**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. 1303 _____ Коренев Д.А.

Преподаватель _____ Беляев С.А.

Санкт-Петербург

2023

**Цель работы.**

Изучение возможностей применения библиотеки React (https://reactjs.org/) для разработки интерфейсов пользователя web-приложений и использование фреймворка NestJS (https://nestjs.com/) для разработки серверных приложений.

**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:

1. Информация о брокерах (участниках) и параметрах акций сохраняется в файле в формате JSON
2. В качестве сервера используется NestJS с использованием языка Type-Script
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 годы. 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 (переведен 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 процессы имитации торгов и предоставления информации об изменении курсов акций всем брокерам по web-сокетам с учетом 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 «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/)

**Выполнение работы.**

**Логика сервера (NestJS):**

Был создан сервер с использованием фреймворка *NestJS*. С помощью команды *npx nest g resource* были созданы следующие ресурсы:

Для брокеров:

Написан класс *BrokerGateway*, который является *WebSocket*-шлюзом, который обрабатывает веб-сокет сообщения для операций с брокерами. Используется сервис *BrokerService* для выполнения бизнес-логики.

- *create(@MessageBody() createBrokerDto: CreateBrokerDto)*: обрабатывает сообщение «*createBroker*» от клиента. Вызывается метод *create* из *BrokerService* и отправляет результат обратно клиенту через событие «*update*»
- *findAll()*: обрабатывает сообщение «*findAll*» от клиента. Вызывается метод *findAll* из *BrokerService* и возвращает результат
- *findOne()*: обрабатывает сообщение «*findOne*» от клиента. Вызывается метод *findOne* из *BrokerService* и возвращает результат
- *update(@MessageBody() updateBrokerDto: UpdateBrokerDto)*: обрабатывает сообщение «*updateBalance*» от клиента. Вызывается метод *update* из *BrokerService* и возвращает результат
- *remove(@MessageBody() id: number)*: обрабатывает сообщение «*deleteBroker*» от клиента. Вызывается метод *remove* из *BrokerService* и отправляет результат обратно клиенту через событие «*remove*»

Написан класс *BrokerService*, который предоставляет сервисы для работы с брокерами. Он читает и записывает данные брокеров в файл *brokers.json*. В классе написаны следующие методы:

- *create(createBrokerDto: CreateBrokerDto)*: создает нового брокера на основе данных из *createBrokerDto*, добавляет его в Map *brokers*, обновляет файл *brokers.json* и возвращает созданного брокера
- *loadBrokers()* парсит JSON файл с брокерами и загружает их в локальную переменную brokers: Map<number, Broker>
- *findAll()*: Возвращает все брокеры из массива *brokers*
- *findOne(number):* Находит брокера по id и возвращает его

- *update(updateBrokerDto: UpdateBrokerDto)*: находит брокера по *id* из *updateBrokerDto*, обновляет его баланс, обновляет файл *brokers.json* и возвращает обновленного брокера. Если брокер не найден, возвращает строку «*Broker not found*»
- *remove(id: number)*: удаляет брокера по *id*, обновляет файл *brokers.json* и возвращает удаленного брокера.

Написан модуль *BrokerModule*, который объявляет *BrokerGateway* и *BrokerService* в качестве своих провайдеров.

Написан класс *Broker*, который представляет сущность брокера, который хранит в себе:

- *id*: уникальный идентификатор брокера
- *login*: логин брокера
- balance: баланс брокера
- *actives*: Мар акций, которые принадлежат брокеру. Каждая акция представлена объектом, содержащим *id, name, quantity* и *price*

Для акций:

Написан класс *StockGateway* является *WebSocket*-шлюзом, который обрабатывает веб-сокет сообщения для операций с акциями. Используется сервис *StockService* для выполнения бизнес-логики

- *findAllSocket()*: обрабатывает сообщение от клиента «findAllSocket». Вызывается метод *findAll* из *StockService* и возвращает все акции
- *findAllStocksByDate(@Body() {date: string})* обрабатывает сообщение от клиента «*findAllStocksByDate*» и возвращает все акции на текущую датуНаписан класс *StockService* предоставляет сервисы для работы с акциями. Он читает данные акций из файлов JSON и предоставляет методы для получения всех акций и выбора акций по индексам.
- *findAll()*: возвращает все акции из массива *stocks*

Написан класс *StocksService*, который предоставляет сервисы для работы с акциями. В классе написаны следующие методы:

- *loadStocks()* парсит файлы с историей торгов акций, и заполняет локальную переменную stocks: Map<number, Stock>
- *findAll()* возвращает массив всех акций
- *findOne(number)* возвращает всю информацию об акции с требуемым id
- *create(createStockDto)* создает новый элемент акции

Написан модуль *StockModule*, который объявляет *StockGateway* и *StockService* в качестве своих провайдеров, а в качестве котроллера *StockGateway*.

Написан класс *Stocks*, который представляет сущность акций, который хранит в себе:

- *id*: уникальный идентификатор акций
- *name*: название компании, которой принадлежат акции
- *prices*: Map цен, ключем является дата, значением цена акции
- *quantity*: количество акций

Имеет метод getStockElement(Date) который возвращает StockElement для требуемой даты.

Написан класс *StockElement*, который представляет сущность акций, который хранит в себе:

- *id*: уникальный идентификатор акций
- *name*: название компании, которой принадлежат акции
- *prices*: цена акции
- *quantity*: количество акций

**Логика сервера (React):**

С помощью фреймворка *React* были созданы следующие страницы:

Файл *brokers.jsx* содержит компонент *Brokers*, который отображает страницу брокеров.

Файл tables.*jsx* содержит компонент *Tables*, который отображает страницу с таблицами акций.

Файл *table.jsx* содержит компонент *Tables*, который отображает таблицу акций.

Страница Торгов:

Файл *Charts.jsx* содержит компонент *Charts*, который отображает страницу графиков.

Файл *Chart.jsx* содержит компонент *Chart*, который отображает график акций.

Страница настроек:

Файл Settings.jsx содержит компонент Settings, который отображает настройки торгов.

**Макет интерфейса:**

При создании макета использовалась Figma.

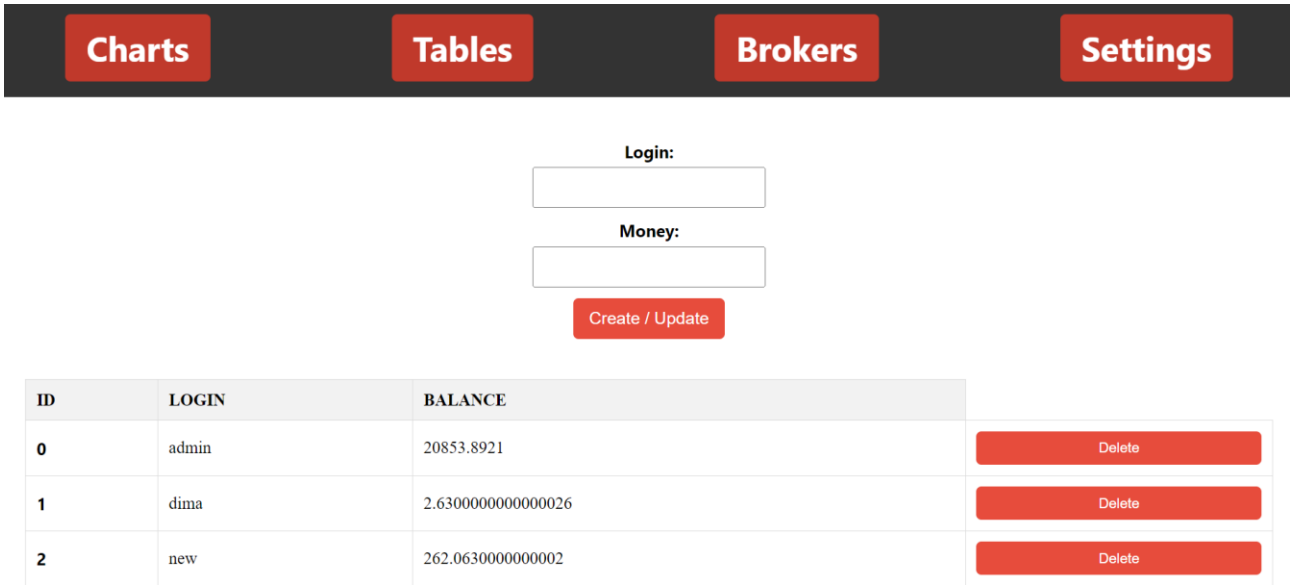



Рисунок 1. Страница брокеров

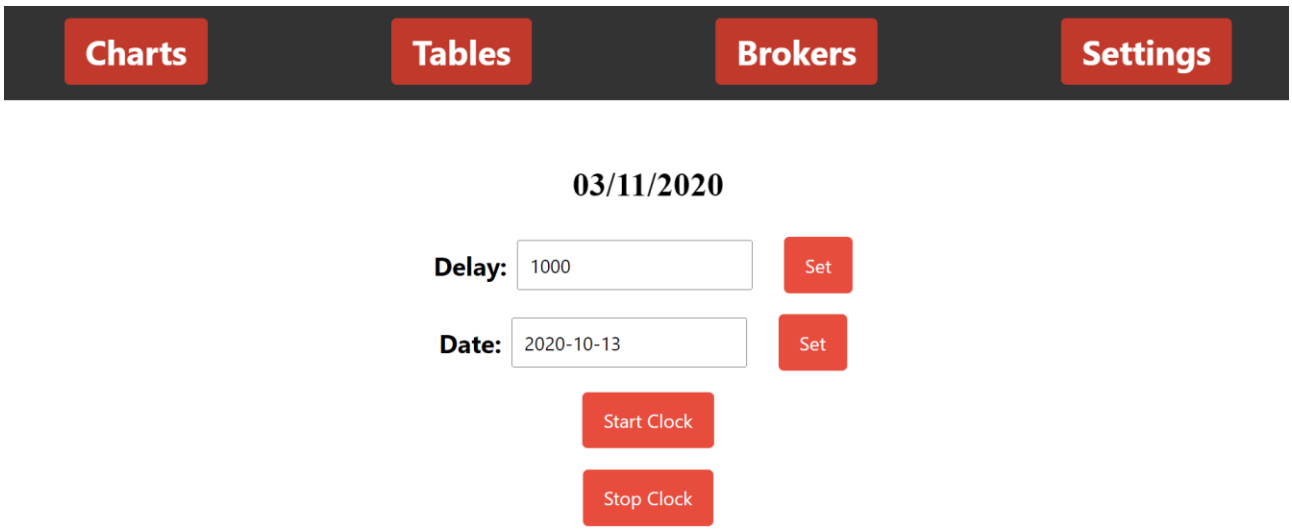
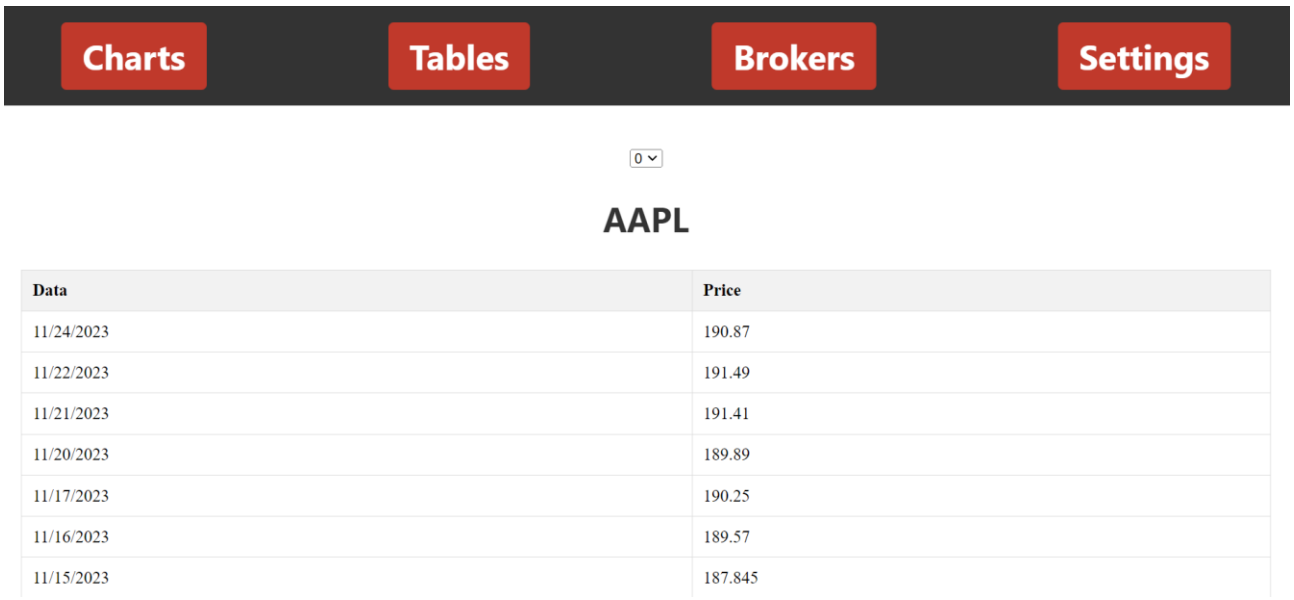



Рисунок 2. Страница настроек




| Data | Price |
|---|---|
| 11/24/2023 | 190.87 |
| 11/22/2023 | 191.49 |
| 11/21/2023 | 191.41 |
| 11/20/2023 | 189.89 |
| 11/17/2023 | 190.25 |
| 11/16/2023 | 189.57 |
| 11/15/2023 | 187.845 |

Рисунок 3. Страница таблиц

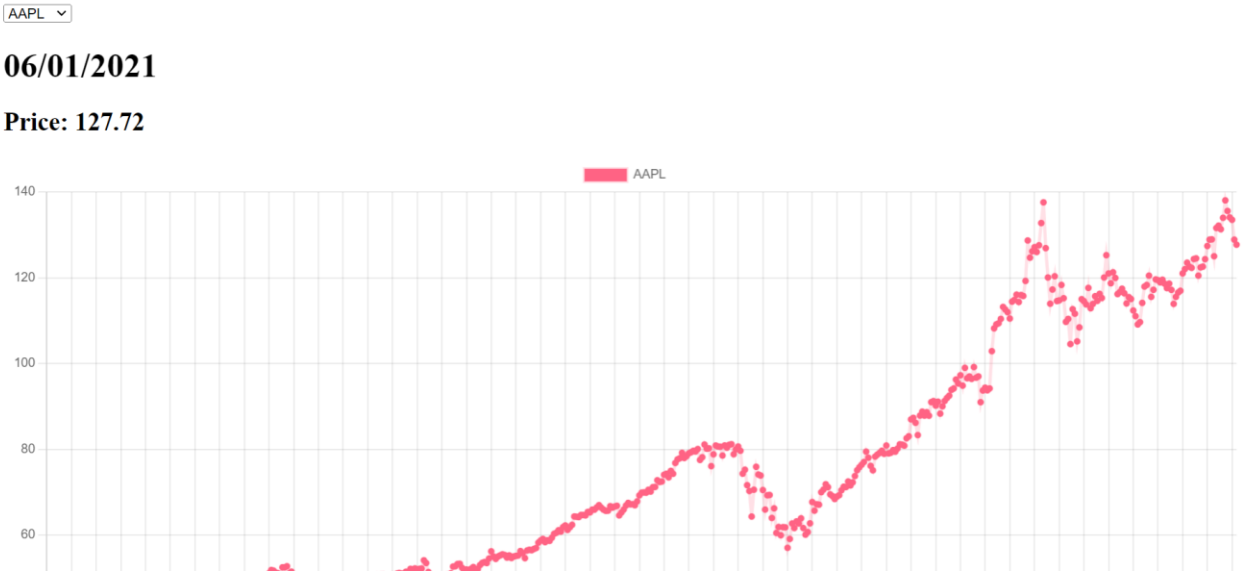



Рисунок 4 Страница графиков

**Макеты страниц.**

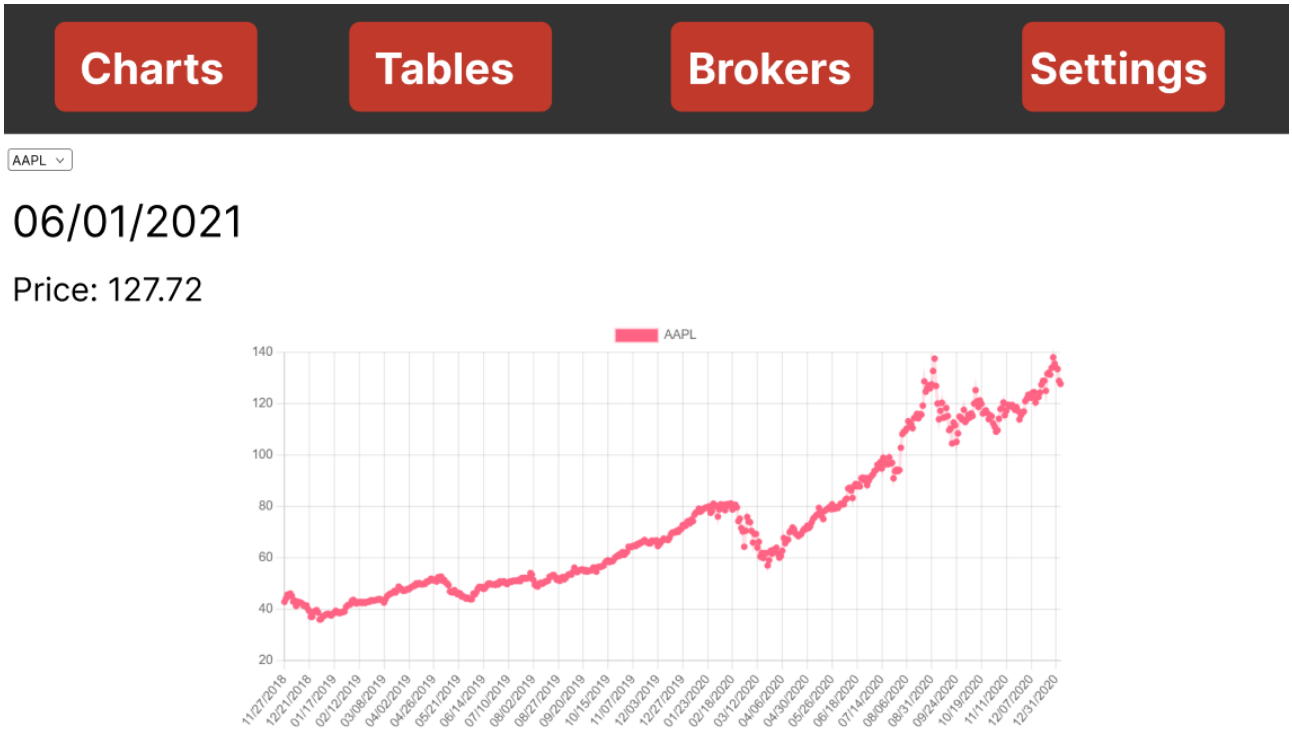



Рисунок 5 макет страницы графиков

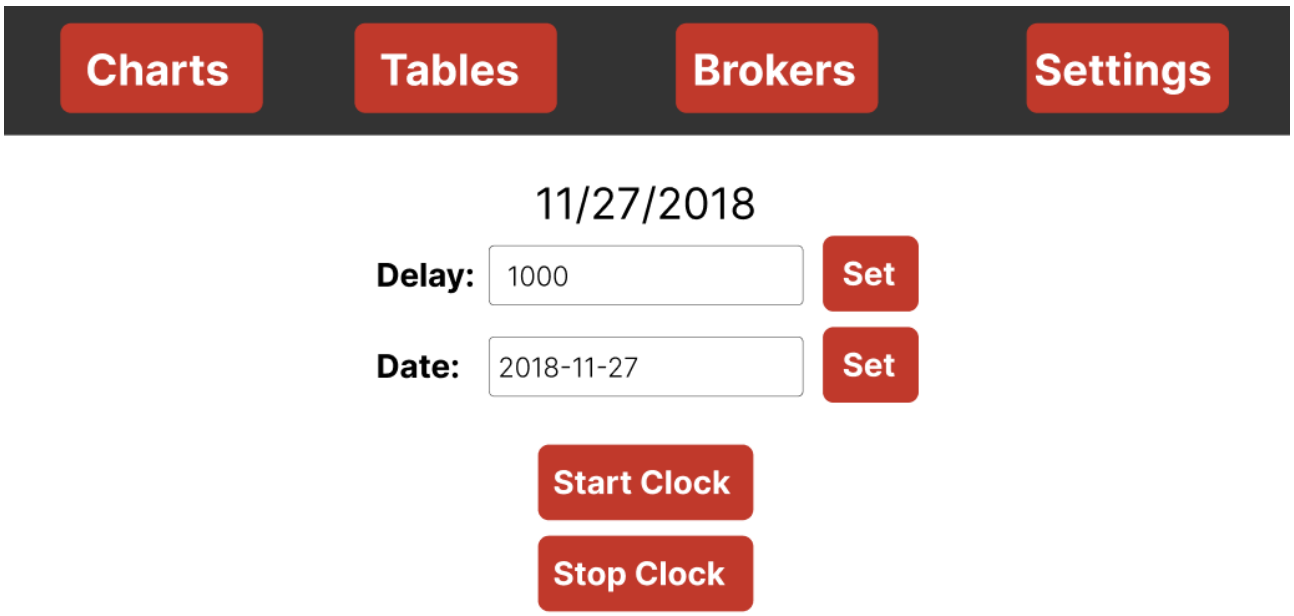



Рисунок 6 Макет страницы настроек

**Вывод.**

Изучение возможностей применения библиотеки React для разработки интерфейсов пользователя в приложении и использован фреймворк NestJS для разработки сервера.