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

Студент гр. 2382 _____ Муравин Е. Е.

Преподаватель _____ Беляев С. А.

Санкт-Петербург

2024

Цель работы

Целью работы является изучение возможностей применения библиотеки React (https://reactjs.org/) для разработки интерфейсов пользователя web приложений и использование фреймворка NestJS (https://nestjs.com/) для раз работки 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 Re act;

– 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.

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 де нежных 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), Am azon.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, остав лены 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 использовани ем компонентов React. Маршрутизация реализована с использованием «react router-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 (https://mui.com/ru/).

Основные теоретические сведения

React – библиотека на JavaScript для построения интерфейса пользовате ля. React представляется удобным инструментом для создания масштабируе мых web-приложений (в данном случае речь идет о клиентской части), осо бенно в тех ситуациях, когда приложение является одностраничным.

В основу React заложены принципы Redux, предлагающее предсказуе мый контейнер хранения состояния web-приложения.

Вся структура веб-страницы может быть представлена с помощью DOM. Для решения проблемы производительности предложена концепция вирту ального DOM, который представляет собой облегченную версию DOM. React работает именно с виртуальным DOM. Реализован механизм, который пери одически сравнивает виртуальный DOM с реальным и вычисляет минималь ный набор манипуляций для приведения реального DOM к состоянию, кото рое 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.

Выполнение работы

Реализуем шаблон проекта на сайте Figma.

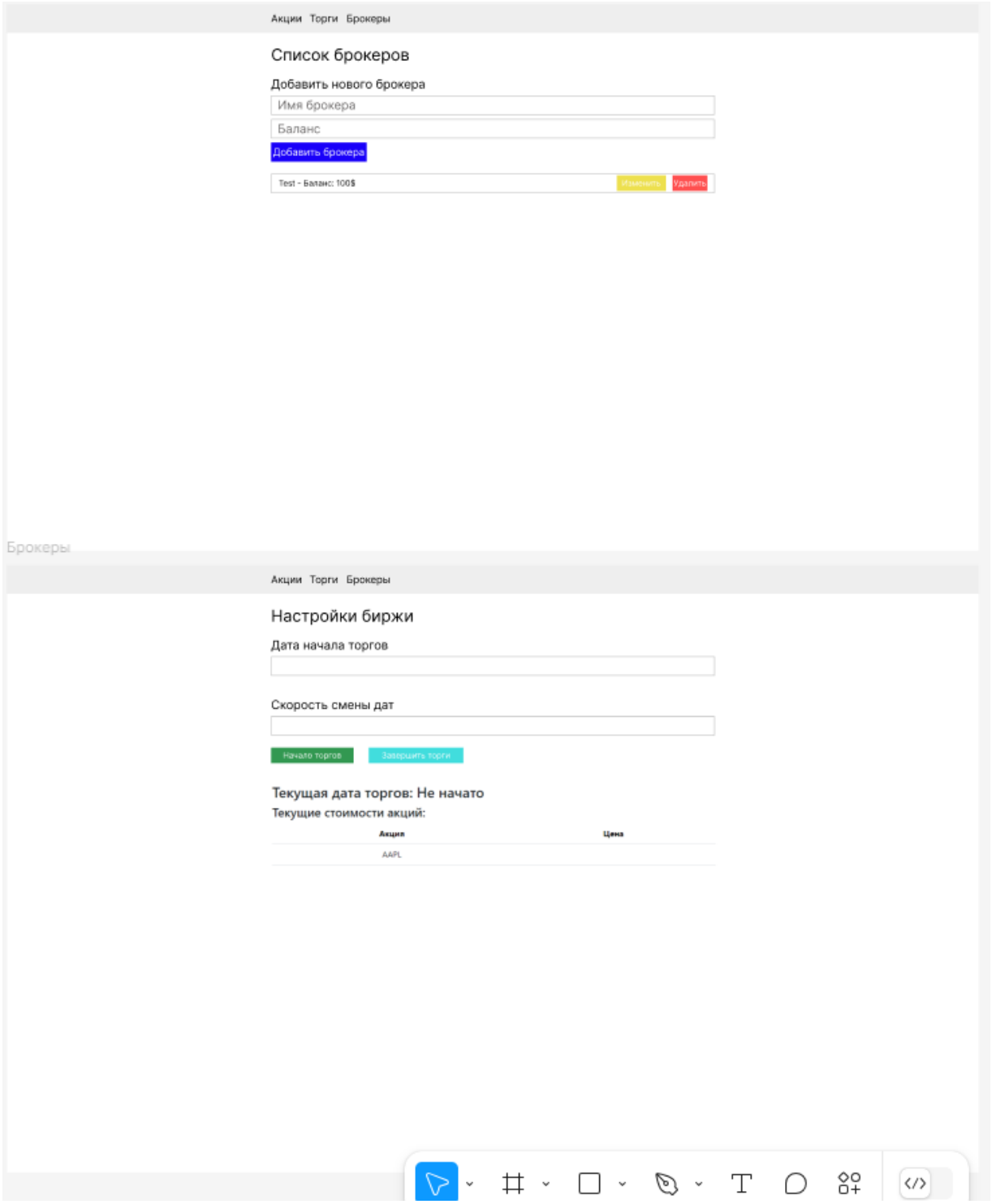

Рисунок 1 – шаблон проекта 1

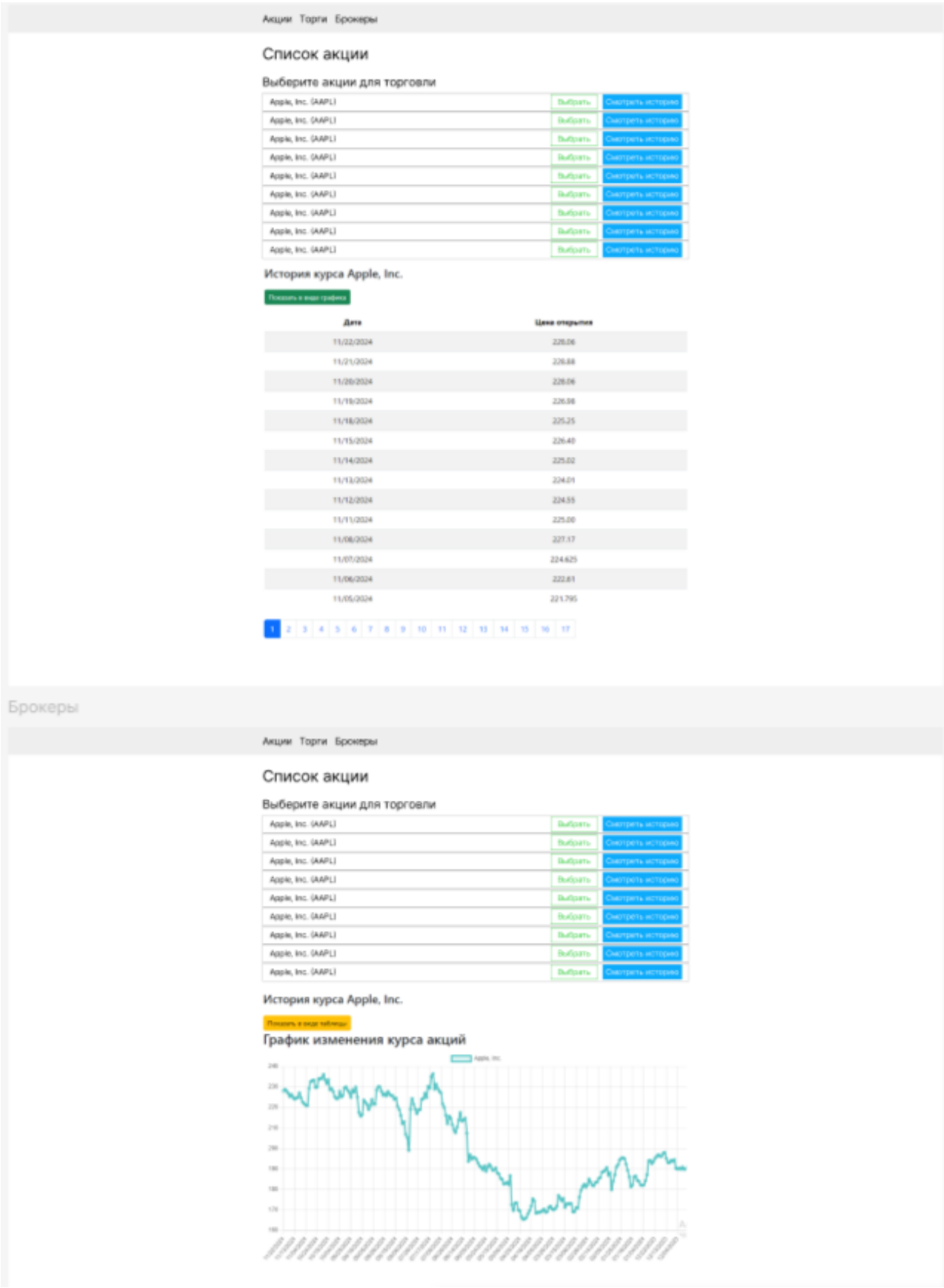

Рисунок 2 – шаблон проекта 2

Серверная часть программы разработана на языке программирования TypeScript с использованием фреймворка NestJS. Данные о брокерах и истории торгов акций хранятся в JSON-файлах. Сервер запускается на порту 4000.

В приложении реализованы контроллер и сервис под названием list-broker. В файле контроллера описаны методы, которые обернуты в декораторы, указывающие на тип HTTP-запроса. Для получения всех брокеров используется метод getAllBrokers, обёрнутый в декоратор @Get. Аналогично, метод getBrokerById позволяет получить данные конкретного брокера. Для добавления новых брокеров и их удаления реализованы методы, использующие HTTP-запросы типа POST, а для изменения данных брокера применяется HTTP-запрос типа PUT.

Все функции, необходимые для работы методов контроллера, реализованы в сервисе list-broker. Сервис отвечает за обработку данных, включая их получение, добавление, обновление и удаление, предоставляя соответствующую логику для взаимодействия с JSON-файлами.

В дополнение к функционалу работы с брокерами реализованы контроллер и сервис для управления акциями. В данном модуле предусмотрены два конечных точки (endpoint) — для получения информации о всех акциях и для получения данных о конкретной акции.

Данные об акциях хранятся в JSON-файле в виде массива объектов. Каждый объект содержит следующие поля: id, label, name и history. Поле id используется для уникальной идентификации акции, label содержит её краткое обозначение, а name — полное название компании. Поле history представляет собой массив, включающий значения цены акций и соответствующие даты, что позволяет отслеживать исторические изменения стоимости.

Сервис, связанный с управлением акциями, реализует всю необходимую логику для работы методов контроллера, включая обработку запросов и взаимодействие с данными в JSON-файле.

Для обмена данными между сервером и клиентом о ходе торгов используется механизм web-сокетов. Реализация обеспечения клиента актуальной информацией о процессе торгов выполнена в файле bidding.gateway.ts.

Торги запускаются через канал startTrading. На этот канал передаются параметры: дата начала торгов, список акций, участвующих в торгах, и скорость моделирования. После получения этих данных активируется метод beginTrading, который создаёт интервал для выполнения инструкций через заданные временные промежутки.

Внутри интервала происходит обработка данных из JSON-файла, содержащего информацию об акциях. Для текущей даты определяется соответствующая цена каждой акции. Затем формируется пакет данных, включающий обновлённую дату и актуальные цены акций, который отправляется клиенту.

Процесс торгов завершается либо вручную, по нажатию кнопки "Завершить торги", либо автоматически, при достижении конечной даты, заданной в настройках.

Клиентская часть приложения разработана на языке JavaScript с использованием библиотеки React. Для управления состоянием на клиенте применяется система Redux, которая используется для хранения данных о брокерах и акциях. Основная логика работы с этими данными реализована в файлах brokerSlice и stockSlice. В них предусмотрены методы для добавления, удаления и изменения информации.

Для взаимодействия с сервером используется библиотека Axios. Соответствующие запросы к серверу описаны в файлах brokerService и

stockService. Эти файлы содержат функции для выполнения HTTP-запросов, таких как получение, добавление, обновление и удаление данных.

Большая часть функционала кнопок на клиентской стороне построена на вызовах серверных методов через Axios. После успешного выполнения серверной операции данные синхронизируются с хранилищем Redux, что обеспечивает обновление пользовательского интерфейса и согласованность данных между клиентом и сервером.

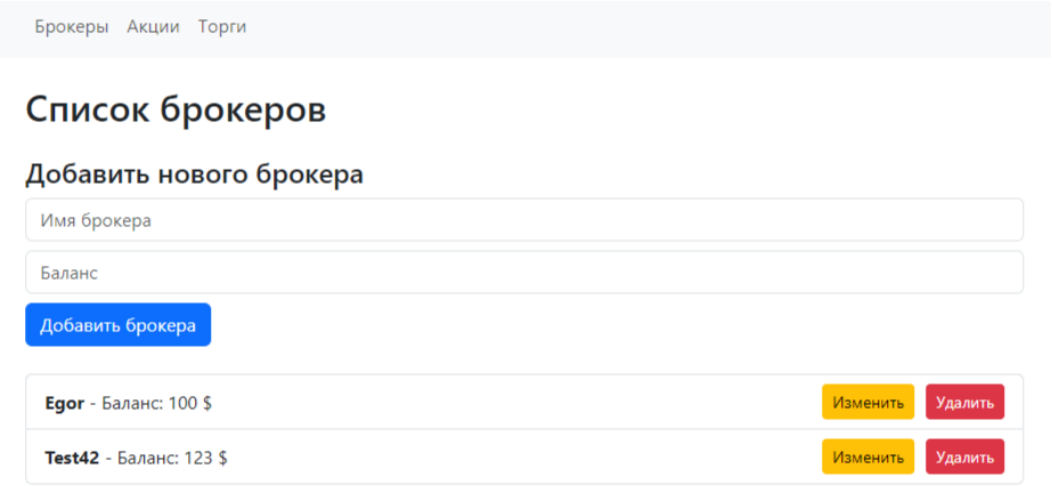

Рисунок 3 – Страница брокеров

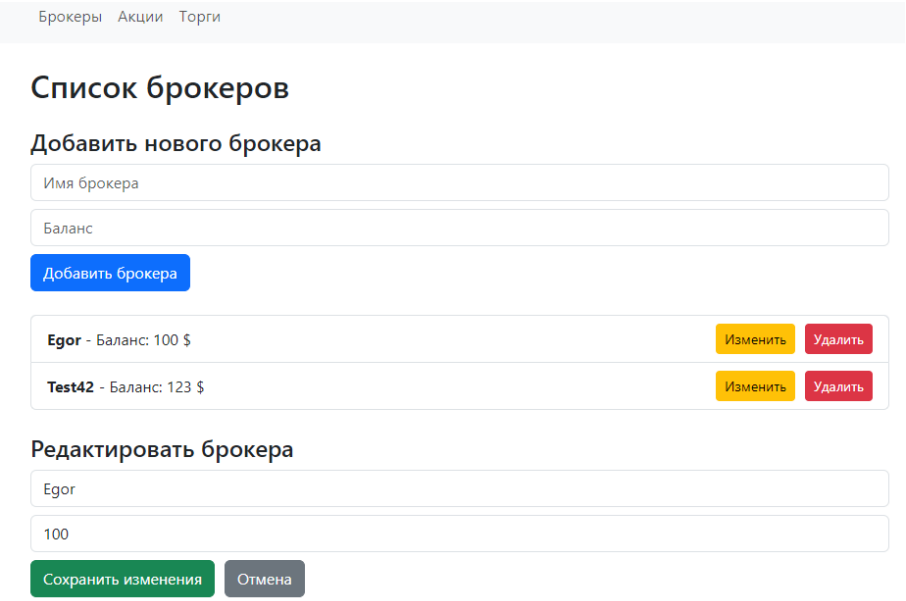

Рисунок 4 – Страница брокеров, изменение данных

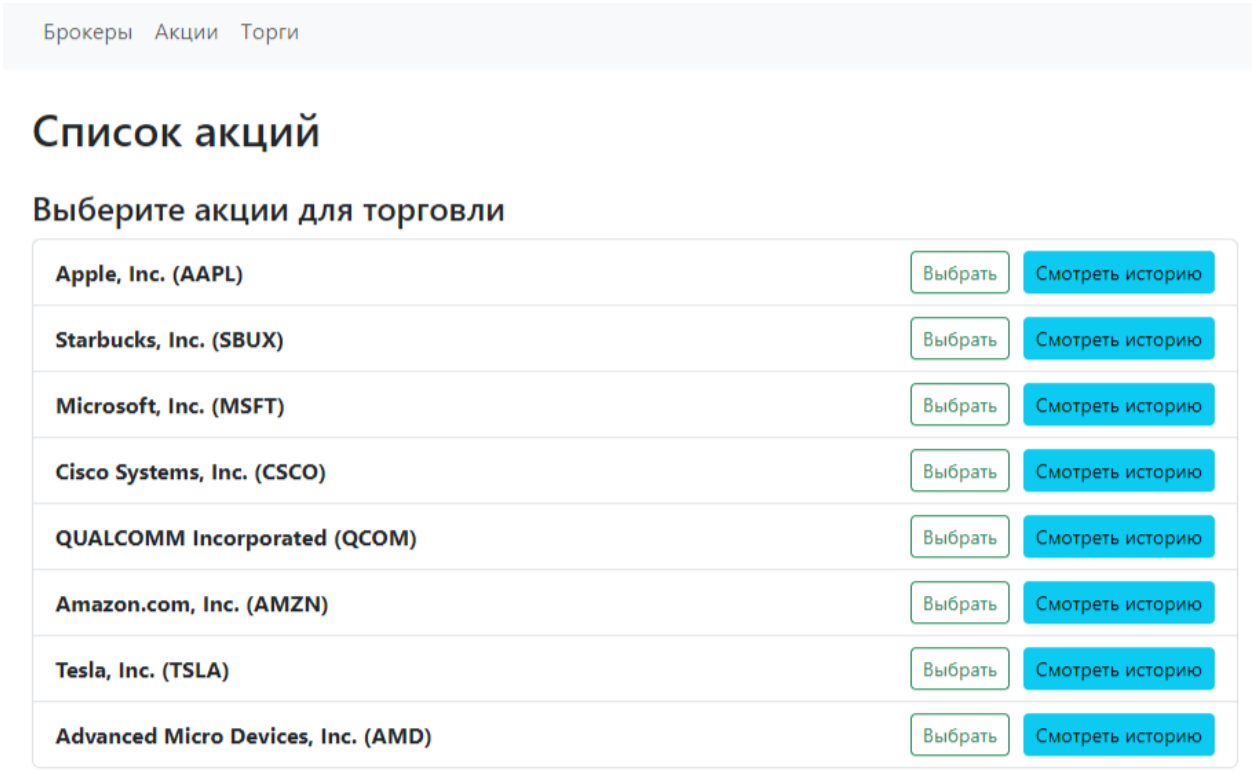

Рисунок 5 – Страница акций

Рисунок 6 – График изменения цены

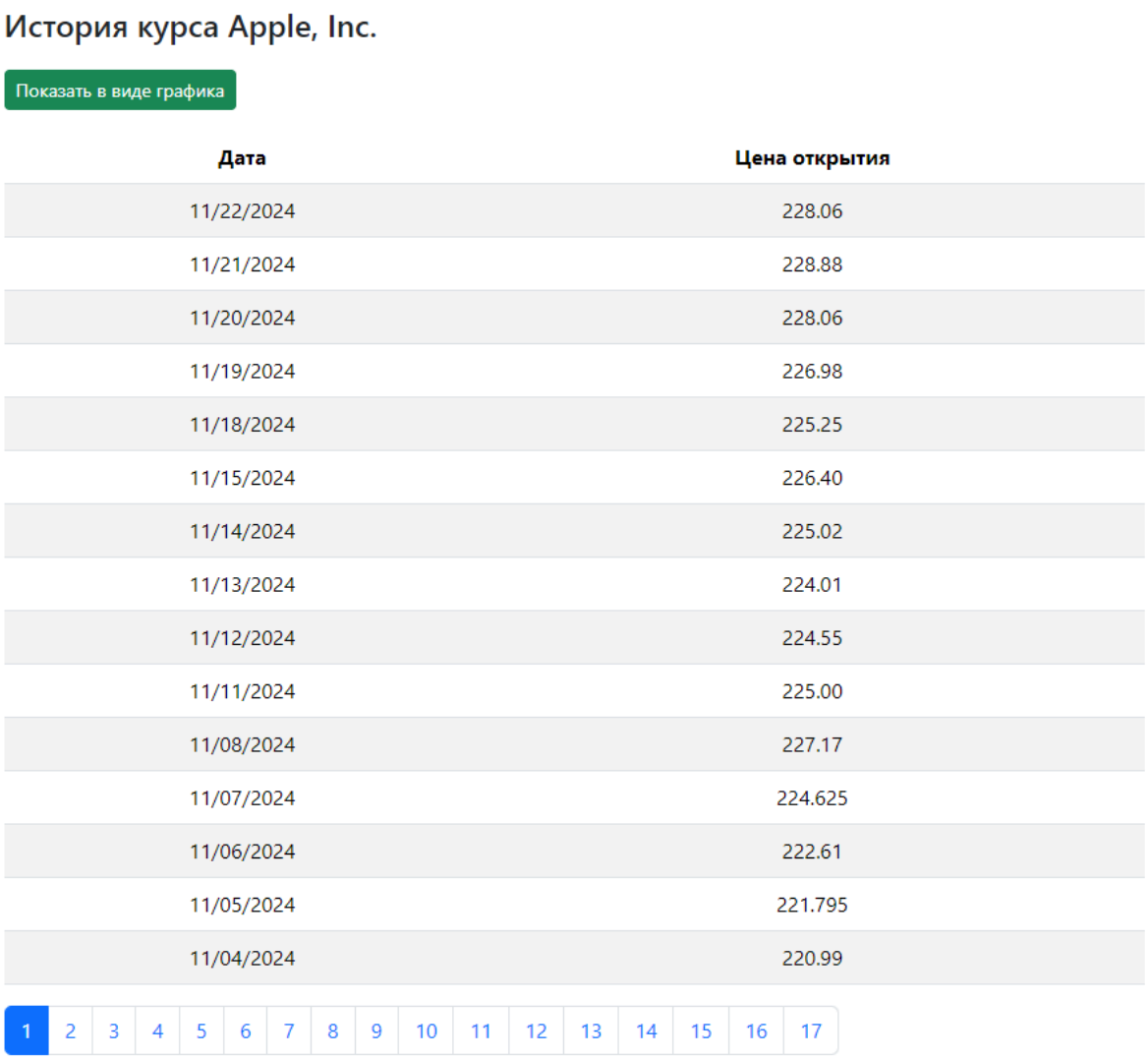

Рисунок 7 – Представление цен в виде таблицы

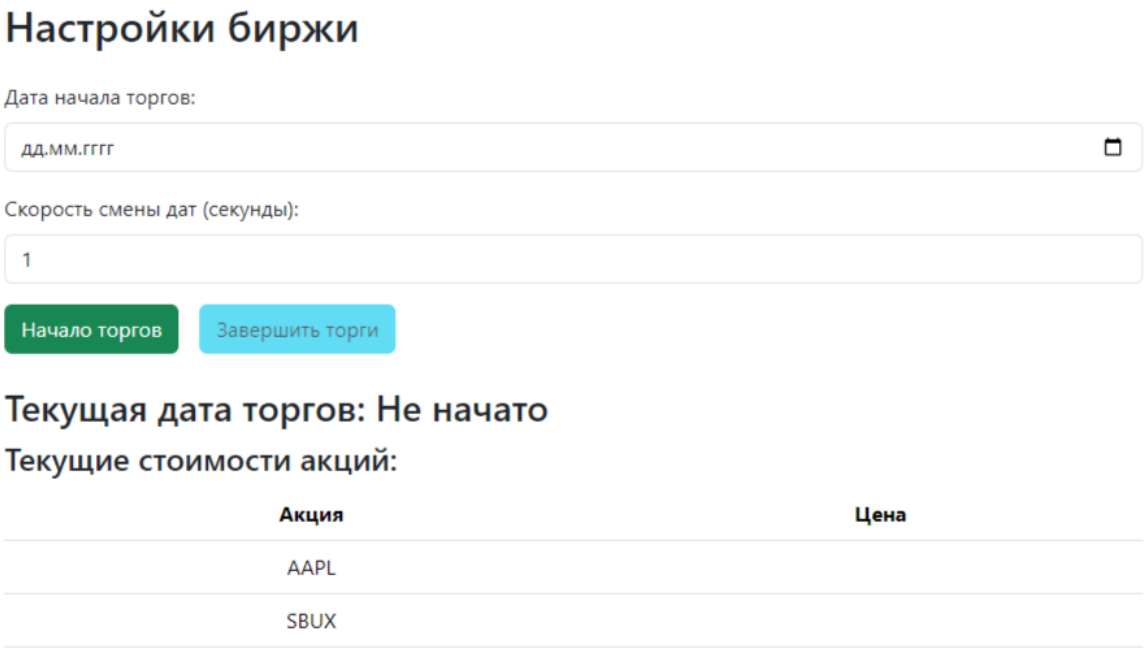

Рисунок 8 – Страница торгов

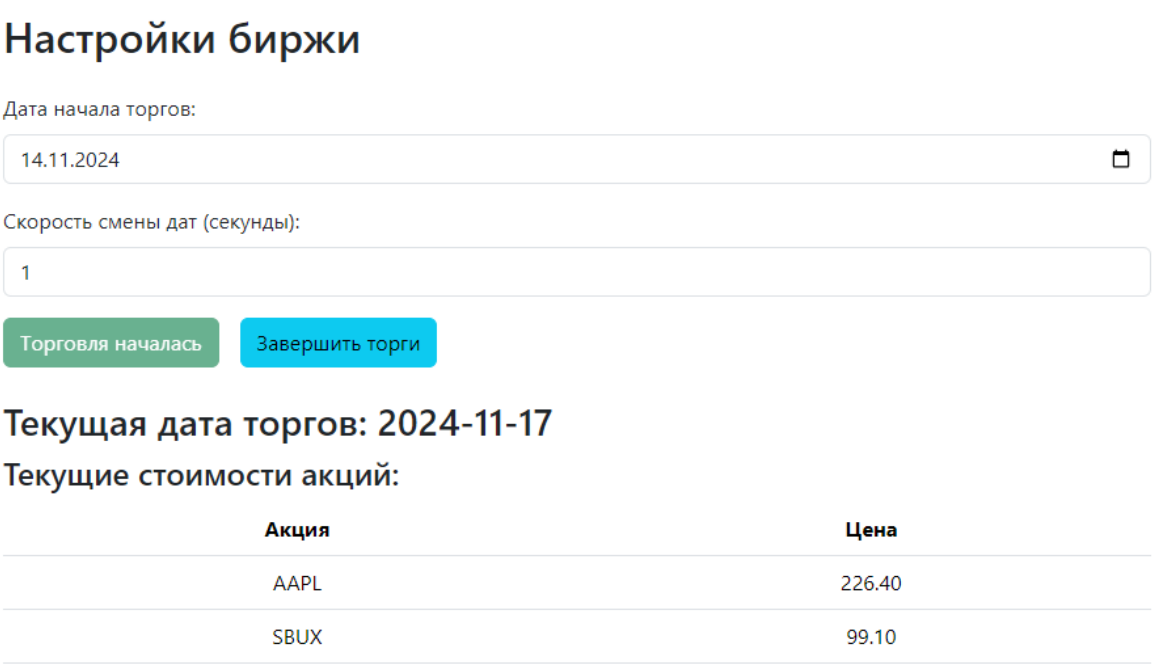

Рисунок 9 – Процесс торгов

Выводы

В ходе работы было изучено создание серверных приложений с использованием фреймворка NestJS и языка TypeScript, а также принципы построения клиентских приложений с использованием React. Исследовались возможности работы с состоянием через Redux и взаимодействия с сервером с помощью библиотеки Axios. Реализованы основные компоненты системы, включая модули для управления брокерами и акциями, динамическое обновление данных торгов через web-сокеты, и интерфейс с адаптивным дизайном.