**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра МО ЭВМ**

отчет

**по производственной практике**

Тема: Frontend-разработка административной панели

|  |  |  |
| --- | --- | --- |
| Студентка гр. 1304 |  | Чернякова В.А. |
| Руководитель |  | Бриток Н.С. |

Санкт-Петербург

2024

**ЗАДАНИЕ**

**на производственную практику**

|  |  |  |
| --- | --- | --- |
| Студентка Чернякова В.А. | | |
| Группа 1304 | | |
| Тема практики: Frontend-разработка административной панели | | |
| Задание на практику:  Frontend-разработка административной панели для управления парковками, клиентами, локациями, оборудованием и финансами подключенных парковок. В каждом из подпунктов должны быть определенные данные, которые можно редактировать. Панель должна быть лаконична и легко понятна в использовании. | | |
| Сроки прохождения практики: 26.06.2024 – 09.07.2024 | | |
| Дата сдачи отчета: 09.07.2024 | | |
| Дата защиты отчета: 09.07.2024 | | |
|  | | |
| Студентка |  | Чернякова В.А. |
| Руководитель |  | Бриток Н.С. |

**Аннотация**

В ходе производственной практики была разработана frontend-часть административной панели для управления парковками, клиентами, локациями, оборудованием и финансами. Разработка велась с использованием библиотек *React* и *Mantine.* Была создана страница авторизации для обеспечения безопасности доступа к панели. Запросы к серверу выполнялись с использованием *API*, что позволило оперативно получать и обновлять данные в реальном времени. Для упрощения процесса разработки и повышения эффективности были созданы шаблоны страниц и компонентов, что позволило избегать повторного создания однотипных элементов.

**Summary**

During the production practice, frontend was developed-part of the administrative panel for managing parking lots, customers, locations, equipment and finances. The development was carried out using the React and Mantine library. An authorization page has been created to ensure the security of access to the panel. Requests to the server were made using the API, which made it possible to quickly receive and update data in real time. To simplify the development process and increase efficiency, page and component templates were created, which made it possible to avoid re-creating the same type of elements.

**содержание**

[Введение 5](#_Toc170224057)

[1. Поставленные цели и задачи 6](#_Toc170224058)

[1.1. Описание целей практики 6](#_Toc170224059)

[1.2. Список задач на практику 6](#_Toc170224060)

[2. Достижение целей 7](#_Toc170224061)

[2.1. Создание страницы для входа в административную панель. 7](#_Toc170224062)

[2.2. Создание основной страницы административной панели 8](#_Toc170224063)

[2.3. Поддержка выхода из системы 10](#_Toc170224064)

[3. Использованные технологии 12](#_Toc170224065)

[Заключение 13](#_Toc170224066)

[Приложение А. Отзыв руководителя 14](#_Toc170224067)

# введение

Для упрощения работы с большими объемами данных: информацией о парковках, клиентах, сотрудников и т.п. – необходима административная панель. Удобный интерфейс административной панели позволяет администраторам быстро и эффективно находить и редактировать нужные данные, что значительно экономит время и снижает вероятность ошибок. Система авторизации обеспечивает безопасность, ограничивая доступ только авторизованным пользователям.

Целью производственной практики являлось создание такой административной панели. По достижению цели должна быть создана не только панель, включающая в себя данные для просмотра, добавления и редактирования и с возможностью фильтрации и поиска, но и страница входа для проведения авторизации.

Для достижения поставленной цели необходимо было написать поддерживаемый код для возможности дальнейшего редактирования на языке *Java Script* с применением библиотек *React* и *Mantine.*

# 1. ПОСТАВЛЕННЫЕ ЦЕЛИ И ЗАДАЧИ

## 1.1. Описание целей практики

* Создание страницы для входа в административную панель.
* Создание основной страницы административной панелей, представляющих собой таблицы с данными, а также поддерживающие поиск и фильтрацию, редактирование и добавление данных.
* Поддержка выхода из системы.

## 1.2. Список задач на практику

Для создания административной панели необходимо было ознакомиться и применить следующие технологии библиотеки *React:*

* Хуки – функция *JavaScript*, которая позволяет создавать и получать доступ к состоянию и жизненным циклам *React*.
* Компоненты и свойства – позволяют разделить пользовательский интерфейс на независимые, повторно используемые части.
* *Redux Toolkit* – набор инструментов для упрощения и ускорения процесса разработки приложений на *React* с использованием *Redux* хранилища.
* *Redux Toolkit Query –* библиотека, предоставляющая набор перехватчиков и утилит, которые помогут извлекать данные из *API* и управлять их состоянием в хранилище *Redux*.

# 2. достижение целей

## 2.1. Создание страницы для входа в административную панель.

Для страницы входа в административную панель была создана функция *Login()*,которая представляет собой компоненту *React.*

Для сбора данных, вводимых пользователем, такие как логин и пароль, был использован стандартный хук *React* – *useForm*. Данный хук позволяет проинициализировать форму, а также вернуть объект, который содержит методы и состояние для управления формой.

Для отправки данных, которые находятся в форме, необходимо было написать запрос на сервер. На данной странице для получения/отправки данных и далее использовалась библиотека *Redux Toolkit Query*.

Так как происходит процесс входа в систему – необходимо хранение токена для клиента, входящего в систему, поэтому был создан метод *getToken*. Этот метод позволяет либо из хранилища по ключу *‘token’* получить нужные данные либо сгенерировать новый.

Для успешного входа и дальнейшего нахождения в системе необходимо было настроить *fetchBaseQuery*. *fetchBaseQuery* – это функция, которая позволяет настроить базовый запрос для выполнения сетевых запросов в *API.* Настроена эта функция так, что к каждому запросу автоматически добавляется заголовок с токеном. Для этого внутри *fetchBaseQuery* настраивался параметр *prepareHeaders*, это позволило динамически настраивать заголовки для каждого запроса, в данном параметре прописана проверка на токен и устанавливаются нужные заголовки, например, авторизация и в обязательном порядке токен: *headers.set('Authorization', `${token}`).*

Далее было необходимо создать *API.* Написано это с помощью функции *createApi*, в нее передавались следующие параметры:

*reducerPath* - путь для редюсера, как и где будет храниться состояние данных. В случае с авторизацией это хранилище *'loginApi'*.

*baseQuery* – определение базового запроса. Использовался настроенный раннее *fetchBaseQuery* для добавления токенов в заголовки запросов*.*

*endpoints* – определение конечных точек *API*.

В рамках задания практики все конечные точки *API* прописывались с помощью метода *builder* для создания различных запросов. Перед написанием самого запроса, был определен *builder.mutation*, то есть настроена операция типа мутации(изменение данных).

Далее прописан сам *query* – функция, которая возвращает объект с параметрами запроса. В данном случае он включает в себя:

*url* – конечная точка для запроса. Адрес на сервере, связанный с авторизацией.

*method* – *HTTP* метод запроса, в случае авторизации это тип *POST*.

*body* – данные, которые будут отправлены в теле запроса: логин и пароль соответственно.

Для отображения страницы внутри основной функции прописан *JSX* код, который возвращает разметку страницы: заголовок, форму входа с полями для ввода *TextInput*, для которых также настроена валидация и параметр *required* для обязательного заполнения полей, кнопку для отправки формы *Button*, и стилизацию элементов с помощью встроенных стилей.

Внутри приложения административной панели также была настроена навигация. При успешном входе в систему пользователь перенаправляется на основную страницу панели.

## 2.2. Создание основной страницы административной панели

При создании основной страницы главной задачей являлось создание такой верстки, написание такого кода, чтобы его было возможно использовать как шаблон. То есть все разделы основной страницы административной панели между собой похожи: это таблицы с данными, которые можно редактировать, отфильтровать, осуществить поиск по ним и добавить что-то новое.

Сама страница представляет из себя две основные зоны – боковая панель навигации по разделам панели администрирования и основная область. Для такой реализации была написана функция *Shell()*. В функции определяется массив *options*, который содержит объекты для каждого элемента навигации. Каждый объект включает *link, label, icon* и *component* (компонент *React* для отображения при активации ссылки). Для правильного отображения боковой навигационной панели и подсветки выбранного раздела установлен обработчик событий *onClick,* который обновляет состояние переменной *active* (используется хук *useState* для возвращения/хранения состояния) при клике на ссылку. Так как в массиве, описанном раннее, описаны названия компонент, то в зависимости от нажатой ссылки в основную область подгружается компонента с названием соответствующим полю *component.*

Так как все разделы панели администрирования написаны таким образом, что используют одни и те же шаблоны, то детальнее будет разобран всего один.

Рассмотрим раздел заказы. Для него была создана функция *Orders().*

Для загрузки данных был использован хук *useEffect*, который позволяет выполнять действия при изменении состояния компонента, при начале ее рендеринга. Внутри хука прописан запрос с помощью *Redux Toolkit Query*. Он по структуре похож на запрос авторизации, описанный выше, также включает в заголовке токен, но имеет следующие отличия: *reducerPath* отсутствует (в нем нет необходимости)*,* вызов *builder.query (не требуется изменения данных), url* изменен на связанный с заказами и метод самого запроса *GET*.

Для отображения данных была создана функция *TablePattern()* с использованием *props*, это позволит использовать таблицу как шаблон, передавая в нее в качестве *props* данные из различных разделов. В данной функции также прописана работа и разметка объектов, отвечающих за фильтрацию.

Объекты фильтрации представляют собой как простой *TextInput*, ввод поиска, так и конкретные *Select*. Внутри *TextInput* используется *props.search*, то есть родительской компонентой задаются поля, по которым осуществляется поиск. Внутри фильтраций настроен обработчик событий *onChange*, при его срабатывании фиксируется *event.target.value* в случае *TextInput* для получения вводимой информации и просто event в случае *Select.* Далее вызывается метод *handleFilterChange*, который позволяет обновить список вида ключ-значение, где в качестве первого выступает поле по которому идет фильтрации, второго – значение, и также отправляется запрос на сервер с помощью *Redux Toolkit Query.* Он по структуре похож на запрос авторизации, описанный выше, также включает в заголовке токен, но имеет следующие отличия: *reducerPath* отсутствует (в нем нет необходимости)*,* вызов *builder.query (не требуется изменения данных), url* изменен на связанный с фильтрацией и метод самого запроса *GET*.

Также верстка таблиц настроена так, что при нажатии на соответствующую строчку открывается модальное окно для изменения данных. Это возможно за счет того, что при отображении данных в каждой таблице отображаются *ID*, и при нажатии на строку, оно “достается” из конкретного поля и отправляется запрос на сервер с помощью *Redux Toolkit Query*. Он по структуре похож на запрос авторизации, описанный выше, также включает в заголовке токен, но имеет следующие отличия: *reducerPath* отсутствует (в нем нет необходимости)*,* вызов *builder.query (не требуется изменения данных), url* изменен на связанный с конкретным заказом и метод самого запроса *GET*.

Также есть кнопка для добавления новых данных в таблицу, которая открывает модальное окно с формой для заполнения.

При нажатии на кнопку сохранить (в случае с изменением данных)/отправить (в случае с добавлением данных) запрос снова уходит на сервер. Вся структура запроса похожа на описанную в предыдущем абзаце, кроме замены *url* на соответствующие и в случае изменения тип запроса *PUT*, добавление – *POST*.

## 2.3. Поддержка выхода из системы

На административной панели также есть верхний Navbar, в котором находится кнопка выхода из системы. При клике на кнопку отправляется действие *logoutSuccess* через *dispatch*, которая создается на основе *useDispatch*. *useDispatch* – хук, который позволяет отправить действие в хранилище *Redux*, то есть в качестве аргумента переменной *dispatch* передается действие в хранилище. Это изменит состояние *Redux Store* и выполнит выход из приложения. Также благодаря настроенной навигации пользователь будет перенаправлен на страницу входа.

# 3. Использованные технологии

Для создания панели администрирования использовались следующие технологии: *JavaScript (JSX), React, Mantine, Redux, Redux Toolkit, CSS.* Также для контроля версий и сохранении рабочей версии при внесении новых фич использовался *git.*

# заключение

В ходе прохождения производственной практики удалось поучаствовать в работе над реальным проектом, получить опыта *frontend* разработки и возможность применить свои навыки, приобретённое время обучения в Вузе. Были изучены все технологии/библиотеки, связанные с *React*, из тех, что указаны в задачах для достижения целей. В итоге получилась административная панель, которую в дальнейшем можно расширять, так как верстка и запросы внутри разделов поддерживаются за счёт передачи данных от родительского компонента дочерним.

# приложение а отзыв руководителя

