|  |
| --- |
|  |
| МИНОБРНАУКИ РОССИИ |
| Федеральное государственное бюджетное образовательное учреждение  высшего образования  **«МИРЭА − Российский технологический университет»**  **РТУ МИРЭА** |
| Институт кибербезопасности и цифровых технологий |
| Кафедра КБ-14 «Цифровые технологии обработки данных» |

**ОТЧЕТ**

**по практической работе**

**«№6. РАЗРАБОТКА СЕРВЕРНОГО ВЕБ-ПРИЛОЖЕНИЯ»**

**по дисциплине «Обработка данных с веб-ресурсов и поисковых машин»**

|  |  |  |  |
| --- | --- | --- | --- |
| Выполнил |  |  | Парамонов П.К.  *фамилия, имя, отчество* |
| шифр | 22Б0723 | группа | БСБО-05-22 |
|  |  |  |  |
| Проверил |  | к.т.н., доцент  *ученая степень, должность* | Ильин Д.Ю.  *фамилия, имя, отчество* |

**Москва 2024г.**

## **Цель практической работы**

Цель настоящей практической работы — научиться разрабатывать серверные веб-приложения для построения веб-страниц и вывода данных.

## **Задачи практической работы**

Для достижения поставленной цели необходимо решить следующие задачи:

1. Сформировать проект на Node.js на основе результатов предшествующих работ.
2. Выбрать фреймворк для разработки веб-приложения.
3. Разработать маршрутизацию для доступа к веб-страницам приложения.
4. Подключить ORM-модели из практической работы №3.
5. Разработать методы для построения веб-страниц, разработанных в практической работе №5.
6. Реализовать вывод данных с использованием шаблонизации и выбранных средств визуализации.
7. Подготовить ответы на контрольные вопросы.
8. Составить отчет о проведенной работе.

**Ход выполнения практической работы**

В ходе выполнения практической работы для серверной части был использован Express.js, как шаблонизатор использовался EJS, для более удобной работы с динамическими данными. Для построения графиков использовалась Chart.js, Как CSS-фреймворк был выбран Bootstrap.

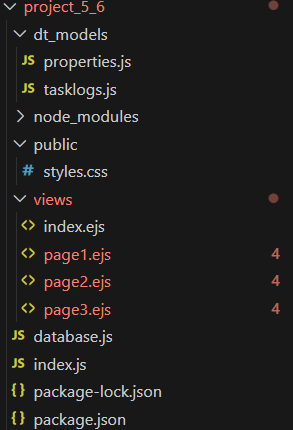


Рис 1 – Структура проекта

Скрипты properties.js, tasklogs.js и database.js были взяты из аналогичных в 3 практической работе, все view были взяты из 5 работы. Полный код index.js будет представлен в приложении А.

Для маршрутизации на сайте, был использован EJS.  


Рис 2 – пример реализации маршрутизации

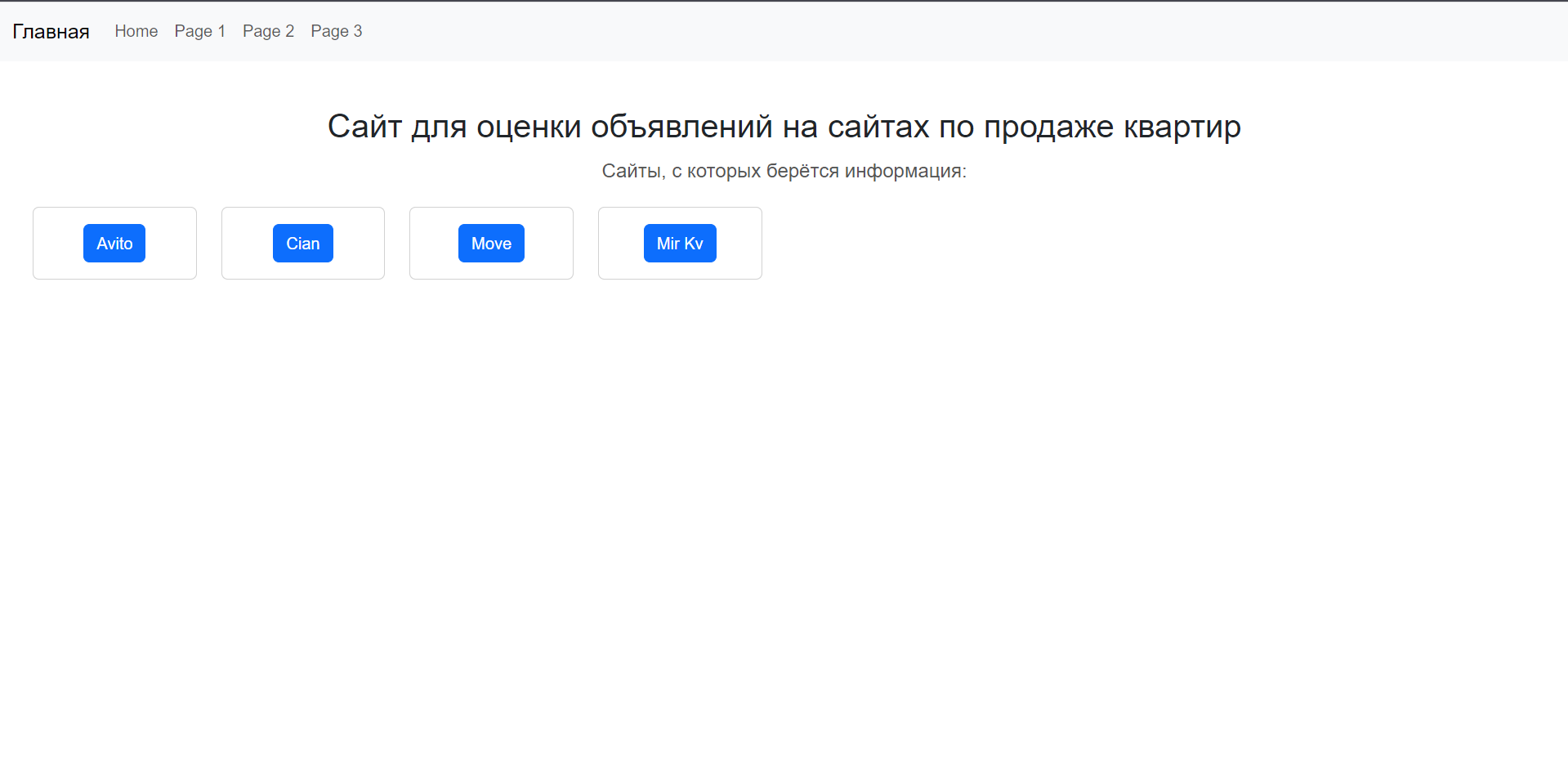


Рис 3 – главная страница

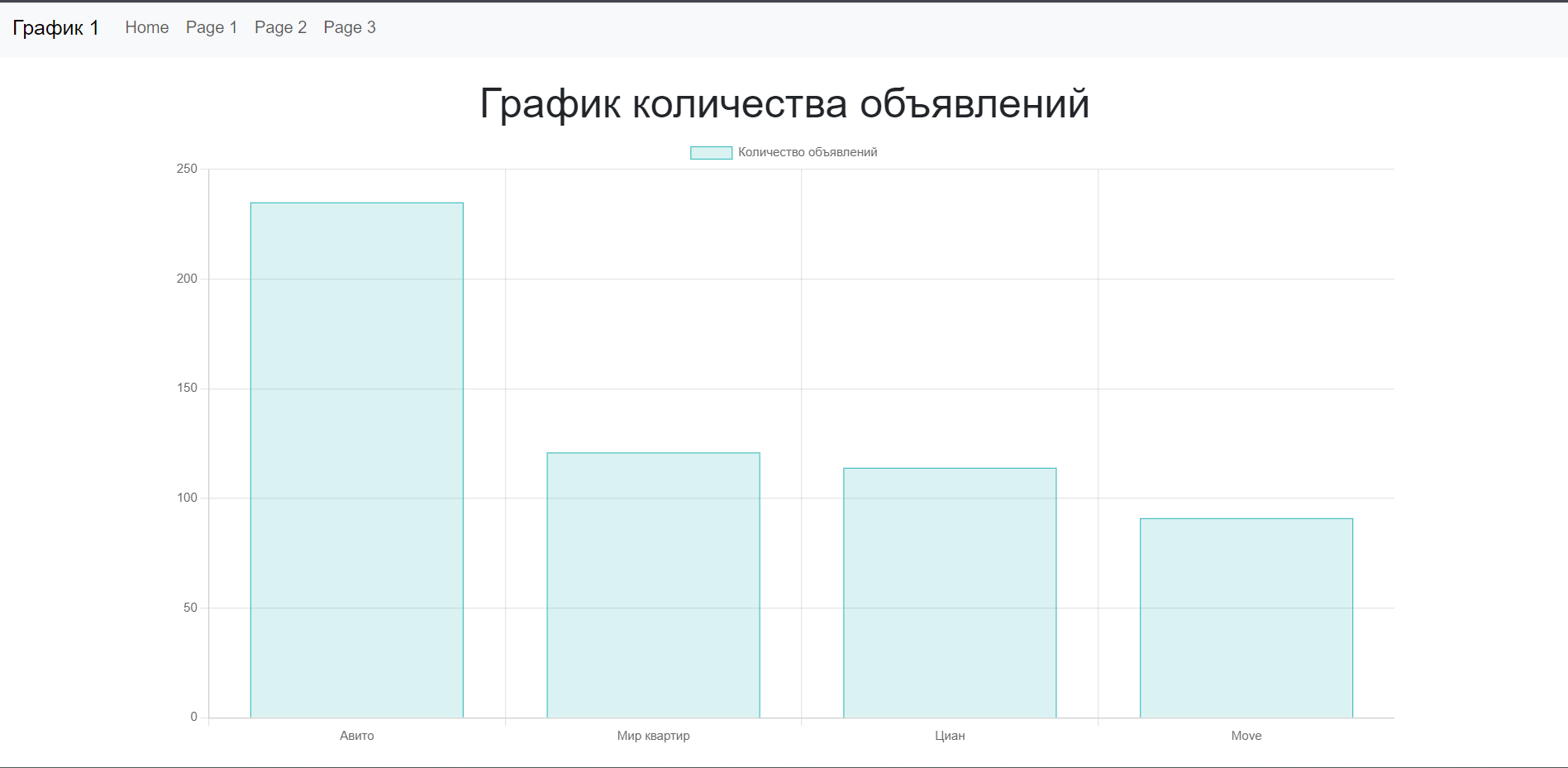


Рис 4 – страница с графиком количества объялений



Рис 5 – страница с графиком распределения квартир по станциям метро



Рис 6 – страница с графиком средней цены квартир по источникам

**Трудности в ходе выполнения практической работы**

Возникли трудности при настройке маршрутизации и загрузки динамических данных при помощи шаблонизатора EJS.

**Ответы на контрольные вопросы**

1. **Отличия Express.js от HTTP в Node.js**:

* Express.js — это фреймворк, который упрощает разработку, предоставляя функции для маршрутизации, middleware, работы с шаблонизаторами и обработкой запросов.
* HTTP — встроенный модуль Node.js для создания серверов, предоставляющий базовые методы, но требует больше кода для реализации аналогичных функций.

1. **Шаблонизаторы в Express.js**:

* Популярные: Pug, EJS, Handlebars.
* Они позволяют генерировать HTML на основе данных, интегрированных в шаблоны.

1. **Преимущества серверных фреймворков**:

* Упрощают создание маршрутов и middleware.
* Поддерживают шаблонизаторы и работу с сессиями.
* Снижают объем кода, повышают читаемость и ускоряют разработку.

1. **Отличия SPA и MPA**:

* SPA (Single Page Application): загружается один HTML-документ, навигация динамическая, требует API для данных.
* MPA (Multi Page Application): каждая страница загружается отдельно, простая SEO-оптимизация, больше запросов на сервер.

1. **Запуск Express.js на нескольких ядрах**:

* Использовать модуль cluster для форка процессов.

const cluster = require('cluster');

const os = require('os');

if (cluster.isMaster) {

os.cpus().forEach(() => cluster.fork());

} else {

require('./app'); // основной файл приложения

}

**Вывод**

В ходе выполнения практической работы «Разработка серверного веб-приложения» было разработано веб-приложение, включающее в себя динамическую загрузку данных через шаблонизатор, вычисление агрегированной информации (средние цены, количество объявлений, распределение по станциям метро).

**Приложение А Листинг кода**

Index.js

const express = require('express');

const sequelize = require('./database');

const properties = require('./dt\_models/properties');

const tasklog = require('./dt\_models/tasklogs');

const app = express();

const PORT = 3000;

// Настройка шаблонизатора EJS

app.set('view engine', 'ejs');

app.set('views', './views');

// Подключение статических файлов

app.use(express.static('./public'));

// Маршрут для главной страницы

app.get('/', (req, res) => {

  console.log('Rendering index.ejs');

  res.render('index'); // Рендерит файл views/index.ejs

});

// Маршрут для первой страницы

app.get('/page1', async (req, res) => {

  console.log('Rendering page1.ejs');

  try {

    // Получение данных из базы

    const data = await properties.findAll({

      attributes: ['source', [sequelize.fn('COUNT', sequelize.col('source')), 'count']],

      group: ['source'],

      order: [[sequelize.fn('COUNT', sequelize.col('source')), 'DESC']]

    });

    // Форматируем данные для передачи в шаблон

    const chartData = data.map(row => ({

      source: row.source,

      count: row.dataValues.count

    }));

    console.log(chartData);

    res.render('page1', { chartData });

  } catch (error) {

    console.error('Ошибка при получении данных:', error);

    res.status(500).send('Ошибка сервера');

  }

});

app.get('/page2', async (req, res) => {

  try {

    // Получение данных для 10 самых популярных станций

    const topStations = await properties.findAll({

      attributes: ['metro', [sequelize.fn('COUNT', sequelize.col('metro')), 'count']],

      group: ['metro'],

      order: [[sequelize.fn('COUNT', sequelize.col('metro')), 'DESC']],

      limit: 10

    });

    // Получение данных для остальных станций

    const otherStations = await properties.findAll({

      attributes: ['metro', [sequelize.fn('COUNT', sequelize.col('metro')), 'count']],

      group: ['metro'],

      order: [[sequelize.fn('COUNT', sequelize.col('metro')), 'DESC']],

      offset: 10

    });

    // Форматируем данные для передачи в шаблон

    const chartData = topStations.map(row => ({

      metro: row.metro,

      count: row.dataValues.count

    }));

    const otherStationsList = otherStations.map(row => ({

      metro: row.metro,

      count: row.dataValues.count

    }));

    res.render('page2', { chartData, otherStationsList });

  } catch (error) {

    console.error('Ошибка при обработке маршрута /page2:', error);

    res.status(500).send('Ошибка сервера');

  }

});

app.get('/page3', async (req, res) => {

  try {

    // Получение данных из базы данных

    const averagePrices = await properties.findAll({

      attributes: ['source', [sequelize.fn('AVG', sequelize.col('price')), 'average\_price']],

      group: ['source'],

      order: [[sequelize.fn('AVG', sequelize.col('price')), 'DESC']]

    });

    // Форматируем данные для передачи в шаблон

    const chartData = averagePrices.map(row => ({

      source: row.source,

      averagePrice: parseFloat(row.dataValues.average\_price).toFixed(2) // Преобразуем в число с двумя знаками после запятой

    }));

    res.render('page3', { chartData });

  } catch (error) {

    console.error('Ошибка при обработке маршрута /page3:', error);

    res.status(500).send('Ошибка сервера');

  }

});

app.listen(PORT, () => {

  console.log(`Server is running on http://localhost:${PORT}`);

});