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. 1304 _____ Шаврин А.П.

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 следующие:

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 использование «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.

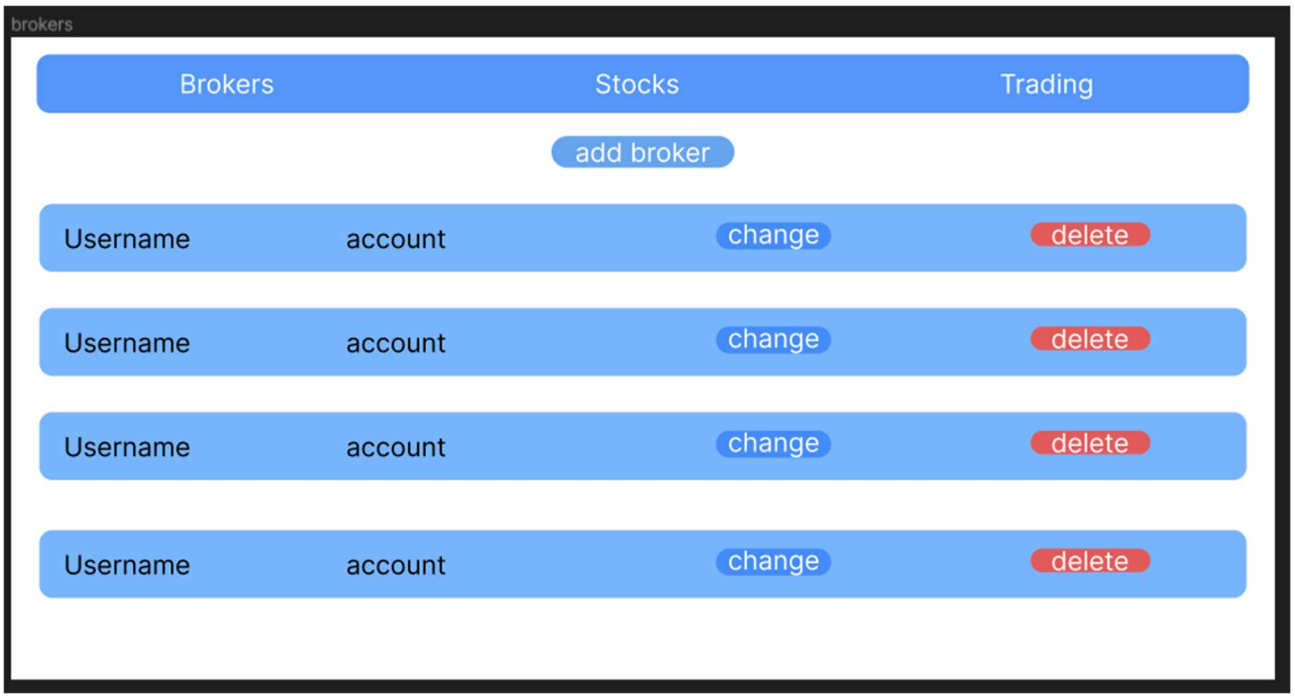
11. Для всех страниц web-приложения разработан макет интерфейса с использованием Figma (https://www.figma.com/).

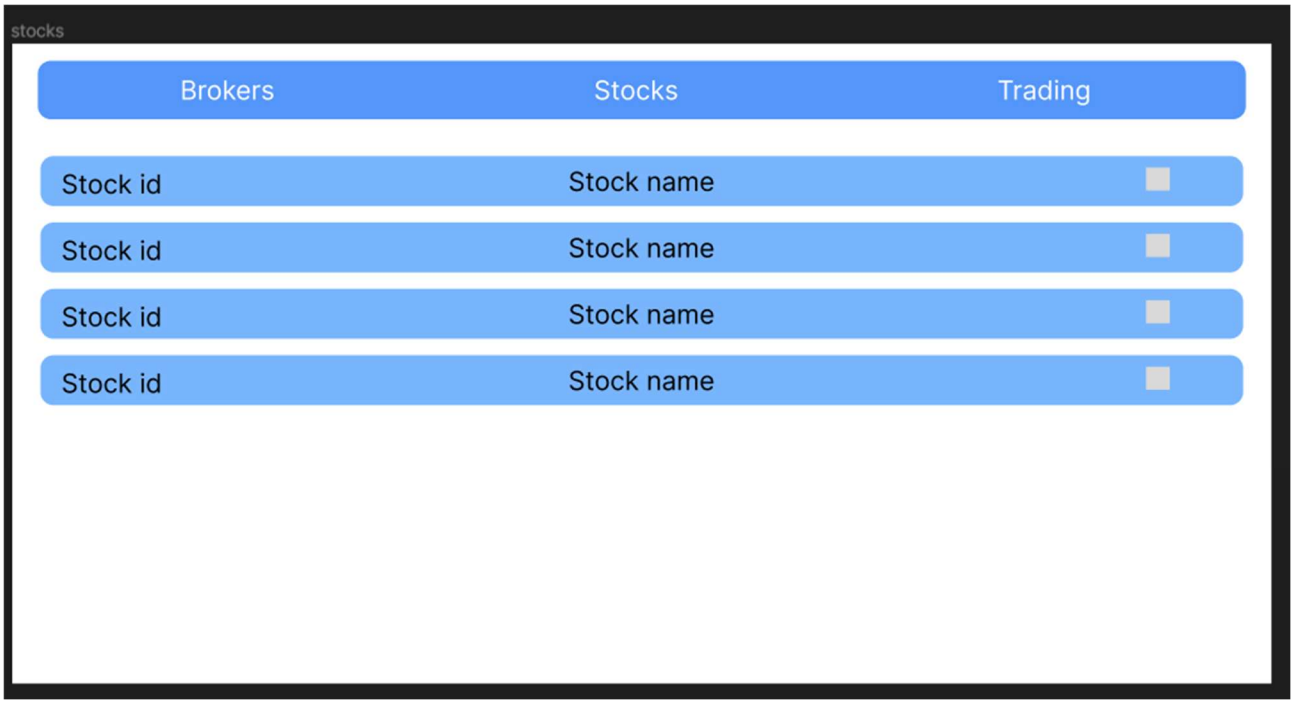
Преимуществом будет создание и использование аутентификации на основе passport.js (http://www.passportjs.org/).

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

Выполнение работы.

1. Созданы макеты страниц в Figma:


brokers
Brokers
Stocks
Trading
add broker
Username
account
change
delete
Username
account
change
delete
Username
account
change
delete
Username
account
change
delete


stocks
Brokers
Stocks
Trading
Stock id
Stock name
Stock id
Stock name
Stock id
Stock name
Stock id
Stock name

2. Для серверной части был написан контроллер и сервис-реализующий запросы

```
import {Body, Controller, Delete, Get, Param, Post, Put, Query} from '@nestjs/common';
import { AppService } from './app.service';


@Controller()
export class AppController {
  constructor(private readonly appService: AppService) {}

  @Get("/getAllBrokers")
  getAllBrokers(): string {
    //console.log(this.appService.getAllBrokers())
    return this.appService.getAllBrokers();
  }

  @Get("/getStocks")
  getStocks(): string {
    //console.log(this.appService.getStocks())
    return this.appService.getStocks();
  }

  @Post("/addBroker")
  addBroker(@Body() body: any): string {
    //console.log('ADD ')
    return this.appService.addBroker(body);
  }

  @Put("/changeBroker/:id")
  changeBroker(@Param() { id }: any, @Body() body: any): string {
    return this.appService.changeBroker(id, body);
  }

  @Delete("/deleteBroker/:id")
  deleteBroker(@Param() { id }: any): string {
    return this.appService.deleteBroker(id);
  }
}
```

Контроллер

```
import { Injectable } from '@nestjs/common';
import BROKERS from "./brokers.json";
import STOCKS from "./stocks.json";
import * as fs from 'fs';


@Injectable()
export class AppService {
  getAllBrokers(): string {
    //console.log("brokers", BROKERS)
    return JSON.stringify(BROKERS);
  }

  getStocks(): string {
    return JSON.stringify(STOCKS);
  }

  addBroker(body: any): string {
    let last_index = BROKERS.length - 1;
    let id = BROKERS[last_index].id + 1;
    body.id = id;

    BROKERS.push(body);

    fs.writeFile('src/brokers.json', JSON.stringify(BROKERS), (err) => {
      if (err) throw err;
      console.log('The file has been saved!');
    });

    return JSON.stringify({ "mes": "success" });
  }

  changeBroker(id: any, body: any): string {
    const index = BROKERS.map((broker: any) => {
      return broker.id;
    }).indexOf(Number(id));
```

```
    }).indexOf(Number(id));

    if (index === -1) {
      return JSON.stringify({ "mes": "fail" });
    } else {
      BROKERS[index].account = parseInt(body.account);

      fs.writeFile('./brokers.json', JSON.stringify(BROKERS), (err) => {
        if (err) throw err;
        console.log('The file has been saved!');
      });

      return JSON.stringify({ "mes": "success" });
    }
  }

  deleteBroker(id: any): string {
    const index = BROKERS.map((broker: any) => {
      return broker.id;
    }).indexOf(Number(id));

    if (index === -1) {
      return JSON.stringify({ "mes": "fail" });
    } else {
      BROKERS.splice(index, 1);

      fs.writeFile('src/brokers.json', JSON.stringify(BROKERS), (err) => {
        if (err) throw err;
        console.log('The file has been saved!');
      });

      return JSON.stringify({ "mes": "success" });
    }
  }
}
```

Сервис

3. На сервере реализован сокет сервис

```typescript
@WebSocketGateway(
    {
        cors: {
            origin:"*"
        }
    }
)
export class SocketService implements OnGatewayConnection{
    private index: number = 0;
    private interval: any;

    handleConnection(client: any) {
        console.log("CONNECTED");
    }

    @SubscribeMessage("start")
    handleEvent(@MessageBody() dto: any, @ConnectedSocket() client: any){
        console.log(dto)
        this.index = Number(dto.index);
        const res = {type: "send", dto}
        this.interval = setInterval(()=>{
            console.log(this.index);
            client.emit("trading", this.index);
            this.index += 1;
        }, 1000*dto.speed);
    }

    @SubscribeMessage("stop")
    handleStopEvent(@MessageBody() dto: any, @ConnectedSocket() client: any){
        clearInterval(this.interval)
        console.log("stop")
        this.index = 0
    }
}
```

Сокет сервис

4. На клиенте также реализован сокет сервис

```typescript
import * as socketIo from 'socket.io-client';
import { useEffect } from "react";


const SERVER_ERL = "http://localhost:8080/";


export class SocketIoService{
    static socket  = null
    static connection() {
        this.socket = socketIo.connect(SERVER_ERL);

        this.socket.on('connect', ()=>{
            console.log("connected")
        });

        this.socket.on('disconnect', ()=>{
            console.log("disconnect")
        });
    }
}



export const useConnectSocket = () => {
    const socketConnect = ()=>{
        SocketIoService.connection();
    }

    useEffect(()=>{
       socketConnect()
    },[])
}
```

Клиентски сокет сервис

5. На клиенте реализован роутер маршрутов

```
function Router() {
    return (
        <BrowserRouter>
            <NavigationBar/>
            <Routes>
                <Route exact path="/brokers" element={<Brokers/>}></Route>
                <Route path="/stocks" element={<Stocks/>}></Route>
                <Route path="/trading" element={<Trading/>}></Route>
                <Route path="*" element={<NoMatch/>}></Route>
            </Routes>
        </BrowserRouter>
    );
}
```

Клиентский роутер

6. Использована технология Redux для передачи данных о выбраных акциях для трейдинга

```
const defaultState = { tradingList: [] };

const reducer = (state = defaultState, action) => {
  switch (action.type) {
    case "SAVE":
      return { ...state, tradingList: action.tradingList };
    default:
      return state;
  }
}

const store = createStore(reducer);
```

Redux

7. Итоговый внешний вид приложения:

Страница брокеров

Страница акций

Страница торгов

Выводы.

Изучены возможности применения библиотеки React (https://reactjs.org/) для разработки интерфейсов пользователя web-приложений и использован фреймворк NestJS (https://nestjs.com/) для разработки серверных приложений.