стараясь сохранить расположение и вид всех элементов.

После создания прототипа мы переходим к созданию макета.

**Макет.** Макет представляет собой структурированное представление дизайна интерфейса, который создается и редактируется с помощью инструментов Figma. Макеты могут включать в себя различные элементы, такие как текстовые блоки, изображения, кнопки, формы и другие компоненты, которые используются для создания пользовательского интерфейса веб-сайтов, приложений или других цифровых продуктов.



**Рисунок 2.2 - Макет главной страницы**

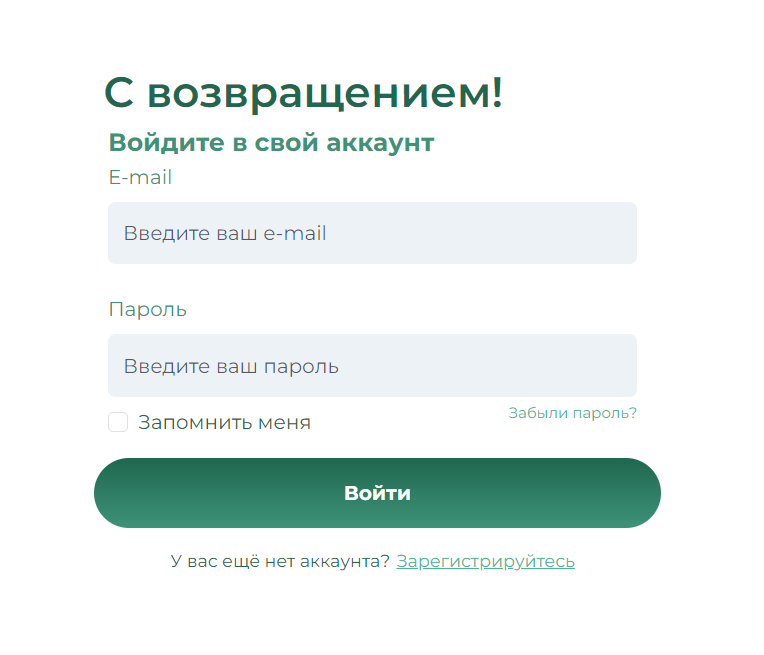
Макет уже имеет в себе все нужные стили и шрифты для последующего начала разработки самого приложения, так как мы будем создавать внешний вид сайта полностью основываясь на готовом макете.

## **Проектирование взаимодействия с авторизацией**

### **2.3.1 Проектирование авторизации**

При запуске приложения пользователю присваивается роль неавторизованного пользователя, и он сразу оказывается на главной странице приложения. В навигационной панели, находящейся в верхней части веб-сайта, присутствует значок в виде человечка, при нажатии на который пользователь попадает на страницу авторизации. На данной странице есть два текстовых поля (для e-mail и пароля), кнопка, для попытки авторизации, а также существует кнопка для перехода на экран регистрации. Кроме того, при введении неверных данных предусмотрен вывод на экран соответствующих ошибок. Экран авторизации продемонстрирован на рисунке 2.3.

При нажатии на кнопку “Войти” происходит запрос на сервер с попыткой авторизоваться. На backend приложении происходит проверка на корректность e-mail и пароля. Если данные верны, сервер генерирует токены для подтверждения авторизации, после чего отправляет эти токены пользователю. В противном случае сервер генерирует и отправляет клиенту сообщение об ошибке.

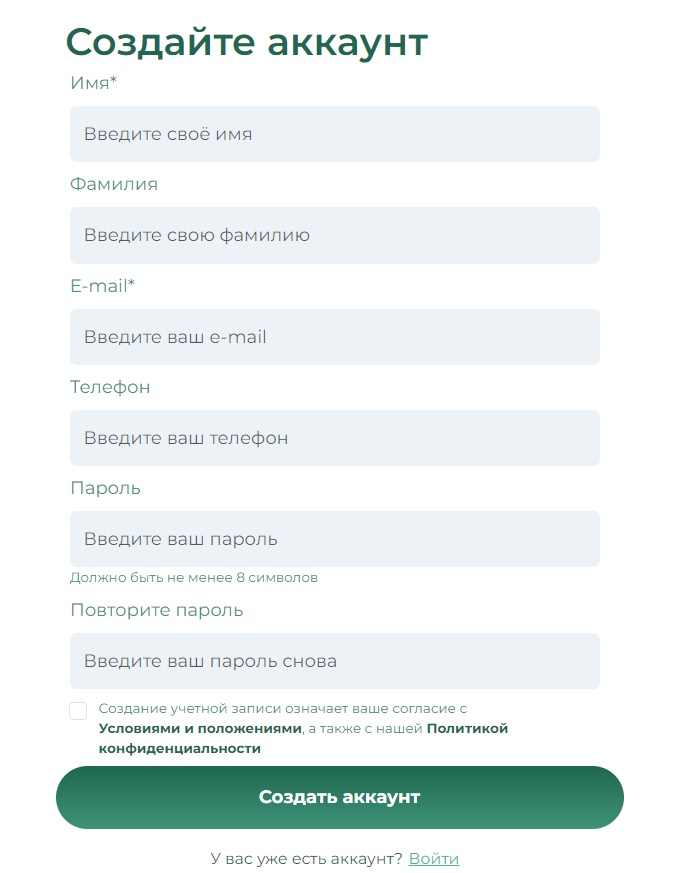


**Рис. 2.3 – Страница авторизации**

### **2.3.2 Проектирование регистрации**

На окне регистрации находятся шесть текстовых полей (для e-mail, имени, фамилии, телефона, пароля и подтверждения пароля), кнопка, для попытки регистрации, а также существует кнопка для перехода на экран авторизации. Кроме того, при введении неверных данных предусмотрен вывод на экран соответствующих ошибок. Экран регистрации продемонстрирован на рисунке 2.6.

При нажатии на кнопку “Создать аккаунт” происходит запрос на сервер с попыткой регистрации. На backend приложении происходит проверка на корректность всех введённых данных. Если данные верны, сервер генерирует hash-значение из пароля и создает пользователя заменяя пароль данным значением, после чего отправляет клиенту ответ об успешном выполнении. В противном случае сервер генерирует и отправляет клиенту сообщение об ошибке.

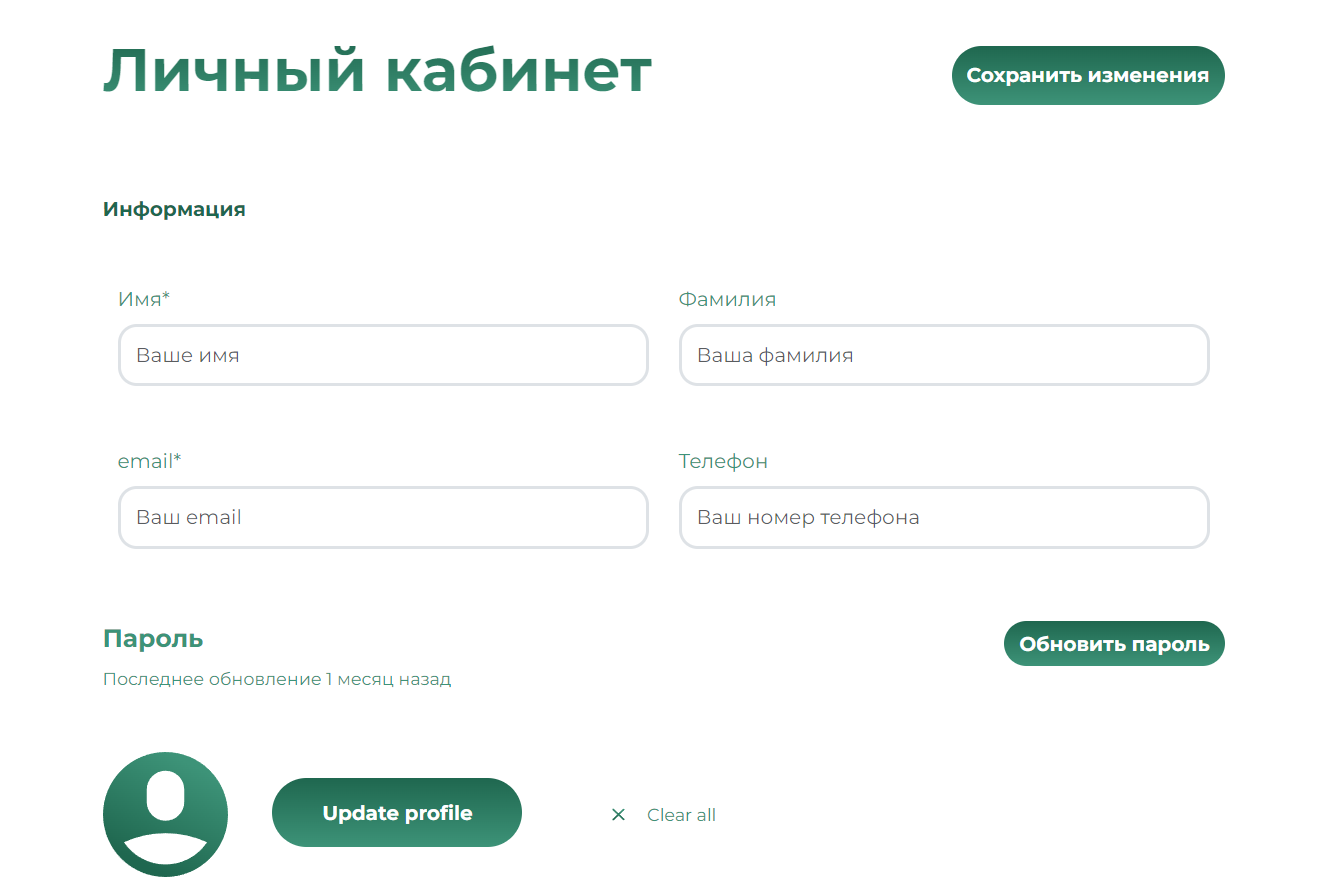


**Рис. 2.4 – Страница регистрации**

## **Проектирование изменения информаций пользователя**

В приложении предусмотрена возможность изменения информации об аккаунте. Данная возможность реализована во вкладке “Личный кабинет”.

Через навигационную панель авторизованный пользователь может попасть в личный кабинет (рис 2.7). На данной странице расположена форма настроек аккаунта. Поля в данной форме доступны для изменения и способны выводить соответствующие ошибки. На данном окне также находится кнопка для изменения изображения аккаунта, которая позволяет пользователю загрузить изображение на сервер.



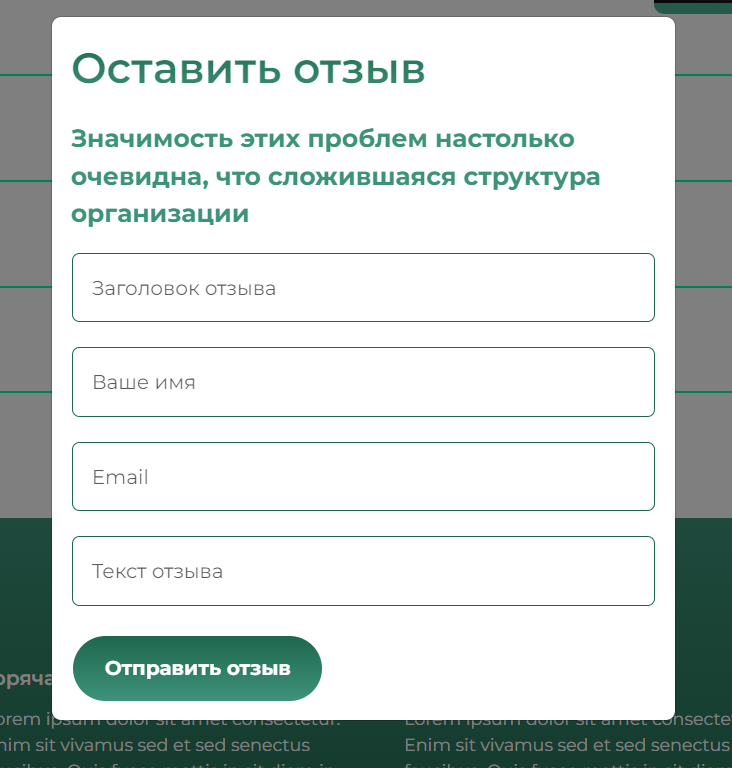
**Рис. 2.5 – Личный кабинет пользователя**

После нажатия на кнопку “Сохранить изменения”, клиентская часть отправляет запрос на обновление данных соответствующей формы. После этого сервер получает данный запрос и производит обновление на основе полученных данных. При возникновении ошибки сервер генерирует и отправляет на клиент соответствующую ошибку.

## **Проектирование публикации отзывов, благодарностей и вопросов**

### **2.5.1 Проектирование публикации отзывов**

В приложении предусмотрена возможность авторизованным пользователям оставлять отзывы при помощи формы. В форме отзывов находится 4 текстовых поля (заголовок отзыва, имя, email и текст). Кроме того, при введении неверных данных предусмотрен вывод на экран соответствующих ошибок. Окно для отправки отзыва продемонстрировано на рисунке 2.6.

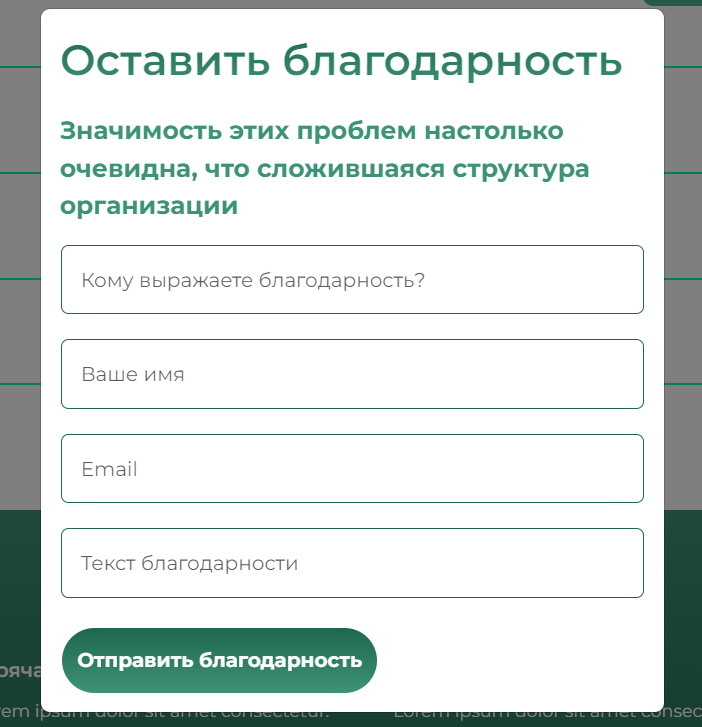


**Рис. 2.6 – Окно для отправки отзыва**

После нажатия на кнопку “Отправить отзыв” клиентская часть отправляет запрос на добавление данных соответствующей формы. После этого сервер получает данный запрос и создаёт строку в таблице на основе полученных данных. При возникновении ошибки сервер генерирует и отправляет на клиент соответствующую ошибку.

### **2.5.2 Проектирование публикации благодарностей**

Как и форма для отправки отзывов, в приложении также предусмотрена возможность авторизованным пользователям оставлять благодарность при помощи формы. В форме благодарности находится 4 текстовых поля (кому выражается благодарность, имя, email и текст). При введении неверных данных предусмотрен вывод на экран соответствующих ошибок. Окно для отправки благодарности продемонстрировано на рисунке 2.7.

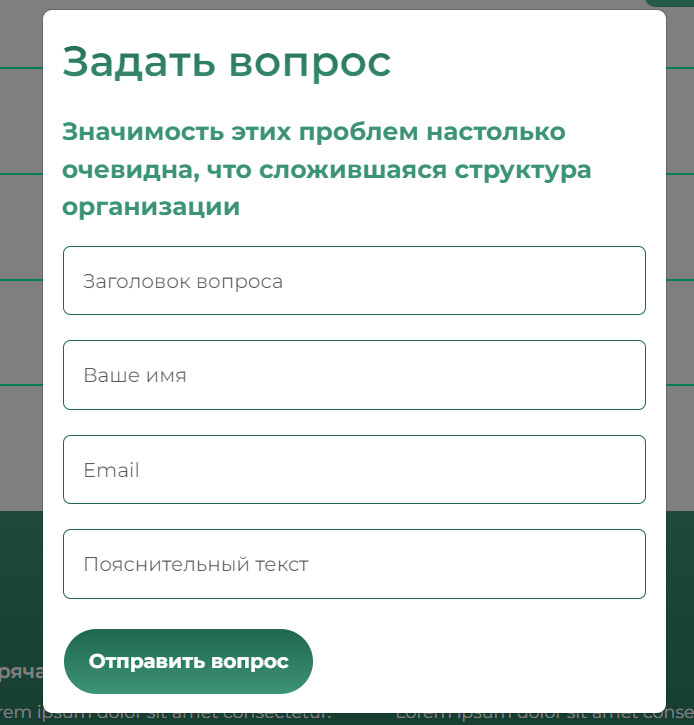


**Рис. 2.7 – Окно для отправки благодарности**

После нажатия на кнопку “Отправить благодарность” клиентская часть отправляет запрос на добавление данных соответствующей формы. После этого сервер получает данный запрос и создаёт строку в таблице на основе полученных данных. При возникновении ошибки сервер генерирует и отправляет на клиент соответствующую ошибку.

### **2.5.3 Проектирование публикации вопросов**

Как и остальные формы публикации, в приложении также предусмотрена возможность авторизованным пользователям оставлять вопрос к персоналу при помощи формы. В форме вопроса находится 4 текстовых поля (заголовок вопроса, имя, email и пояснительный текст). При введении неверных данных предусмотрен вывод на экран соответствующих ошибок. Окно для отправки благодарности продемонстрировано на рисунке 2.8.



**Рис. 2.8 – Окно для отправки вопроса**

После нажатия на кнопку “Отправить вопрос” клиентская часть отправляет запрос на добавление данных соответствующей формы. После этого сервер получает данный запрос и создаёт строку в таблице на основе полученных данных. При возникновении ошибки сервер генерирует и отправляет на клиент соответствующую ошибку.

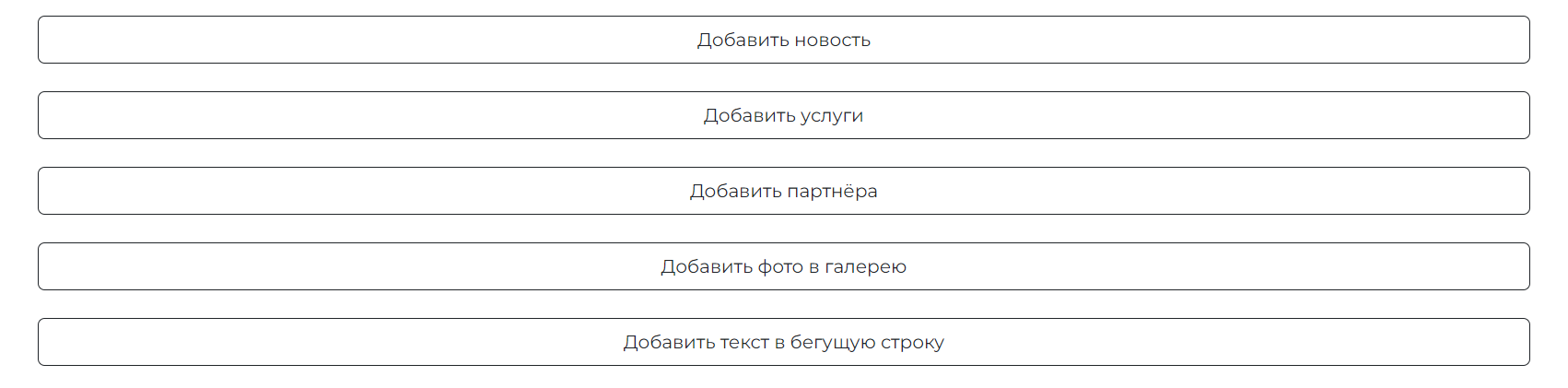
.

## **2.6 Проектирование пагинации**

### **2.6.1 Проектирование создания данных при помощи админ-панели**

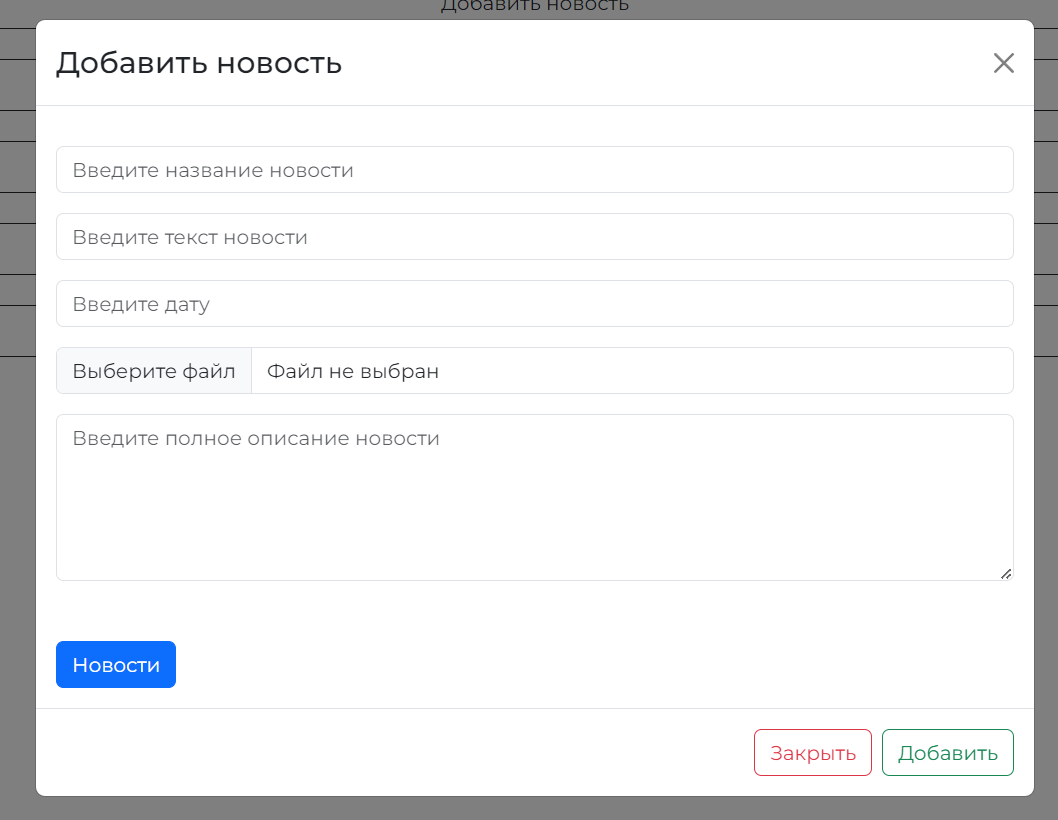
В нашем проекте пагинация используется практически в каждой вкладке, в той или иной степени. Для организации данной функции будут использованы следующие инструменты: глобальное хранилище Mobx и axios. Для добавления элементов пагинации определенного блока, будь то новости или публикации, в нашем приложении предусмотрена админ-панель. Так как блоков, в которые добавляются элементы, достаточно много, было принято решение разделить админ-панель на компоненты, каждый из которых отвечает за свою страницу, в блоки которой будет добавляться информация.

Страница админ-панели имеет в себе одну или несколько кнопок. Количество кнопок зависит от страницы, за которую она отвечает (рисунок 2.9)



**Рис. 2.8 – Админ-панель для главной страницы**

На данной админ-панели есть 5 кнопок, каждая из которых отвечает за открытие формы (рисунок 2.9), которая, в свою очередь, отвечает за загрузку данных в БД для соответствующей таблицы.



**Рис. 2.9 – Форма для загрузки данных в БД для таблицы новостей**

В форме для загрузки новостей есть 5 полей для ввода (4 текстовых и 1 для файла). Поле для файла предполагает загрузку изображения на сервер.

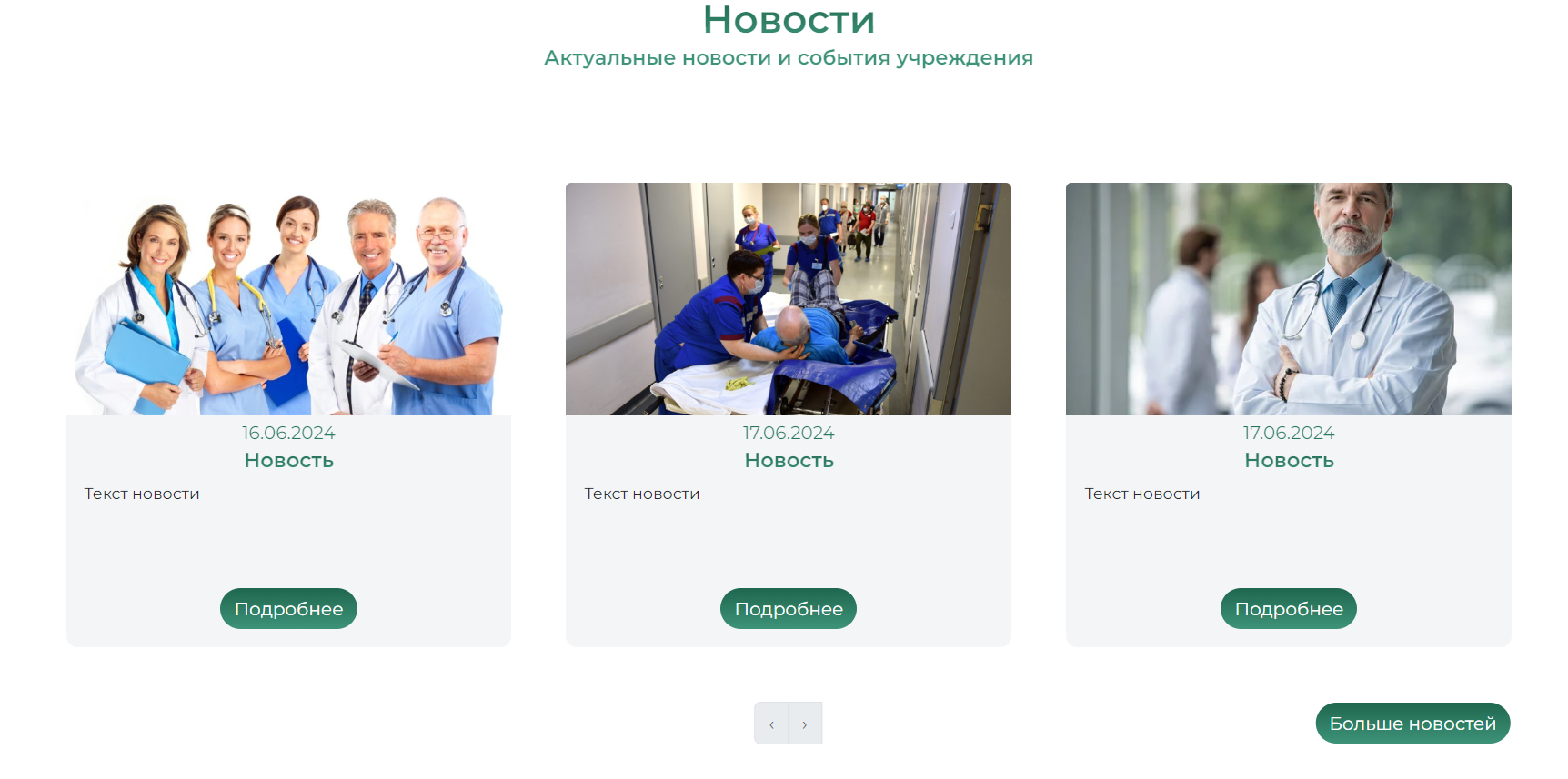
После нажатия на кнопку “Добавить” клиентская часть отправляет запрос на добавление данных соответствующей формы. В это же время изображению генерируется случайное имя и само изображение попадают в статическую папку, расположенную на сервере.

После этого сервер получает данный запрос и создаёт строку в таблице на основе полученных данных. При возникновении ошибки сервер генерирует и отправляет на клиент соответствующую ошибку.

В приложении предусмотрено то, что доступ к админ-панели могут иметь только администратор и модераторы. Кнопка для перехода в админ-панель также видна только этим ролям.

### **2.6.2 Проектирование отображения элементов пагинации**

Для того, чтобы пагинация корректно работала, нужно сначала ограничить количество отображаемых элементов, приходящих нам от сервера. Ограничение выставляется в контроллере для соответствующей таблицы на сервере, в глобальном хранилище и в запросе от клиента. После отправки запроса с клиента на получение данных от сервера, мы в соответствующем компоненте для пагинации присваиваем полученные данные в наше глобальное хранилище и уже после прописываем рендер для каждого элемента объекта нашей глобальной переменной. Ранее установленные ограничения позволяют нам увидеть соответствующее количество элементов пагинации на странице (рисунок 2.10)



**Рис. 2.10 – Пагинация новостей**

Также, при нажатии на кнопку “Подробнее” появляется модальное окно, в котором отображается вся информация соответствующей новости, только вместо короткого её описания отображается полное, которое ранее добавлялось через админ-панель (рисунок 2.11).



**Рис. 2.11 – Подробная информация новости**

## **2.7 Вывод по главе 2**

Во второй главе были спроектирована общая архитектура приложения, а именно архитектура её клиентской части. Были спроектированы основные возможности приложения и выбраны способы их реализации. После того, как были спроектированы основные возможности приложения можно приступать к их реализации программным кодом.

В качестве среды разработки приложения будут использована Visual Studio Code, а в качестве программной платформы - Node.js. Для создания frontend части будет использован Фреймворк React. Используемый язык программирования – JavaScript

# **ГЛАВА 3**

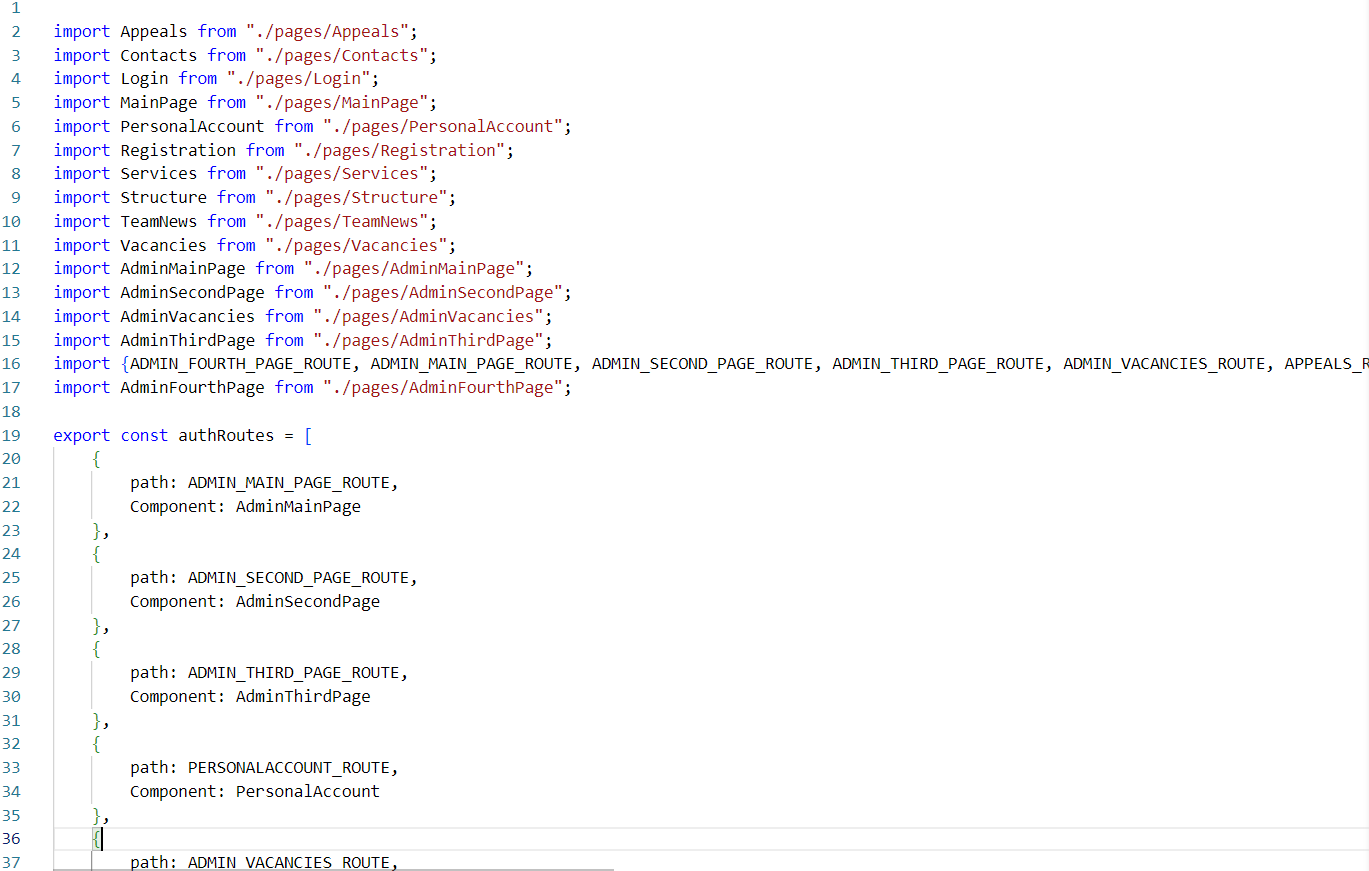
# **ПРОГРАММНАЯ РЕАЛИЗАЦИЯ КЛИЕНТСКОЙ ЧАСТИ ВЕБ-САЙТА**

## **Создание общей структуры приложения**

## **3.1.1 Навигация и роутинг**

Навигация по страницам является неотъемлемой частью любого веб-сайта, она позволяет переходить с одной страницы на другую, отображая их отдельно друг от друга.

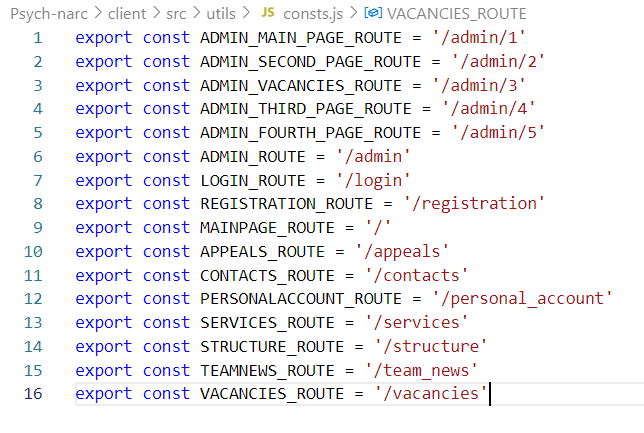
Для начала мы создали файл routes.js (рисунок 3.1) с путями к нашим страницам, в котором будет находится список маршрутов, разделённый на маршруты для авторизованных и неавторизованных пользователей соответственно



**Рис. 3.1 – Файл с путями к страницам routes.js**

У нас имеются лист с маршрутами для авторизованных пользователей authRoutes и лист с публичными маршрутами publicRoutes. Каждый маршрут имеет в себе путь и компонент страницы, к которому он будет вести.

Также все пути мы преобразовали в константы и поместили в файл consts.js, показанный на рисунке 3.2.



**Рис. 3.2 – Файл с константами путей consts.js**

Саму же навигацию мы реализуем в файле AppRouter.js, показанный на рисунке 3.3. Импортируем массив с роутами, затем пробегаемся по нему с помощью функции map и вытаскиваем из компонента путь и компонент, и для каждого элемента массива мы отрисовываем роут, где указываем путь и компонент, к которому он привязан.

Также указываем ключ, так как каждый элемент массива должен быть уникальным. Еще присутствует проверка, которая выполняет код только для авторизованных пользователей, но в ином случае пропускается. И код, который в любом случае будет отображать публичные роуты. Также мы получили глобальную переменную user из нашего хранилища, в которой и содержится информация о том, авторизован пользователь или нет.

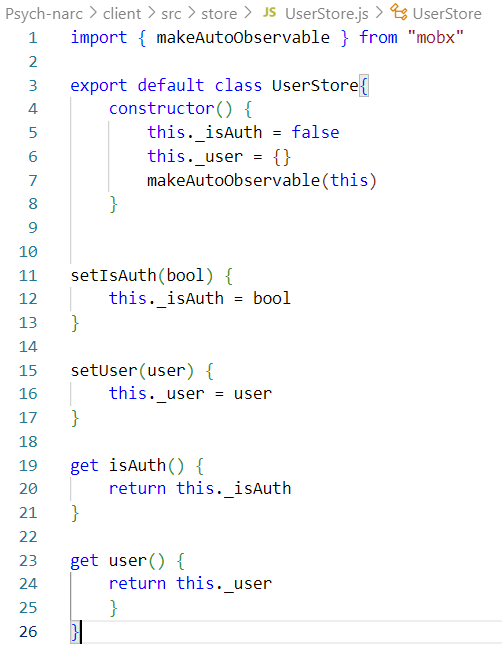


**Рис 3.3 – Логика навигации по страницам AppRouter.js**

## **3.1.2 Глобальное хранилище**

Mobx позволит нам реализовать глобальное хранилище для хранения и обновления данных о пользователях, которые мы можем получить из любой точки нашего приложения.

Создадим файл UserStore.js (рисунок 3.4), в котором прописываем одноимённый класс, в котором мы создаём коструктор, который будет вызываться при создании объекта данного класса. Прописываем две неизменяемые переменные и вызываем функцию makeAutoObservable. Так Mobx будет следить за изменениями этих переменных и при их изменении будут перерендериваться компоненты, которые используют глобальную переменную user. Также создаём action’ы, т.е. функции, которые изменяют состояние наших переменных. Еще создаём computed функции, чтобы получать переменные из нашего состояния.

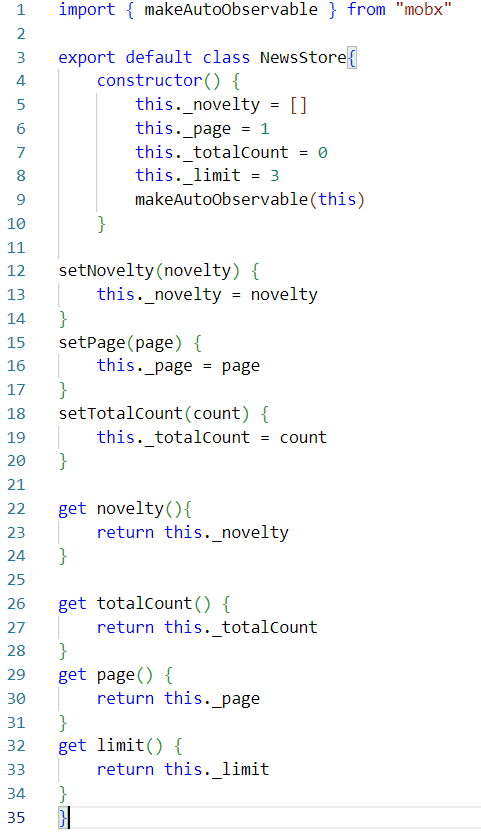


**Рис 3.4 – Глобальное хранилище UserStore.js**

По похожей логике реализуются хранилища для нашей пагинации.

Только в хранилищах пагинации, помимо переменной novelty, в которой будут хранится все данные строки таблицы, которая нам пришла с сервера, также присутствуют переменные, которые позволяют нам ограничить отображение элементов пагинации.

На рисунке 3.5 продемонстрировано хранилище для новостей NewsStore.js, в котором переменная page отвечает за текущую страницу пагинации, totalCount за общее количество элементов, которые были получены от сервера, а также переменная limit, которая как раз и ограничивает отображение элементов пагинации.



**Рис 3.5 – Глобальное хранилище NewsStore.js**

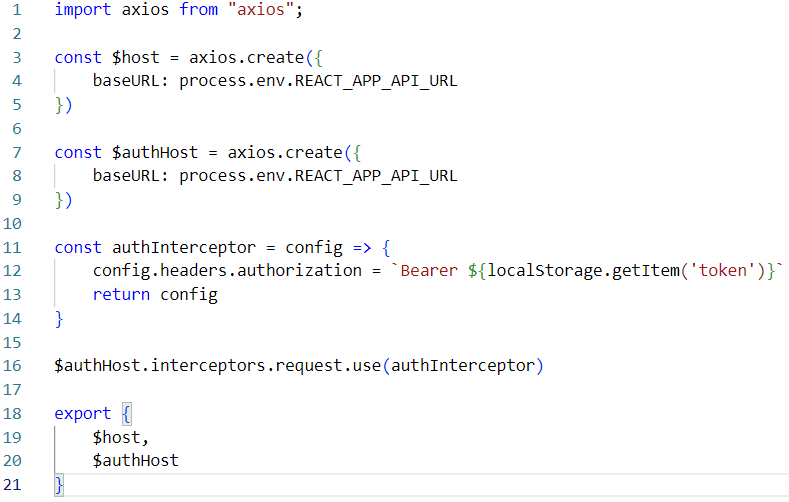
Все хранилища мы импортируем в основной файл приложения index.js в качестве новой глобальной переменной (рисунок 3.6).



**Рис 3.6 – Объявление глобальных переменных в файле index.js**

## **3.1.3 Настройка axios**

Для реализации подключения к back-end части приложения мы будем использовать Axios. Его настройка будет произведена в файле index.js папки http , показанным на рисунке 3.6.



**Рис 3.6 – Настройка axios в файле index.js**

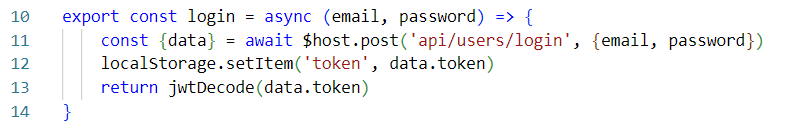
Создаём файл index.js и настраиваем сам axios. Создаём два инстанса: первый ($host) будет для обычных запросов, которые не требуют авторизации, кто второму запросу ($authHost) автоматически будет подставляться хедер authorization и туда будет добавляться токен. В опциях к инстансу указывается URL на который отправляются запросы. В файле .env прописана системная переменная с API к серверу, которую мы передаём в качестве URL в инстанс.

В случае второго инстанса мы должны создать интерцептор, который будет автоматически подставлять токен к каждому запросу. В конфиге в поле headers добавляем хедер authorization и указываем наш токен, который мы будем получать из локального хранилища по ключу token. При авторизации мы его добавляем в наше локальное хранилище. Затем для инстанса $authHost мы добавляем интерцептор на запрос, который будет отрабатывать перед каждым запросом и подставлять токен в хедер authorization

## **Реализация авторизации, регистрации, изменения и проверки пользователя**

### **3.2.1 Реализация авторизации**

Реализация клиенткой части авторизации находится в компоненте Login и в файле userAPI. Рассмотрим процесс авторизации, который представлен на рисунке 3.4.



**Рис. 3.7 – Функция с запросом для авторизации**

Создаём стрелочную функцию для авторизации, которая в качестве параметров принимает требуемые переменные. В переменную data будем помещать ответ, который будет возвращаться от сервера, а именно POST запрос, в котором базовый URL берётся из системной переменной, и добавляем к нему наши эндпоинты, указанные на стороне сервера. Как тело запроса указываем передаваемые параметры. Если запрос прошёл успешно, из ответа data извлекается токен data.token. Этот токен сохраняется в localStorage под ключом 'token'. После этого токен декодируется при помощи библиотеки jwtDecode.

Далее мы рассмотрим компонент Login на рисунке 3.8.



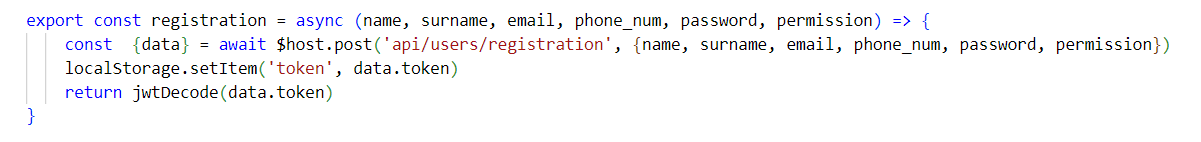
**Рис. 3.8 – Авторизация в компоненте**

Функция signIn вызывается при попытке пользователя войти в систему.

Эта функция вызывает функцию login из нашего userAPI файла, и в случае успеха обновляет состояние пользователя user.setUser(user) и user.setIsAuth(true), а затем перенаправляет его на главную страницу. Проверка также проходит и на стороне сервера. Переменные email и password передаются из формы авторизации.

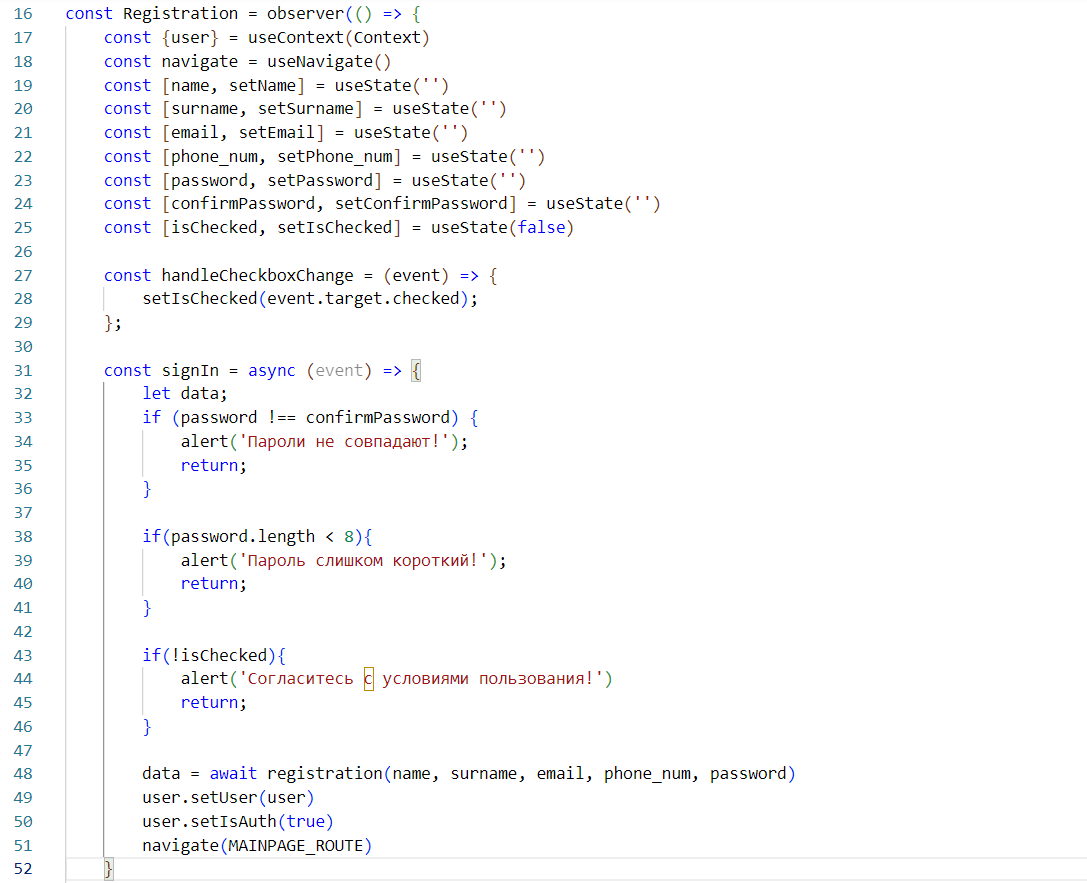
### **3.2.2 Реализация регистрации**

При регистрации процесс аналогичен с авторизацией, но с одним отличием: больше передаваемых параметров. На рисунке 3.9 можно увидеть функцию registration в файле userAPI.js.



**Рис. 3.9 – Функция с запросом для регистрации**

Теперь рассмотрим процесс регистрации в компоненте Registration на рисунке 3.10.



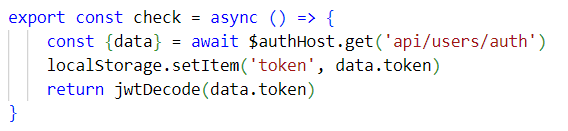
**Рис. 3.10 – Процесс регистрации в компоненте**

Аналогично с процессом авторизации функция signIn вызывается при попытке пользователя зарегистрироваться в системе.

Эта функция вызывает функцию registration из нашего userAPI файла, и в случае успеха обновляет состояние пользователя user.setUser(user) и user.setIsAuth(true), а затем перенаправляет его на главную страницу. Проверка также проходит и на стороне сервера. Переменные, передаваемые в запросе, получаются из формы регистрации.

### **3.2.3 Реализация проверки пользователя**

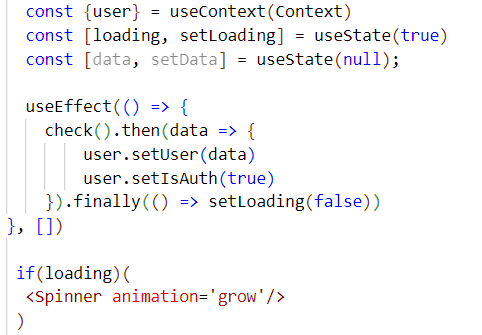
Теперь рассмотрим функцию проверки пользователя, которая делится на функцию check в userAPI и хук useEffect в файле App.js. Для начала взглянем на функцию с запросом на рисунке 3.11.



**Рис. 3.11 – Функция с запросом для проверки пользователя**

Аналогично с остальными двумя функциями, эта функция также отправляет запрос на сервер по эндпоинту и сохраняет и декодирует токен в сулчае успеха. Единственное отличие в том, что эта функция не принимает никаких параметров.

Теперь перейдём к процессу проверки пользователя на рисунке 3.12.

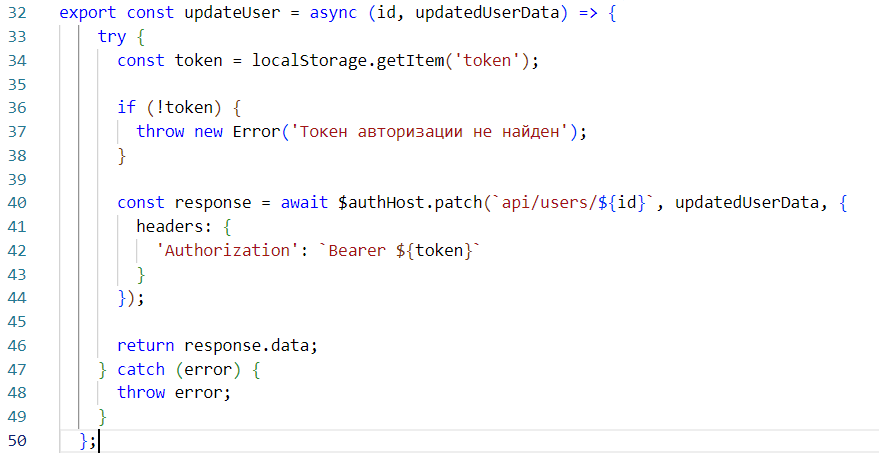


**Рис. 3.12 – Процесс проверки пользователя**

В данном случае используется хук useEffect, который вызывается единожды при обновлении страницы, который вызывает функцию check, которая при успешном выполнении опрокидывает данные в нашу глобальную переменную и делает нашего пользователя авторизованным, завершая загрузку.

### **3.2.4 Реализация изменения пользователя**

Как обычно начнём с рассмотрения функции updateUser в файле userAPI на рисунке 3.13.



**Рис. 3.13 – Функция изменения пользователя**

Эта функция в качестве параметров принимает id пользователя и объект с его обновлёнными данными updatedUserData. Внутри самой функции извлекается токен авторизации для аутентификации запроса. Если токен не найден, функция выбрасывает ошибку “Токен не найден”. Если токен найден, то она отправляет PATCH запрос на сервер по авторизованному (authHost) эндпоинту. В запросе передаются: id пользователя, объект с обновлёнными данными updatedUserData и в заголовке передаётся токен авторизации в формате Bearer ${token}. Если запрос успешно выполнен, функция возвращает данные ответа (response.data), если нет, то опрокидывает ошибку. Теперь же мы рассмотрим функцию изменения пользователя внутри компонента на рисунке 3.14.

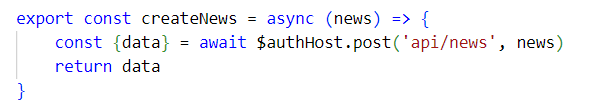


**Рис. 3.14 – Функция изменения пользователя в компоненте**

В этом компоненте мы сначала достаём нашу глобальную переменную из контекста. Затем мы инициализируем локальное состояние компонента используя хук useState, которое будет меняться от введенных пользователем данных в форме. Каждое состояние инициализируется значением из глобальной переменной user. Функция updateUserInfo выполняет следующие действия: создает объект updatedUserData с текущими значениями email, name, surname и phone\_num из состояния компонента. Вызывает функцию updateUser из нашего userAPI файла, передавая id пользователя и обновленные данные. Если обновление прошло успешно, обновляет пользователя в контексте с помощью user.setUser(updatedUser) и показывает сообщение об успешном обновлении. Если произошла ошибка, показывает сообщение об ошибке.

## **Реализация добавления данных в БД при помощи админ-панели**

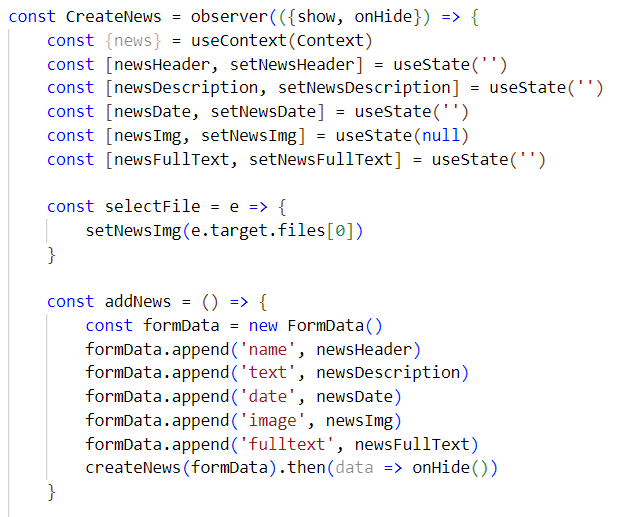
В нашем приложении, чтобы добавлять данные в БД, обращаясь к серверу, мы реализовали админ-панель, которая даёт возможность нам это сделать. На рисунке 3.13 продемонстрирован файл newsAPI, в котором есть функция createNews, которая отправляет POST запрос на сервер для создания строки с данными о новости.



**Рис. 3.15 – Функция создания новости**

Эта функция принимает в себя переменную news, которая расположена на сервере, а также отправляет POST запрос по авторизованному (authHost) эндпоинту на сервер. В случае успеха при помощи деструктуризации объекта из ответа запроса извлекается свойство data, которое содержит данные, возвращенные сервером.

Теперь рассмотрим функцию для создания новости в админ-панели на рисунке 3.14.



**Рис. 3.16 – Функция создания новости в админ-панели**

Для начала мы импортируем нашу глобальную переменную из хранилища, далее используем useState для различных полей новости. Функция selectFile используется для обновления состояния с выбранным файлом, когда пользователь выбирает файл. newsHeader, newsDescription, newsDate, newsImg и newsFullText мы получаем от пользователя из полей формы. Функция addNews создаёт объект formData и добавляет различные поля новости (которые должны соответствовать полям таблицы, в которую мы пытаемся записать данные) в formData, далее она вызывает функцию createNews из нашего newsAPI файла c formData в качестве аргумента. После успешного создания новости модальное окно с созданием новости закрывается.

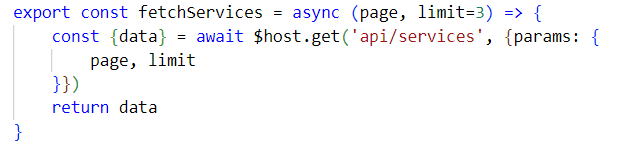
Функции создания строк в таблицы при помощи админ-панели мало чем отличаются друг от друга, именно поэтому не вижу смысла перечислять их все.

## **Реализация пагинации**

Пагинация — это метод разделения большого количества данных на более мелкие части, чтобы улучшить производительность и удобство использования. В нашем случае пагинация используется для отображения различных новостей, публикаций, доски почёта и т.д. Рассмотрим пагинацию на примере услуг.

### **3.4.1 Реализация функции получения данных для пагинации**

Для начала нам стоит рассмотреть функцию fetchServices в нашем servicesAPI файле на рисунке 3.15.



**Рис. 3.17 – Функция получения данных для пагинации**

Данная функция в качестве параметров принимает две переменные: page и limit, которые отвечают за текущую страницу пагинации и максимальное количество элементов на странице соответственно. Также эта функция отправляет GET запрос по эндпоинту, находящемуся на сервере, стоит заметить, что в параметрах запроса тоже находятся переменные page и limit с таким же назначением, только полученные от сервера. Далее эта функция при помощи деструктуризации объекта из ответа запроса извлекает свойство data, которое содержит данные, возвращенные сервером. В конце функция возвращает полученные данные.

Теперь мы переходим к получению данных и их последующему отображению на клиенте. Рассмотрим файл PopularServices.js, показанный на рисунке 3.16.



**Рис. 3.18 – Функция получения данных для пагинации из запроса**

В этой функции мы для начала получаем нашу глобальную переменную из контекста. Далее создаём useState для модальных окон, которые будут открываться для конкретной услуги, а также создаём массив pages, в котором будет храниться количество страниц пагинации. Теперь мы используем хук useEffect, который будет перерендериваться каждый раз, когда массив его зависимостей перетерпел изменения (в нашем случае в качестве зависимостей выступают переменные services.page и services.limit). Он вызывает функцию fetchServices для получения данных о сервисах с указанной страницы и лимита, а затем обновляет состояние сервисов и общее количество сервисов в контексте.

totalPages вычисляется как округленное вверх отношение общего количества сервисов к лимиту сервисов на странице. При помощи цикла for pages заполняется номерами страниц от 1 до totalPages. Функция handlePageChange обновляет текущую страницу в контексте при изменении страницы. Функция handlePageChange обновляет текущую страницу в контексте при изменении страницы. Функция handleServiceClick устанавливает выбранный сервис и показывает модальное окно при клике на сервис. Фунцкия handleCloseModal скрывает модальное окно при его закрытии.

### **3.4.2 Реализация отображения элементов пагинации**

Теперь нам нужно отобразить всю полученную информацию на клиенте. Рассмотрим далее всё тот же компонент PopularServices.js на рисунке 3.16.



**Рис. 3.19 – Отображение элементов пагинации**

В продолжении нашего компонента мы создаём контейнер с компонентом Pagination в начале. В этом компоненте мы добавляем стрелочку влево (назад) и вешаем на неё событие onClick, которое при нажатии будет нас возвращать на одну страницу назад в пагинации. Далее мы создаём Row, каждая колонка которого является элементом пагинации. Функция map позволяет нам создать новый массив с объектом нашей глобальной переменной. Сами элементы пагинации находятся в карточках, где: servicesItem.image – изображение услуги, servicesItem.name – название услуги, servicesItem.text – краткое описание услуги. Также для каждой карточки услуги предусмотрена кнопка “Подробнее”, при нажатии на которую открывается модальное окно всё с той же информацией, только вместо краткого описания выводится полное.

В конце контейнера мы также добавляет компонент Pagination, только в этот раз для стрелочки вправо (вперёд), при нажатии на которую мы идём на одну страницу вперёд в пагинации.

### **3.4.3 Реализация отображения страниц и модальных окон пагинации**

Продолжая рассматривать наш компонент с пагинацией, мы переходим к отображению самих страниц пагинации на рисунке 3.17.



**Рис. 3.20 – Отображение страниц пагинации**

В рассматриваемом фрагменте кода используется метод map для массива pages. Этот метод перебирает каждый элемент массива pages и для каждого элемента создается элемент пагинации. Потом для каждого элемента массива pages создается элемент пагинации Pagination.Item.

Атрибут key={page} используется для уникальной идентификации каждого элемента в списке. Атрибут active={services.page === page} устанавливает, активен ли данный элемент пагинации. Если текущая страница services.page равна текущему элементу массива page, то элемент будет активен. Атрибут onClick={() => services.setPage(page)} определяет обработчик события клика. При клике на элемент пагинации будет вызвана функция services.setPage(page), которая устанавливает текущую страницу в значение page. Внутри элемента Pagination.Item находится текст, который отображает номер страницы page.

Теперь мы переходим к реализации отображения модальных окон для каждого элемента пагинации на рисунке 3.18.



**Рис. 3.21 – Отображение модальных окон элементов пагинации**

В этом коде мы создаём модальное окно с атрибутами, которые отвечают за открытие и закрытие модального окна соответственно. Далее мы получаем изображение из нашей глобальной переменной и отображаем его. После этого в заголовок модального окна помещаем название услуги, которое мы также получаем из глобальной переменной. В тело модального окна помещается полное описание услуги, которое мы, аналогично, получаем из глобальной переменной. Далее мы в футер добавляем кнопку для закрытия модального окна.

## **Вывод по главе 3**

В этой главе была рассмотрена реализация ключевых методов проекта, предоставлено подробное описание функциональности приложения. Анализ показал, что у проекта есть потенциал как для практического использования, так и для академических целей.

При успешной реализации проекта, его можно будет доработать и усовершенствовать, превратив в полноценный информационный сайт. Финальная версия приложения может стать универсальным шаблоном, который поможет другим разработчикам создавать свои собственные информационные сайты.

# **ЗАКЛЮЧЕНИЕ**

В ходе выполнения дипломной работы была разработана клиентская часть веб-сайта Гродненского областного клинического центра “Психиатрия-Наркология”. Данная дипломная работа помогла в приобретении ценных навыков в создании клиентской части веб-сайта.

**Основные достижения проекта:** Основным достижением данного проекта стала разработка общей архитектуры клиентской части приложения. Вышеназванная архитектура обеспечивает надёжность и масштабируемость при дальнейшей работе над проектом. Еще одним важным достижением стала разработанная функциональность. Все созданные функции клиента соответствуют требованиям, которые были выявлены раннее в ходе анализа предметной области. Важнейшим достижением данного проекта можно назвать практическую реализацию проекта. В ходе работы над ним, были описаны и разработаны методы, обеспечивающие работу системы. Конечным результатом данной дипломной работы стала реализация клиентской части веб-сайта Гродненского областного клинического центра “Психиатрия-Наркология”, который является продуктом анализа, проектирования, а также разработки, описанным ранее.

**Дальнейшее развитие:** Дальнейшее развитие сайта предполагает добавление новых функций, а именно: локализация на другие языки (английский, белорусский), добавление поиска по сайту, интеграция API нацбанка и яндекс карт. Также может дальше расширяться и улучшаться уже имеющийся функционал.

Созданная клиентская часть веб-сайта имеет хороший потенциал для дальнейшего развития и может стать полноценным информационным ресурсом с большими объёмами информации. Разработанные методы могут быть адаптированы для использования в различных проектах, что делает этот дипломный проект примером для изучения и создания клиентской части веб-сайтов.

# **СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

1. learn.javascript.ru [Электронный ресурс]: Современный учебник JavaScript. -URL: <https://learn.javascript.ru/> (дата обращения: 05.11.2023)
2. MobX [Электронный ресурс]: MobX State Manager. -URL: <https://mobx.js.org>(дата обращения: 9.01.2024)
3. React [Электронный ресурс]: React Getting Started. -URL [https://ru.legacy.reactjs.org/docs/getting-started.html](https://ru.legacy.reactjs.org/docs/getting-started.html%20) (дата обращения: 02.01.2024)
4. codecademy [Электронный ресурс]: Learn JavaScript. -URL: <https://www.codecademy.com/learn/learn-javascript> (дата обращения: 12.01.2024)
5. reactrouter.com [Электронный ресурс]: React Router Getting Started -URL: [https://reactrouter.com/en/main/start/tutorial](https://reactrouter.com/en/main/start/tutorial%20) (дата обращения: 06.01.2024)
6. Axios [Электронный ресурс]: Axios Intro. -URL: <https://axios-http.com/ru/docs/intro>(дата обращения: 10.01.2024)
7. Bootstrap 5 [Электронный ресурс]: Build fast, responsive sites with Bootstrap. - URL: <https://getbootstrap.com> (дата обращения: 15.01.2024)
8. Bootstrap React [Электронный ресурс]: The most popular front-end framework, rebuilt for React. - URL <https://react-bootstrap.netlify.app/> (дата обращения: 16.01.2024)