**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра МО ЭВМ**

отчет

**по лабораторной работе №5**

**по дисциплине «Web-технологии»**

Тема:Модуль администрирования приложения «Биржа акций»

|  |  |  |
| --- | --- | --- |
| Студентка гр. 1381 |  | Васильева О. М. |
| Преподаватель |  | Беляев С.А. |

Санкт-Петербург

2023

## Цель и задачи работы.

Целью работы является изучение возможностей применения библиотеки React (https://reactjs.org/) для разработки интерфейсов пользователя webприложений и использование фреймворка NestJS (https://nestjs.com/) для разработки серверных приложений. Для достижения поставленной цели требуется решить следующие задачи:

– разработка интерфейса web-приложения;

– создание web-сервера на основе NestJS. Подготовка web-сокетов для обновления информации о стоимости у всех клиентов;

– создание каркаса клиентского web-приложения с использованием React;

– создание каркаса серверного web-приложения с использованием NestJS;

– разработка перечня компонентов; – создание статической версии интерфейса;

– определение минимального и достаточного набора состояний интерфейса;

– определение жизненного цикла состояний;

– программирование потока изменения состояний.

**Основные теоретические сведения.**

React – библиотека на JavaScript для построения интерфейса пользователя. React представляется удобным инструментом для создания масштабируемых web-приложений (в данном случае речь идет о клиентской части), особенно в тех ситуациях, когда приложение является одностраничным. В основу React заложены принципы Redux, предлагающее предсказуемый контейнер хранения состояния web-приложения. Вся структура веб-страницы может быть представлена с помощью DOM. Для решения проблемы производительности предложена концепция виртуального DOM, который представляет собой облегченную версию DOM. React работает именно с виртуальным DOM. Реализован механизм, который периодически сравнивает виртуальный DOM с реальным и вычисляет минимальный набор манипуляций для приведения реального DOM к состоянию, которое хранится в виртуальном DOM. NestJS – фреймворк для разработки серверных приложений на языках JavaScript и TypeScript. Фреймворк построен на основе компонентного подхода и предлагает стандартизованную структуру приложения по аналогии с Angular.

## Общая формулировка задание.

Необходимо создать web-приложение, обеспечивающее настройку биржи брокера, в которой есть возможность задать перечень участников, перечень акций, правила изменения акций во времени. Основные требования следующие:

1. Информация о брокерах (участниках) и параметрах акций сохраняется в файле в формате JSON.

2. В качестве сервера используется NestJS с использованием языка TypeScript.

3. Предусмотрена HTML-страница с перечнем потенциальных брокеров. Брокеров можно добавлять и удалять, можно изменить начальный объем денежных средств.

4. Предусмотрена HTML-страница для перечня акций. Есть возможность просмотреть перечень доступных акций (обозначение, название компании) и исторические данные по изменению курса не менее чем за текущий и предыдущий год. Есть возможность выбрать какие акции будут участвовать в торгах. Минимально должны поддерживаться следующие компании (в скобках – обозначение): Apple, Inc. (AAPL), Starbucks, Inc. (SBUX), Microsoft, Inc. (MSFT), Cisco Systems, Inc. (CSCO), QUALCOMM Incorporated (QCOM), Amazon.com, Inc. (AMZN), Tesla, Inc. (TSLA), Advanced Micro Devices, Inc. (AMD). Реальные исторические данные по изменению курса доступны по адресу: https://www.nasdaq.com/market-activity/quotes/historical. Фрагмент данных для AAPL за три дня (переведён в формат json, оставлены только два столбца: дата и стоимость на время начала торгов):

[{"date": "11/5/2021", "open": "$151.89"},

{"date": "11/4/2021", "open": "$151.58"},

{"date": "11/3/2021", "open": "$150.39"}]

5. Предусмотрена HTML-страница для настроек биржи (дата начала торгов, скорость смены дат в секундах при имитации торгов). На этой же странице должна быть кнопка «Начало торгов», которая запускает процесс имитации торгов и предоставление информации об изменении курсов акций всем брокерам по web-сокетам с учётом заданных настроек биржи, здесь же должна отображаться текущая имитируемая дата торгов и текущая стоимость каждой акции.

6. Все элементы в клиентском приложении реализованы с использованием компонентов React. Маршрутизация реализована с использованием «reactrouter-dom».

7. Для хранения общих данных используется Redux.

8. На сервере спроектированы компоненты и сервисы NestJS для имитации торгов и обработки запросов клиентского приложения.

9. Исторические данные по котировкам представляются как в виде таблиц, так и в виде графиков (например, с использованием Chart.js).

10. Приложение должно реализовывать responsive-интерфейс и корректно работать в том числе при просмотре с мобильного телефона.

11. Для всех страниц web-приложения разработан макет интерфейса с использованием Figma (<https://www.figma.com/>).

Преимуществом будет создание и использование аутентификации на основе passport.js (http://www.passportjs.org/).

Преимуществом будет использование Material UI React (https://mui.com/ru/).

## Выполнение работы.

1. Для сайта в качестве серверного приложения использовался фреймворк NestJS, а в качестве интерфейса пользователя web-приложения использовался React.
2. Со стороны React были написаны файлы brokers.js, Trade.js, Setting.js, Chart.js, Stock.js, StockChart.js и был прописан App.js.

В файле App.js имеется компонент кода, описывающий функцию App, которая отображает панель переходов. Внутри компонента Router задается маршрутизация для приложения. Компонент Navbar используется для создания навбара, в котором содержатся ссылки на различные разделы "Брокеры", "Акции" и "Настройки". Компонент Routes определяет маршруты, соответствующие разделам и компонентам приложения. Каждый маршрут внутри Routes определяет путь и соответствующий ему компонент. Предполагается, что каждый из компонентов (Brokers, Stock, Settings) отвечает за отображение соответствующего раздела приложения. Наконец, экспортируется компонент App, который будет использоваться в главном файле приложения. См. Рис. 1

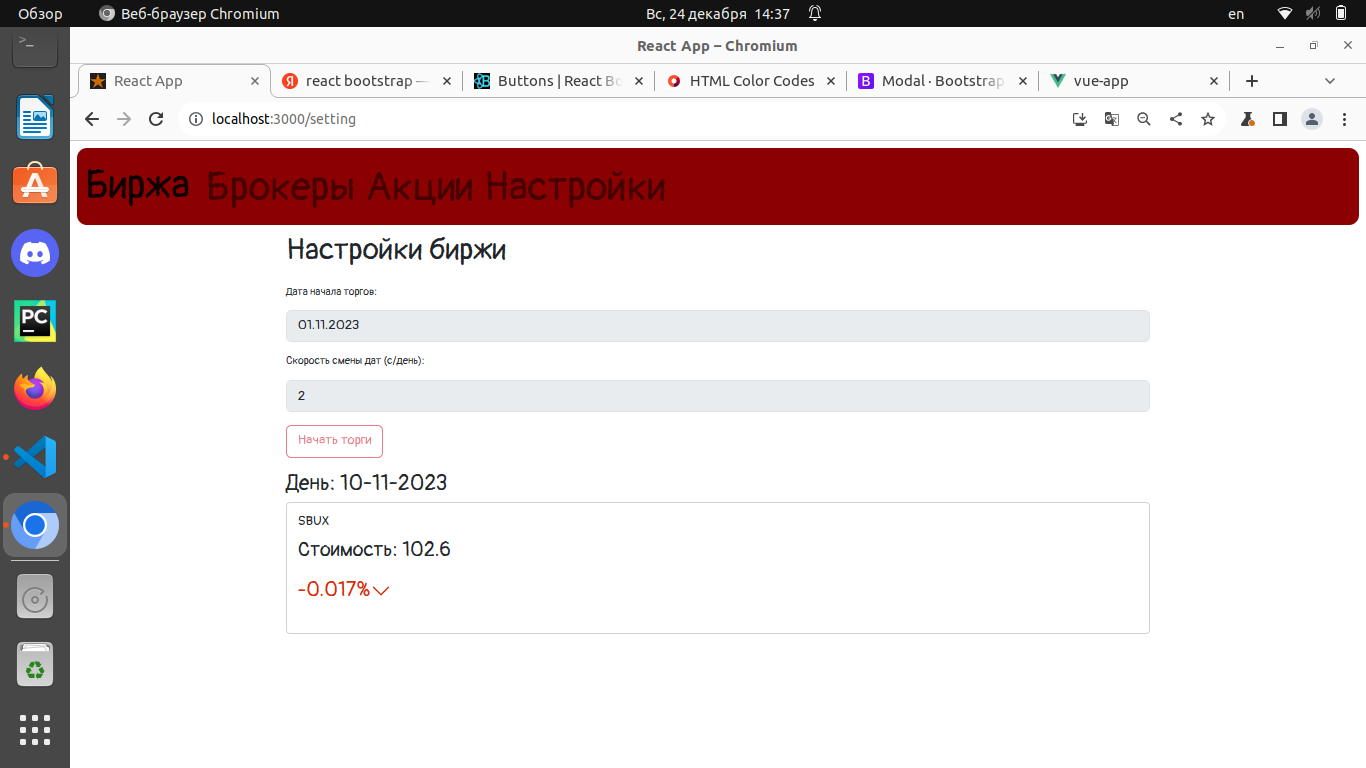


Рис. 1 — Панель переходов

В файле brokers.js определяются два компонента EditDialog и AddDialog, которые представляют собой модальное окно, для редактирования брокера и добавления нового брокера соответственно. Первый компонент (см. рис. 2): содержит форму с полями для ввода отредактированного названия и суммы брокера, а также кнопку "Сохранить". При отправке формы данные также отправляются на сервер с помощью функции fetch. Второй компонент (см. рис. 3): содержит форму с полями для ввода названия и начальной суммы брокера, а также кнопку "Добавить". При отправке формы данные отправляются на сервер с помощью функции fetch.



Рис. 2 — Окно редактирования брокера

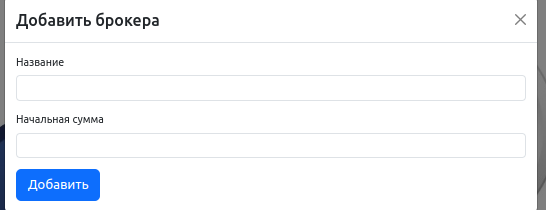
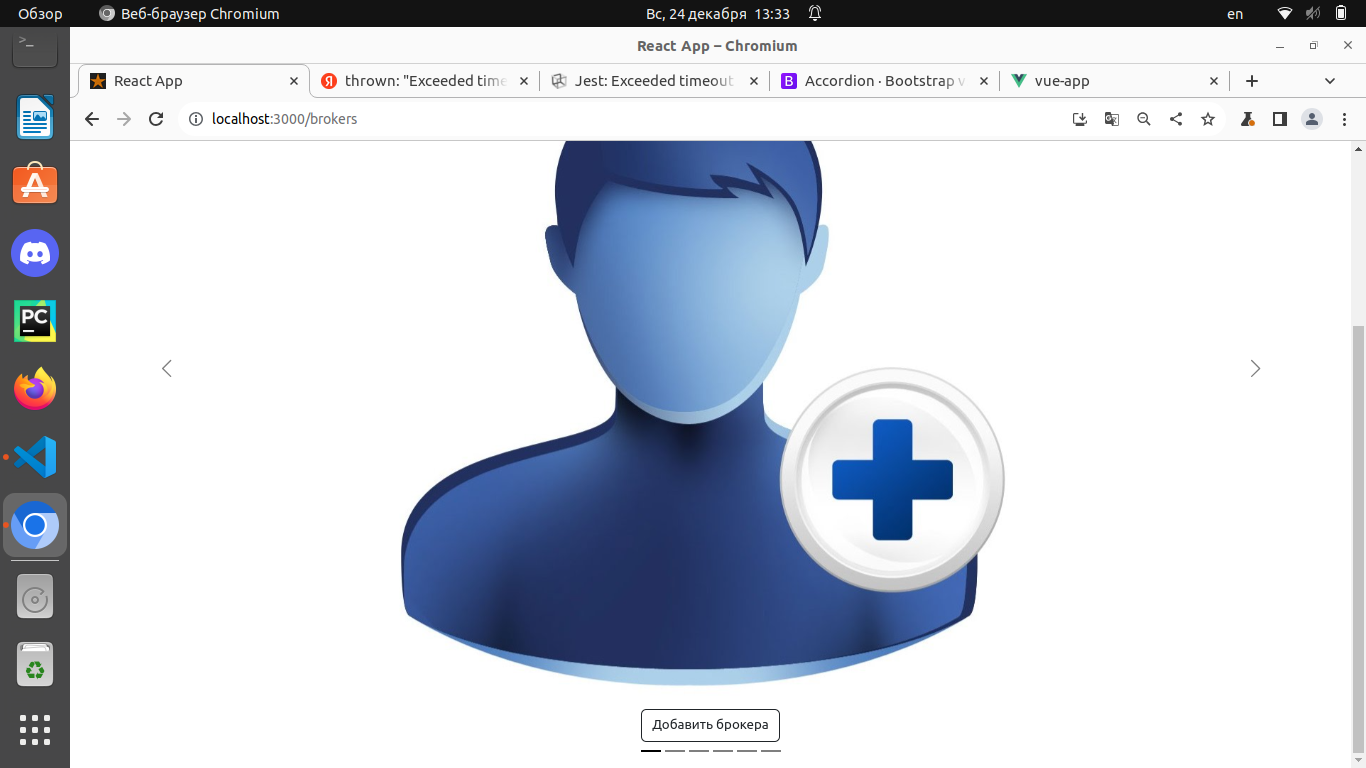
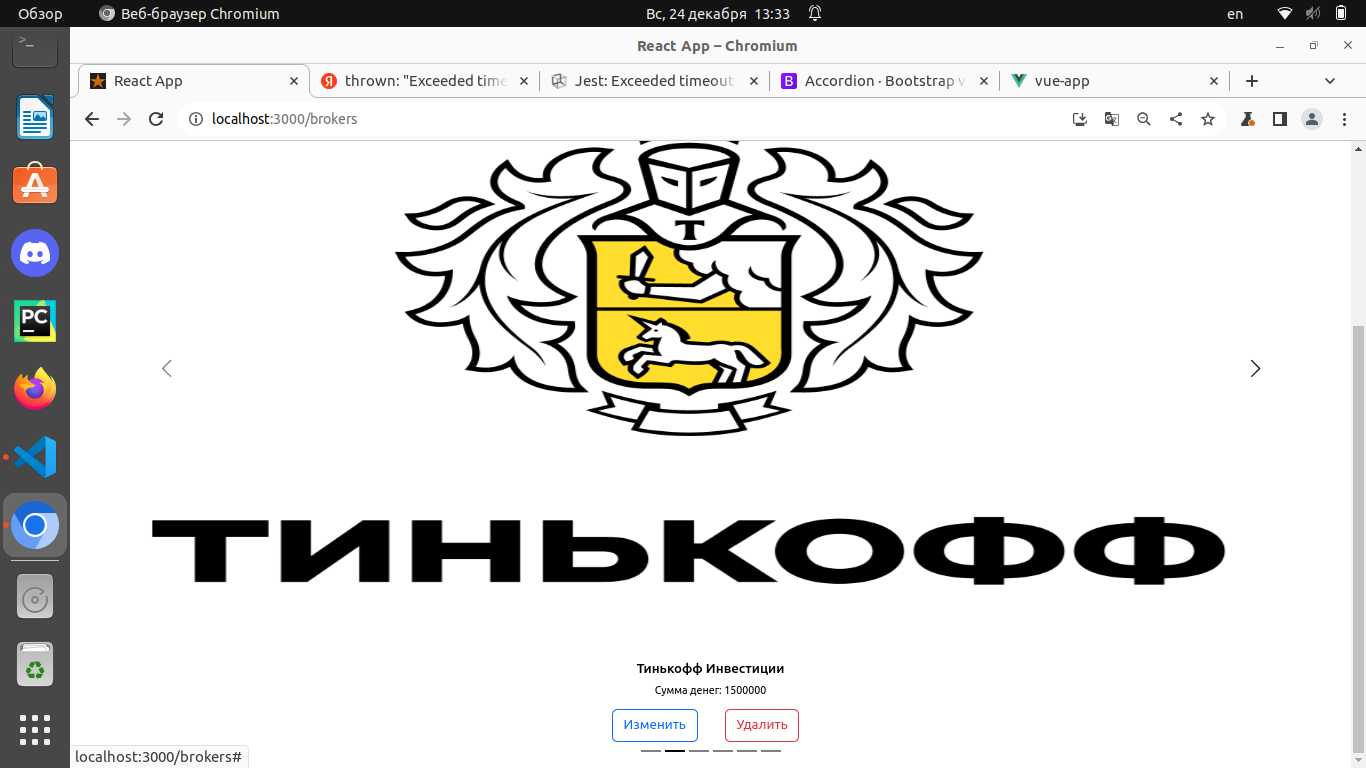


Рис. 3 — Окно добавления брокера

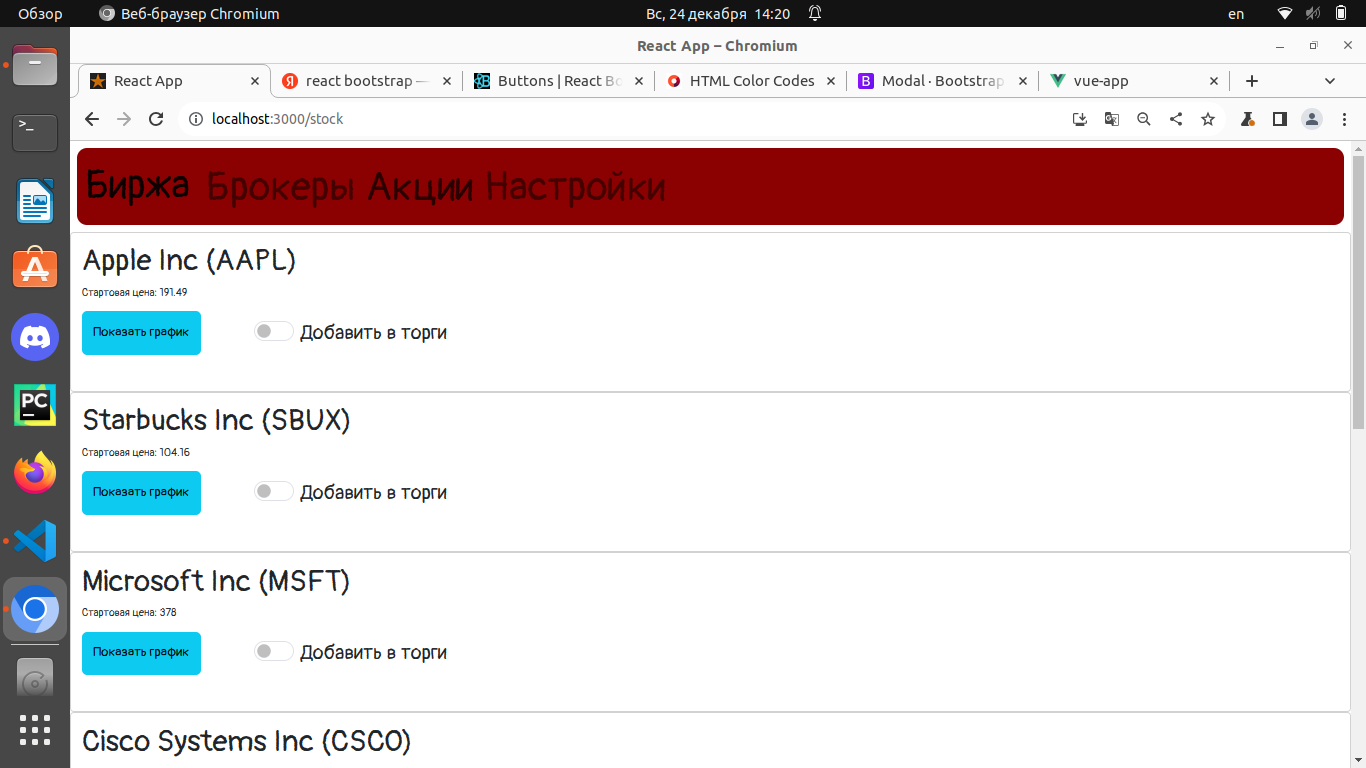
После этого объявляется функциональный компонент Brokers, который выводит информацию о брокерах в виде карусели (Carousel) см. рис. 4-5. Внутри компонента определяются состояния allbroker, update, show и show1 с помощью хука useState. Allbroker будет хранить данные о всех брокерах, update будет сигнализировать о необходимости обновления данных, show будет определять видимость модального окна редактирования брокера, а show1 - модального окна добавления брокера. Затем с помощью хука useEffect происходит загрузка данных о брокерах с сервера. Данные запрашиваются через функцию fetch и устанавливаются в состояние allbroker. Затем состояние update устанавливается в значение false, что означает, что обновление данных завершено. Далее определяется функция delBroker, которая отправляет запрос на удаление брокера на сервер при нажатии на кнопку "Удалить".

Рис. 4 — Слайд добавления брокера

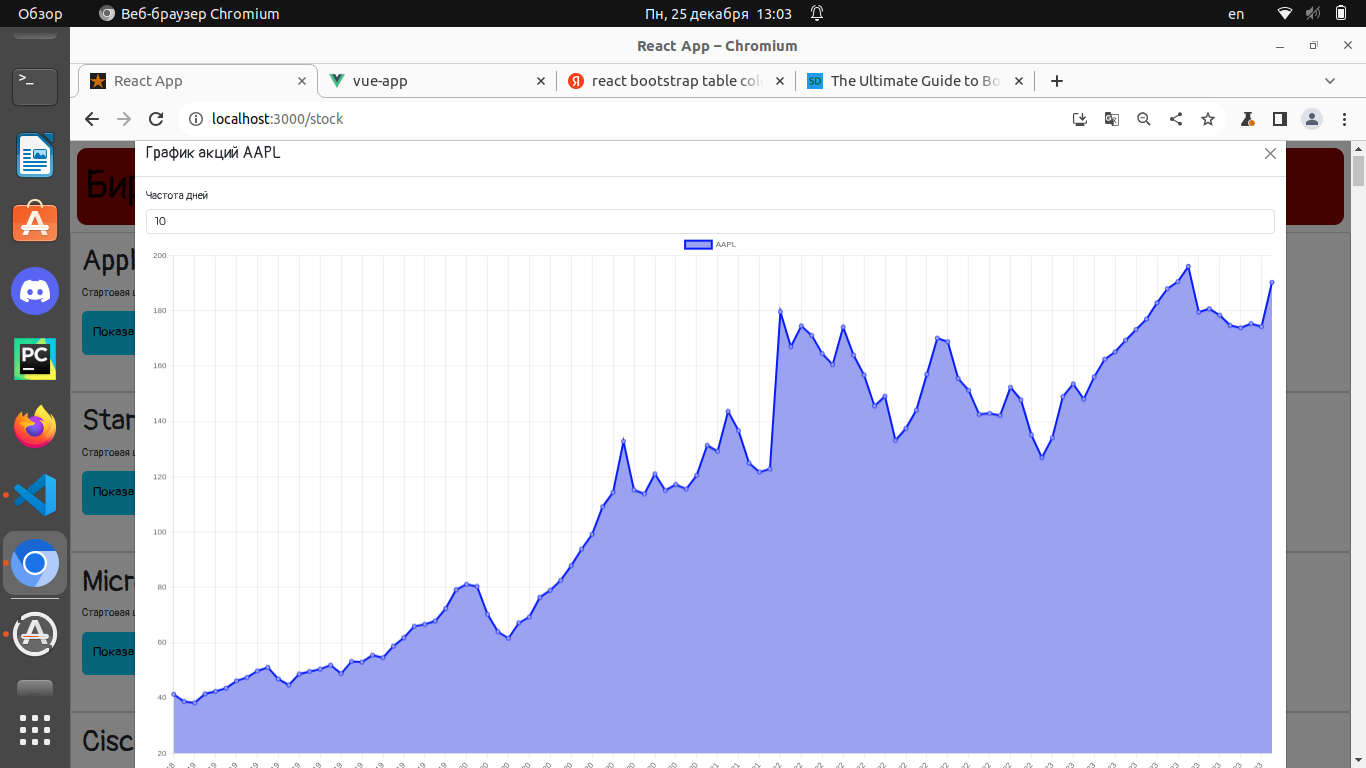
Рис. 5 — Слайд брокера

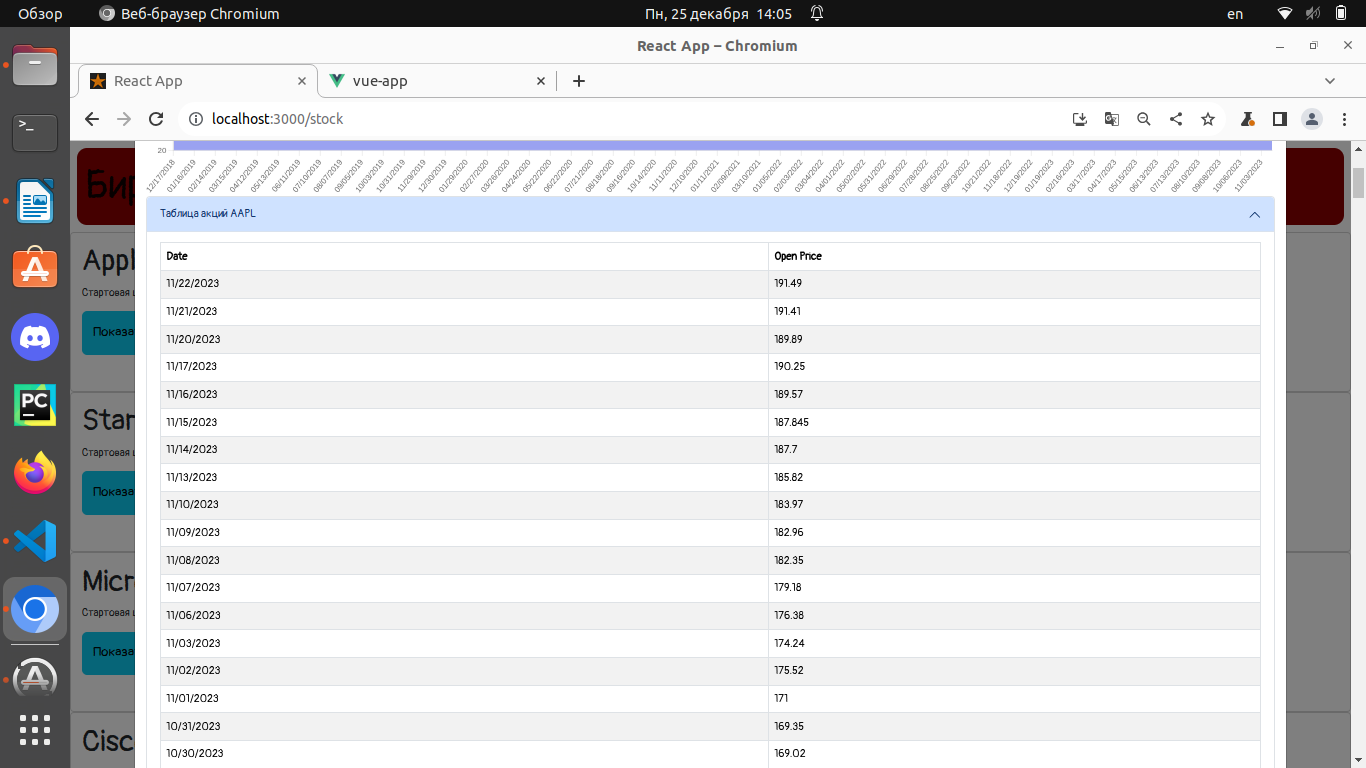
В файле Stock.js определяется функциональный компонент Stock. Внутри компонента Stock определено состояние stocks с помощью хука useState. stocks является массивом, который будет содержать данные об акциях. Затем компонент использует хук useEffect для выполнения асинхронной функции после отрисовки компонента. Внутри этой функции происходит отправка GET-запроса на адрес <http://localhost:4000/stock> и получение данных в формате JSON с помощью метода fetch. Затем полученные данные устанавливаются в состояние stocks с помощью функции setStocks. Хук useEffect также принимает пустой массив зависимостей ([]), что означает, что эта функция будет выполняться только один раз после отрисовки компонента. Далее отображаются карточки акций, содержащие название компании.

В файле StockChart.js определяется функциональный компонент StockChart. Компонент StockChart принимает в качестве свойства stock - код акции. Используется хук useEffect для выполнения асинхронной операции при первом рендеринге компонента. Внутри него выполняется запрос к серверу по адресу '<http://localhost:4000/getstock>' + stock и полученные данные устанавливаются в состояние stockChart. Далее определяются состояния show, которое отвечает за отображение модального окна. В функции updateStore происходит обновление состояния хранилища акций в зависимости от состояния флажка. Если флажок отмечен (event.target.checked=true), то акция добавляется в хранилище (storeStocks.dispatch(addStock(stock))), иначе (event.target.checked=false) - акция удаляется из хранилища (storeStocks.dispatch(delStock(stock))). Функция setValue проверяет наличие акции в хранилище и возвращает true, если акция присутствует, иначе — false. Если доступны данные о выбранной акции (если массив stockChart не пуст), то отображается стартовая цена акции (stockChart[0].Open), кнопка "Показать график" и флажок "Добавить в торги". При нажатии на кнопку открывается модальное окно, в котором отображается график акций с помощью компонента ChartLine. См. Рис. 6.

Рис. 6 — Страница акций.

Файл Chart.js отвечает за отображение графика и таблицу информации торгов (см. рис. 7-8).

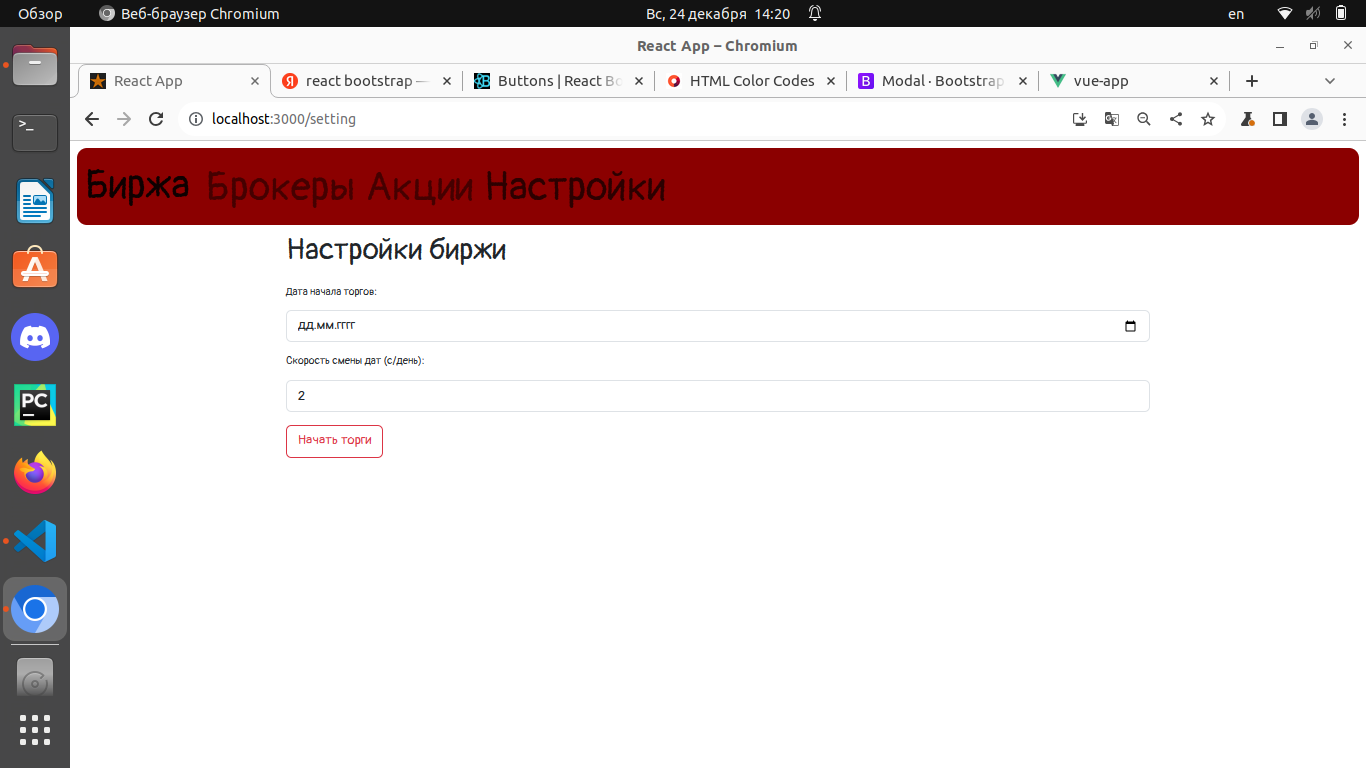
Рис. 7 — График стоимости акции

Рис. 8 — Пример фрагмента таблицы стоимости акции по датам

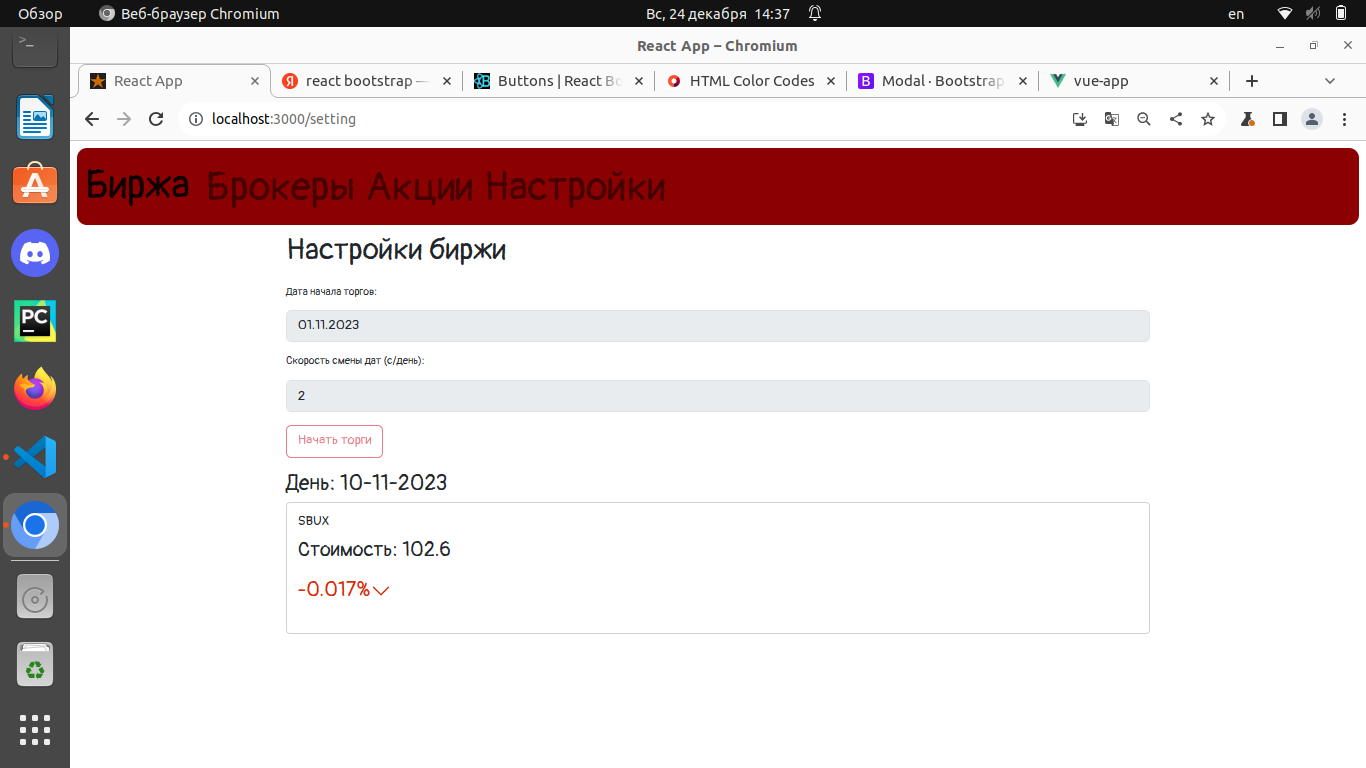
В файле Setting.js определяется функциональный компонент Settings. Внутри компонента "Settings" есть два состояния, определенные с помощью хука useState: tradeStatus: хранит текущий статус торгов (включены или выключены); day: хранит текущий день торгов. Дальше определяется функция "onFormSubmit", которая будет вызвана при отправке формы. Внутри этой функции:

* предотвращается отправка формы по умолчанию.
* создается объект formData, содержащий данные формы.
* преобразуется объект formData в простой объект formDataObj.
* добавляется массив акций из хранилища в объект formDataObj.
* выполняется асинхронный запрос по адресу '<http://localhost:4000/starttrade>' с методом POST.
* после выполнения запроса устанавливается tradeStatus в значение true, чтобы показать, что торги запущены.

См. рис. 9.

Рис. 9 — страница настроек до начала торгов.

В файле Trade.js описан функциональный компонент Trade. Внутри функции Trade используется хук useState для создания состояния stocksCost, которое представляет список со значениями стоимости акций. При помощи хука useEffect происходит обновление стейта при изменении stocksCost. С помощью функций setCost и props.setStatus обновляются значения stocksCost и статус. Затем используется компонент App1, который принимает массив событий ['tradeStarted', 'tradeStatus'] и массив функций [setCost, props.setStatus]. Этот компонент отвечает за получение данных связанных с торговлей. См. Рис. 10.

Рис. 10 — Страница при старте торгов.

2. Для описания NestJs были прописаны файлы app.controller.ts, app.module.ts, edit.ts, gateway.ts, main.ts и app.service.ts.

В файле edit.ts прописаны все основные функции для работы сайта (функции удаление, редактирования и добавления брокеров).

В файле app.controller.ts прописаны основные маршруты.

В файле gateway.ts класс WebGateway является шлюзом для веб-сокетов в NestJS. Он экспортируется и используется для обработки событий в режиме реального времени.

В файле app.service.ts экспортируется класс AppService, который используется для управления и обработки данных. Класс содержит следующие методы и свойства:

* Speed: тип данных number, используется для хранения значения скорости.
* stocks: тип данных массив, используется для хранения списка акций.
* prev\_cost: тип данных массив, используется для хранения предыдущей стоимости акций.
* onedate: тип данных Date, используется для хранения текущей даты.
* twodate: тип данных Date, используется для хранения конечной даты.
* конструктор: принимает в качестве аргумента экземпляр класса WebGateway.
* emitStock(stocks: Update[]): метод, используется для отправки обновленной информации о акциях на сервер.
* emitStatus(status: boolean): метод, используется для отправки информации о статусе торговли на сервер.
* init\_Bidding(set: Bidding\_set): метод, используется для инициализации торговли с заданными параметрами.
* simulated(): метод, используется для симуляции торговли.

## Вывод.

В ходе лабораторной работы были изучены возможности применения библиотеки React (https://reactjs.org/) для разработки интерфейсов пользователя webприложений и использование фреймворка NestJS (https://nestjs.com/) для разработки серверных приложений.