Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет
имени Н. Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н. Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К КУРСОВОЙ РАБОТЕ

НА ТЕМУ:

«Консенсус прогноз по акциям»

Студент ИУ7-63Б _____ Криков А. В.
(Группа) (Подпись, дата) (И. О. Фамилия)

Руководитель курсовой работы _____ Павельев А. А.
(Подпись, дата) (И. О. Фамилия)

2022 г.

РЕФЕРАТ

В данной работе будет реализовано приложение, которое предоставляет функционал для ознакомления с компаниями и прогназами на курс акций этих компаний. Информацию от приложения можно получить с помощью REST API.

Приложение реализовано с помощью языка программирования Java. В качестве основного фреймворка использовался фреймворк — Spring. Для соединения сервера и базы данных использовалась ORM — Hibernate.

Ключевые слова: PostgreSQL, Redis, Java, Spring, Hibernate, SQL.

Расчетно-пояснительная записка к курсовой работе содержит 38 страниц, 8 иллюстраций, 11 таблиц, 12 источников, 4 приложение.

СОДЕРЖАНИЕ

ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ

В настоящей расчетно-пояснительной записке применяют следующие сокращения и обозначения.

СУБД — система управления базами данных

ORM — Object-Relational Mapping

SQL — Structured Query Language

NoSQL — Not only Structured Query Language

ВВЕДЕНИЕ

В настоящее время количество частных инвесторов растет с каждым годом, так, согласно исследованиям с начала 2021 года число физических лиц, имеющих брокерские счета на Московской бирже увеличилось на 6.2 миллиона человек и достигло 15 миллионов. В большинстве случаев выбор того или иного финансового инструмента у начинающего инвестора зависит от мнений авторитетных аналитиков. Как следствие, возникает необходимость ознакомления с большим количеством различных точек зрений. Ввиду отсутствия должного опыта пользователь вынужден посещать обширное количество интернет-ресурсов. Решением данной проблемы может послужить агрегация множества прогнозов в единое веб-приложение, которое предоставит информацию конечному пользователю в удобном и доступном виде.

Цель работы — спроектировать и реализовать базу данных, содержащую данные о прогнозах на курсы акций.

Чтобы достигнуть поставленной цели, требуется решить следующие задачи:

- проанализировать существующие решения;
- формализовать задание и определить необходимый функционал;
- проанализировать варианты представления данных и выбрать подходящий вариант для решения задачи;
- проанализировать существующие СУБД и выбрать подходящую систему для хранения данных;
- спроектировать базу данных для хранения и структурирования данных, описать ее сущности и связи;
- реализовать программное обеспечение, позволяющее получить доступ к данным с помощью REST API [1].

1 Аналитический раздел

В данном разделе будет проанализирована поставленная задача и рассмотрены различные способы ее реализации.

В ходе выполнения курсовой работы должно быть спроектировано и реализовано Web-приложение, позволяющее пользователю ознакомиться с прогнозами курса акций различных аналитиков. Пользователь должен иметь возможность получения прогнозируемой цены одной из предложенных акций. Кроме того, необходимо разработать ролевую модель, на основе которой будет строиться взаимодействие с веб-приложением. Предусмотреть три роли: пользователь, аналитик и администратор. Также необходимо создать систему регистрации и авторизации пользователей и предусмотреть возможность назначения ролей администратором. Сделать возможным для аналитика добавление, редактирование и удаление прогнозов.

1.1 Формализация данных

В базе данных должна содержаться информация о:

- компаниях;
- прогнозах;
- финансовых показателях компаний;
- новостях компаний;
- пользователях;
- группах пользователей;
- правах доступа пользователей.

Таблица 1.1 – Категории данных и сведения о них

Категория	**Сведения**
Компания	ID компании, название, логотип, описание, тикер, ID показателей
Новость	ID новости, заголовок, дата публикации, наполнение, ссылка на источник, автор, ID компании
Прогноз	ID прогноза, инвест-дом, дата публикации, дата обновления, дата истечения, целевая цена, прогноз, описание прогноза, ID компании
Финансовые показатели	ID показателей, цена, капитализация, объем прибыли до вычета налогов, годовая выручка
Пользователь	ID пользователя, имя пользователя в системе, пароль, электронная почта, полное имя
Группа пользователей	ID записи, ID прав доступа, ID пользователя
Права доступа	ID записи, права доступа

1.2 Формализация ролей

Должно быть выделено три категории пользователей.

Таблица 1.2 – Роли пользователей и предоставляемый им функционал

Роль	**Функционал**
Пользователь	просмотр компаний и информации о них, просмотр предоставленных прав доступа просмотр прогнозов на выбранную компанию, просмотр новостей о компании
Аналитик	добавление, просмотр, редактирование, удаление компаний, информации о них, а также прогнозов и новостей на выбранную компанию
Администратор	просмотр, добавление, редактирование, удаление выбранных пользователей и прав доступа

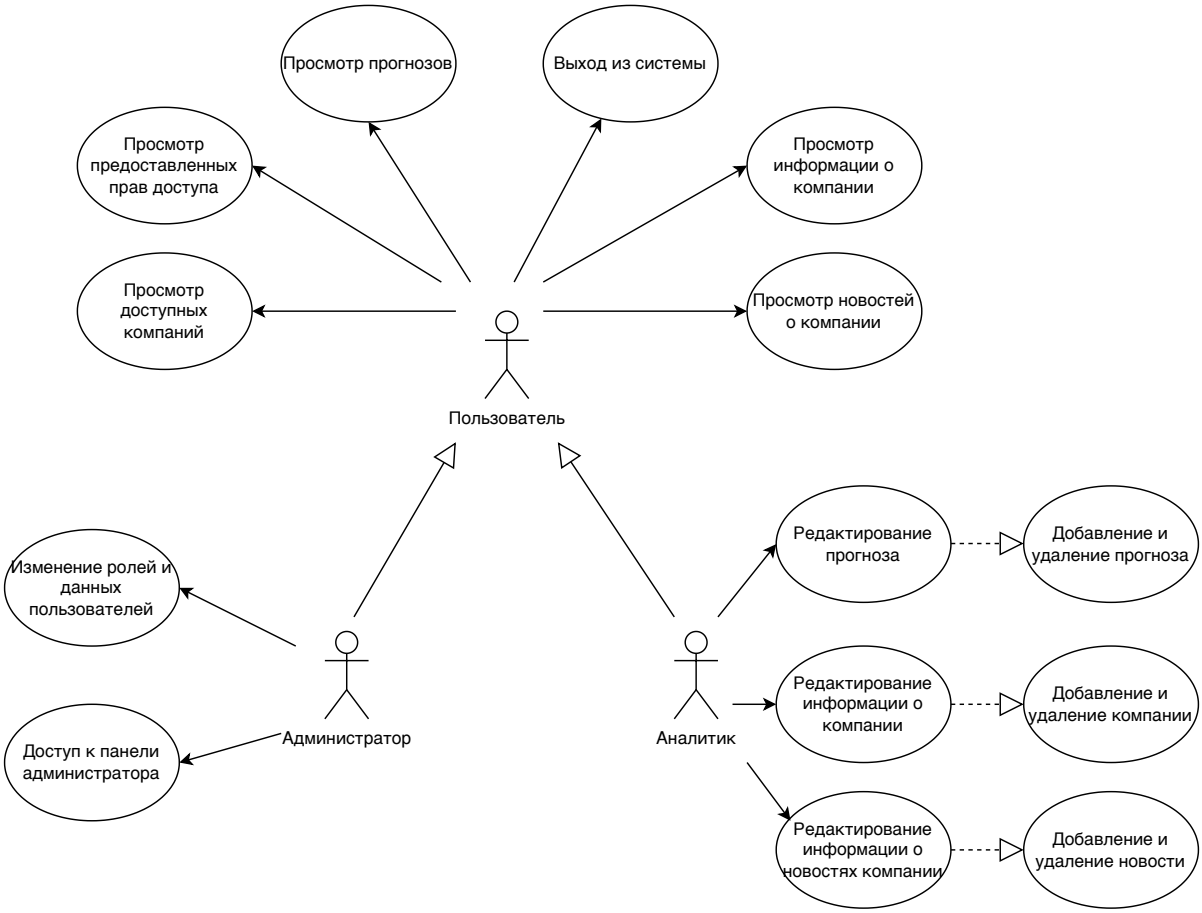

Рисунок 1.1 – Диаграмма сценариев для пользователей, аналитиков и администратора

1.3 Формализация сущностей системы

На рисунках 1.2 и 1.3 отображены диаграммы сущностей системы. Данные диаграммы построены на основе данных таблицы 1.1.

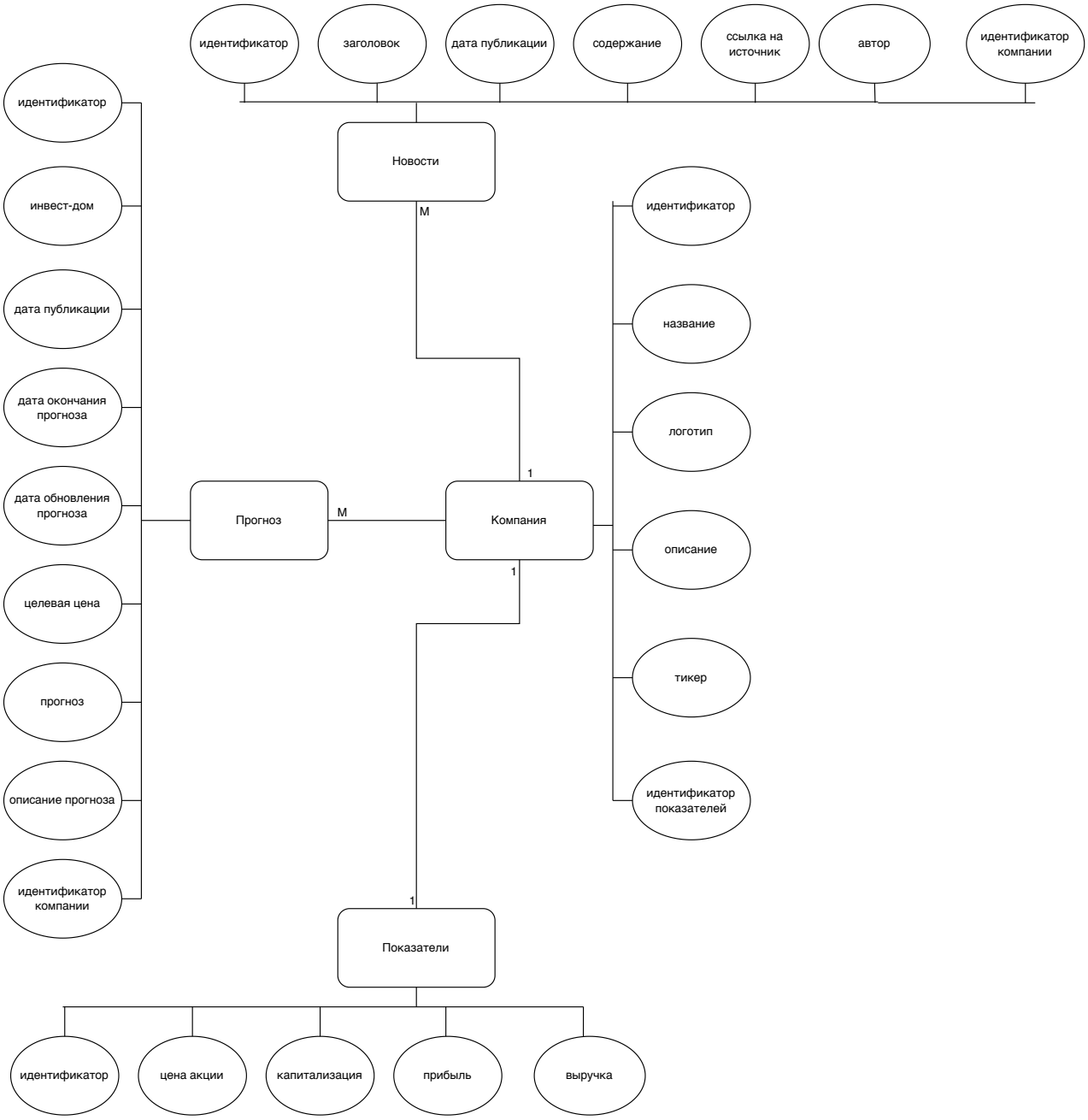

Рисунок 1.2 – ER-диаграмма сущностей базы данных в нотации Чена (1)

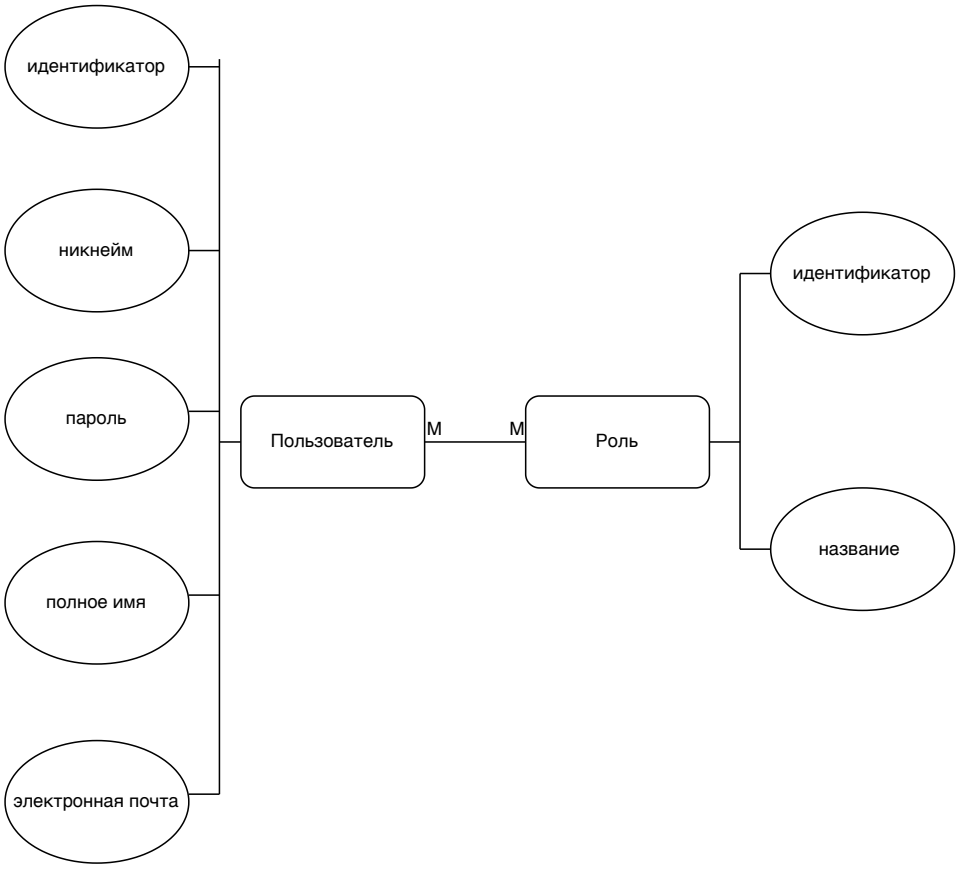

Рисунок 1.3 – ER-диаграмма сущностей базы данных в нотации Чена (2)

1.4 Выбор модели хранения данных

Модель данных — это абстрактное, логическое определение объектов, операторов и прочих элементов, в совокупности состовляющих абстрактную машину доступа к данным, с которой взаимодействует пользователь. Эти объекты позволяют моделировать структуру данных, а операторы — поведение данных [2].

Существуют три основные модели данных:

- иерархическая — база данных с древовидной структурой с дугами — связями и узлами — элементами данных. Основной недостаток данной модели — подобная структура не удовлетворяет требованиям многих задач [3];
- сетевая. В сетевой модели наряду с вертикальными связями допустимы и горизонтальные связи. Главный недостаток данной модели — необходимость четко определять на физическом уровне связи данных и столь

же четко следовать этой структуре свзяей при запросах к базе;

- реляционная. Главное отличие состоит в том, что в данной модели информация хранится в виде таблиц, состоящих из нескольких записей — кортежей, обладающих одним и тем же набором атрибутов или полей.

Реляционная модель данных имеет ряд преимуществ в сравнении с остальными рассмотренными моделями, поскольку она более гибкая и удобная в использовании. Также она лучше всего соответствует описанным ранее связям между сущностями.

1.5 Анализ существующих решений

На сегодняшний день имеются различные сервисы, решающие поставленную цель в разной степени. Для их анализа введем следующие критерии:

- детализация — приведение обширного количества информации о рассматриваемой компании;
- аргументация — автор прогноза обосновывает свой выбор;
- объективность — автор относится к рассматриваемой компании безэмоционально;
- платный доступ — просмотр прогнозов некторых компаний ограничен в бесплатной версии.

В качестве существующих решений для анализа выбраны сервисы Tinkoff Инвестиции, RBC Инвестиции, Investing.com, Finam. В таблице 1.3 представлен сравнительный анализ существующих решений.

Таблица 1.3 – Анализ существующих решений

Сервис	**Детализазация**	**Аргументация**	**Объективность**	**Платный доступ**
Tinkoff Инвестиции	Да	Нет	Да	Нет
RBC Инвестиции	Нет	Нет	Нет	Нет
Investing.com	Да	Нет	Да	Да
Finam	Да	Да	Нет	Нет

Вывод

В данном разделе была рассмотрена структура поставленной задачи, формализованны данные, используемые в системе, а также приведен анализ существующих решений.

2 Конструкторская часть

В данном разделе представлены этапы проектирования баз данных. Первоначально рассматриваются формализуются сущности системы, то есть объекты переносятся из реального мира в прогармму. Далее приводится диаграмма спроектированной базы данных. Также рассматривается модель базы данных с включенным механизмом кэширования. Описывается архитектура приложения.

2.1 Проектирование базы данных

В соответствии с таблицей 1.1 база данных должна содержать следующие таблицы:

Таблица 2.1 – Company (таблица компаний)

Атрибут	**Тип**	**Значение**
id	Целое число	Идентификатор, PK
name	Строка	Название
logo	Строка	Путь к логотипу
description	Строка	Описание
ticker	Строка	Тикер
indicators_id	Целое число	Идентификатор показателей для соответствующей компании, FK

Таблица 2.2 – Forecast (таблица прогнозов)

Атрибут	**Тип**	**Значение**
id	Целое число	Идентификатор, PK
invest_house	Строка	Инвест-дом
date_publishing	Дата	Дата публикации
date_end	Дата	Дата окончания
date_update	Дата	Дата обновления
goal_price	Вещественное число	Целевая цена
forecast	Строка	Прогноз
description	Строка	Описание
company_id	Целое число	Идентификатор компании для соответствующего прогноза, FK

Таблица 2.3 – Indicators (таблица финансовых показателей)

Атрибут	**Тип**	**Значение**
id	Целое число	Идентификатор, PK
price	Вещественное число	Цена
market_cap	Вещественное число	Капитализация
income	Целое число	Прибыль от выручки выраженная в процентах
revenue	Вещественное число	Выручка

Таблица 2.4 – News (таблица новостей)

Атрибут	**Тип**	**Значение**
id	Целое число	Идентификатор, PK
title	Строка	Заголовок
date_publishing	Дата	Дата публикации
content	Строка	Наполнение
url	Строка	Ссылка на источник
author	Строка	Автор новости
company_id	Целое число	Идентификатор компании для соответствующей новости, FK

Таблица 2.5 – User (таблица пользователей)

Атрибут	**Тип**	**Значение**
id	Целое число	Идентификатор, PK
username	Строка	Никнейм пользователя
password	Строка	Пароль пользователя, хранящийся в шифрованном виде
email	Строка	Электронная почта
name	Строка	Имя
surname	Строка	Фамилия

Таблица 2.6 – Role (таблица ролей)

Атрибут	**Тип**	**Значение**
id	Целое число	Идентификатор, PK
role	Строка	Роль

Таблица 2.7 – Users_roles (таблица для связи пользователей и их ролей)

Атрибут	**Тип**	**Значение**
id	Целое число	Идентификатор, PK
user_id	Целое число	Идентификатор пользователя, FK
role_id	Целое число	Идентификатор роли, FK

На рисунке 2.1 представлена диаграмма спроектированной базы данных.

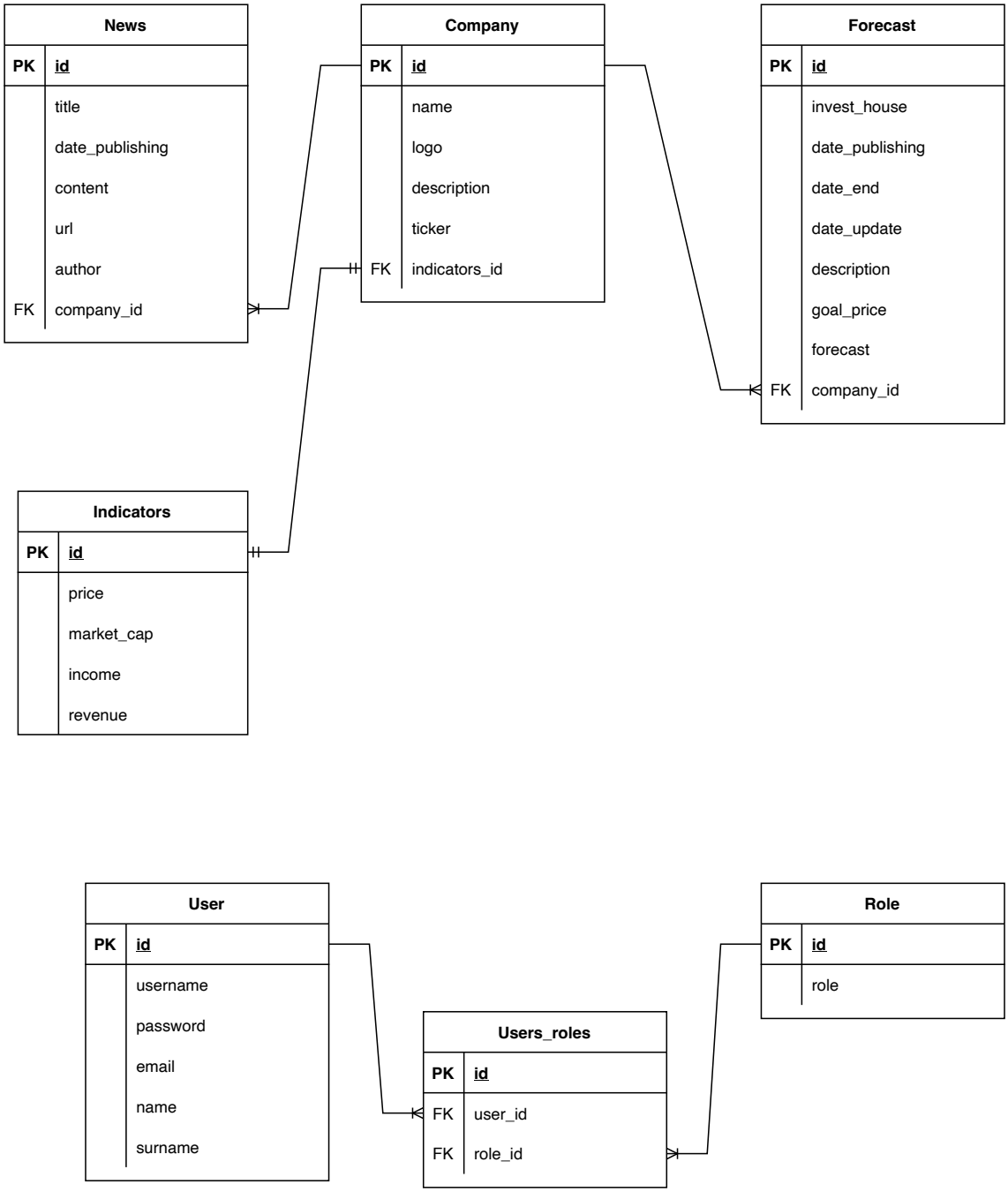

Рисунок 2.1 – Диаграмма спроектированной базы данных

Помимо этого, для всех таблиц будут реализованы триггеры, срабатывающие в момент удаления или обновления данных из таблиц. Данные триггеры служат для синхронизации данных в базе данных и в кэше.

2.2 Проектирование базы данных кэширования

База данных кэширования будет реализована с использованием Redis. Redis — это нереляционная СУБД, хранящая данные в виде пар «ключ-значение».

В базе данных будут полностью продублированны таблицы из основной базы данных. Первичным ключом будет являться поле с уникальным идентификатором (`id`). При повторном запросе данных у приложения сначала будет проводиться проверка, присутствует ли запись в кэше. В случае, если запись присутствует, запрос к основной базе данных производиться не будет и будут возвращены данные из кэша. В противном случае, будет произведен запрос к основной базе данных.

2.3 Проектирование приложения

Разрабатываемое приложение представляет собой многокомпонетное веб-приложение. Данные приложения будут получены с помощью REST API. Сервер обменивается данными с базами данных при помощи коннекторов, позволяющих делать запросы к базе данных с помощью языка программирования. Схема архитектуры приложения представлена на рисунке 2.2.

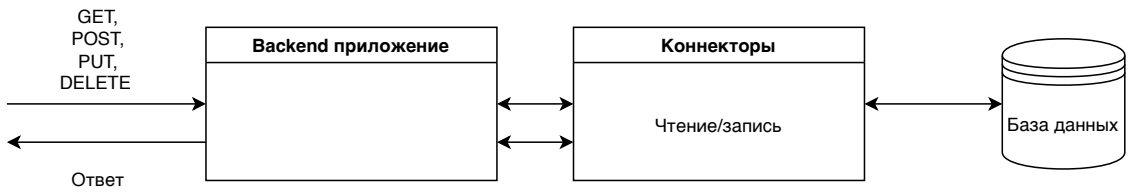

Рисунок 2.2 – Схема архитектуры спроектированного приложения

Вывод

В данном разделе были представлены этапы проектирования баз данных и рассмотрена архитектура программного обеспечения.

3 Технологическая часть

В данном разделе производится выбор средств разработки программного обеспечения, раскрываются детали реализации и представляются способы взаимодействия с программным обеспечением. Также приводится листинг некоторых компонентов приложения. В частности рассмтриваются модели данных и приводится пример метода, с помощью которого осуществляется доступ к данным.

3.1 Выбор СУБД

СУБД — это система управления базами данных. Так называют комплекс программно-языковых средств, позволяющих создавать базы данных и манипулировать данными [4].

Среди самых популярных реляционных СУБД можно выделить следующие:

- MySQL — СУБД с открытым исходным кодом. Является одной из самых популярных реляционных СУБД. Главное недостаток данной СУБД — фрагментарное использование SQL;
- MS SQL Server — СУБД, разрабатываемая компанией Microsoft. Основным недостатком является платный доступ и повышенное потребление ресурсов;
- SQLite — компактная встраиваемая СУБД с открытым исходным кодом. Недостатком является отсутствие системы пользователей, что недопустимо для поставленной задачи;
- Oracle Database — объектно-реляционная СУБД компании Oracle. Данная СУБД подходит для разнообразных рабочих нагрузок и может использоваться практически в любых задачах. Однако основными ее недостатками являются сложность в настраивании и повышенное потребление ресурсов;
- PostgreSQL — современная СУБД с открытым исходным кодом. СУБД предоставляет транзакции со свойствами согласованности, изоляции, долговечности [5].

Для решения задачи была выбрана СУБД PostgreSQL, поскольку данная СУБД является наиболее развитой из открытых СУБД, кроме того Postgre проста в развертывании.

3.2 Выбор языка программирования

Для разработки серверной части был выбран язык программирования Java. Данный выбор обусловлен тем, что Java — кросплатформенный, компилируемый язык программирования, поэтому разрабатанное ПО можно будет запустить на любой ОС.

Для реализации REST API был выбран фреймворк Spring. Для коммуникации серверной части приложения и базами данных была выбрана JPA спецификация.

Для «упаковки» приложения в готовый продукт была выбрана система контейнеризации Docker [6].

Тестирование программного обеспечения производилось с помощью фреймворка JUnit. С помощью данного фреймворка можно писать как модульные, так и функциональные тесты.

3.3 Детали реализации

В листинге 3.1 представлена сущность базы данных — Company.

Листинг 3.1 – Сущность Company

```
package com.example.consensus.entities;

import com.fasterxml.jackson.annotation.JsonIgnore;
import lombok.Data;
import javax.persistence.*;
import javax.validation.constraints.NotBlank;
import javax.validation.constraints.Size;
import java.util.List;

@Table
@Entity
@Data
public class Company {
@Id
@GeneratedValue(strategy = GenerationType.IDENTITY)
private Long id;

@Column(name = "name")
private String name;

@Column(name = "logo")
private String logo;

@Column(name = "description")
private String description;

@Column(name = "ticker")
@NotBlank
@Size(min = 1, max = 6, message = "Ticker␣length␣may␣be␣between␣1
   ␣and␣6")
private String ticker;

@OneToMany(mappedBy = "companyForForecasts")
private List<Forecast> forecasts;

@OneToMany(mappedBy = "companyForNews")
private List<News> news;

@OneToOne(cascade =CascadeType.ALL)
```

```
@JoinColumn(name = "indicators_id", referencedColumnName = "id")
private Indicators indicators;

public void setFields(Company company) {
        setName(company.getName());
        setLogo(company.getLogo());
        setDescription(company.getDescription());
        setTicker(company.getTicker());
        setForecasts(company.getForecasts());
        setIndicators(company.getIndicators());
}
}
```

В листинге 3.2 представлена сущность базы данных — Forecast.

Листинг 3.2 – Сущность Forecast

```
package com.example.consensus.entities;

import com.fasterxml.jackson.annotation.JsonIgnore;
import lombok.Data;
import javax.persistence.*;
import javax.validation.constraints.NotNull;
import java.sql.Timestamp;

@Table
@Entity
@Data
public class Forecast {
        @Id
        @GeneratedValue(strategy = GenerationType.IDENTITY)
        @NotNull
        private long id;

        @Column(name = "invest_house")
        private String investHouse;

        @Column(name = "date_publishing")
        private Timestamp datePublishing;

        @Column(name = "date_end")
        private Timestamp dateEnd;

        @Column(name = "date_update")
        private Timestamp dateUpdate;

        @Column(name = "goal_price")
        private int goalPrice;

        @Column(name = "forecast")
        private String forecast;

        @Column(name = "description")
        private String description;

        @ManyToOne
```

```
        @JoinColumn(name = "company_id", nullable = false)
        @JsonIgnore
        private Company companyForForecasts;

        public void setFields(Forecast forecast) {
                setInvestHouse(forecast.getInvestHouse());
                setDatePublishing(forecast.getDatePublishing());
                setDateEnd(forecast.getDateEnd());
                setDateUpdate(forecast.getDateUpdate());
                setGoalPrice(forecast.getGoalPrice());
                setForecast(forecast.getForecast());
                setDescription(forecast.getDescription());
                setCompanyForForecasts(forecast.
                   getCompanyForForecasts());
        }
}
```

В листинге 3.3 представлена сущность базы данных — Indicators.

Листинг 3.3 – Сущность Indicators

```
package com.example.consensus.entities;


import com.fasterxml.jackson.annotation.JsonIgnore;
import lombok.Data;
import javax.persistence.*;
import javax.validation.constraints.Min;
import javax.validation.constraints.NotNull;
import javax.validation.constraints.Size;

@Table
@Entity
@Data
public class Indicators {
        @Id
        @GeneratedValue(strategy = GenerationType.IDENTITY)
        @NotNull
        private long id;

        @Column(name = "price")
        @Min(value = 0, message = "Price␣can␣not␣be␣less␣then␣0!"
           )
```

```
        private double price;

        @Column(name = "market_cap")
        @Min(value = 0, message = "Market␣cap␣can␣not␣be␣less␣
           then␣0!")
        private int market_cap;

        @Size(min = 0, max = 100, message = "Income␣must␣be␣
           expressed␣in␣%")
        @Column(name = "income")
        private int income;

        @Min(value = 0, message = "Revenue␣must␣be␣more␣then␣0!")
        @Column(name = "revenue")
        private int revenue;

        @OneToOne(mappedBy = "indicators")
        @JsonIgnore
        private Company company;

        public void setFields(Indicators newIndicators) {
                setPrice(newIndicators.getPrice());
                setMarket_cap(newIndicators.getMarket_cap());
                setIncome(newIndicators.getIncome());
                setRevenue(newIndicators.getRevenue());
                setCompany(newIndicators.getCompany());
        }
}
```

В листинге 3.4 представлена сущность базы данных — News.

Листинг 3.4 – Сущность News

```
package com.example.consensus.entities;


import com.fasterxml.jackson.annotation.JsonIgnore;
import lombok.Data;
import javax.persistence.*;
import java.sql.Timestamp;

@Entity
@Table(name = "news")
```

```
@Data
public class News {
        @Id
        @GeneratedValue(strategy = GenerationType.IDENTITY)
        private Long id;

        @Column(name = "title")
        private String title;

        @Column(name = "date_public")
        private Timestamp datePublic;

        @Column(name = "content")
        private String content;

        @Column(name = "url")
        private String url;

        @Column(name = "author")
        private String author;

        @ManyToOne
        @JoinColumn(name = "company_id", nullable = false)
        @JsonIgnore
        private Company companyForNews;

        public void setFields(News news) {
                setAuthor(news.getAuthor());
                setContent(news.getContent());
                setUrl(news.getUrl());
                setCompanyForNews(news.getCompanyForNews());
                setDatePublic(news.getDatePublic());
                setTitle(news.getTitle());
        }
}
```

В листинге 3.5 представлена сущность базы данных — User.

Листинг 3.5 – Сущность User

```
package com.example.consensus.entities;

import lombok.Data;
```

```
import javax.persistence.*;
import javax.validation.constraints.Email;
import java.util.Collection;

@Entity
@Data
@Table(name = "users")
public class User {

        @Id
        @GeneratedValue(strategy = GenerationType.IDENTITY)
        private Long id;

        @Column(name = "username")
        private String username;

        @Column(name = "password")
        private String password;

        @Email
        @Column(name = "email")
        private String email;

        @Column(name = "name")
        private String name;

        @Column(name = "surname")
        private String surname;

}
```

В листинге 3.6 представлена сущность базы данных — Role.

Листинг 3.6 – Сущность Role

```
package com.example.consensus.entities;

import lombok.Data;
import javax.persistence.*;

@Entity
@Table(name = "roles")
@Data
```

```
public class Role {
        @Id
        @GeneratedValue(strategy = GenerationType.IDENTITY)
        private Long id;

        @Column(name = "name")
        private String name;
}
```

В листингах 3.7 – 3.8 приведены примеры взаимодействия клиента с сервером.

Листинг 3.7 – Пример реализации взаимодействия клиента с сервером

```
package com.example.consensus.controllers;

import com.example.consensus.entities.Forecast;
import com.example.consensus.services.ForecastService;
import org.springframework.web.bind.annotation.*;

import java.util.List;

@RestController
@RequestMapping("/api/v1")
public class ForecastController {
        private final ForecastService forecastService;

        public ForecastController(ForecastService forecastService
           ) {
                this.forecastService = forecastService;
        }

        @GetMapping("/company/{id}/forecasts")
        public List<Forecast> getAllForecastsByCompanyId(
           @PathVariable(name="id") Long id) {
                return forecastService.getAllForecastsByCompanyId
                   (id);
        }

        @GetMapping("/forecast/{id}")
        public Forecast getForecastById(@PathVariable(name = "id"
           ) Long id) {
                return forecastService.getForecastById(id);
```

```
        }

        @PutMapping("/forecast/{id}")
        public Forecast updateForecastById(@PathVariable(name = "
           id") Long id, @RequestBody Forecast forecastDetails) {
                return forecastService.updateForecastById(id,
                   forecastDetails);
        }

        @PostMapping("/forecast/")
        public Forecast addForecast(@RequestBody Forecast
           forecast) {
                return forecastService.addForecast(forecast);
        }

        @DeleteMapping("/forecast/{id}")
        public Forecast deleteForecast(@PathVariable(name = "id")
            Long id) {
                return forecastService.deleteForecast(id);
        }
}
```

Листинг 3.8 – Пример реализации взаимодействия клиента с сервером

```
package com.example.consensus.controllers;

import com.example.consensus.entities.Company;
import com.example.consensus.services.CompanyService;
import org.springframework.http.HttpStatus;
import org.springframework.web.bind.annotation.*;

import javax.validation.Valid;
import java.util.List;

@RestController
@RequestMapping("/api/v1")
public class CompanyController {
        private final CompanyService companyService;

        public CompanyController(CompanyService companyService) {
                this.companyService = companyService;
        }
```

```
        @GetMapping("/all")
        public List<Company> getAllCompanies() {
                return companyService.getAllCompanies();
        }

        @GetMapping("/company/")
        public Company getCompanyByName(@RequestParam(value = "
           name") String name) {
                return companyService.getCompanyByName(name);
        }

        @GetMapping("/company/{id}")
        public Company getCompanyById(@PathVariable Long id) {
                return companyService.getCompanyById(id);
        }

        @PutMapping("/company/{id}")
        public Company updateCompany(@PathVariable Long id,
           @RequestBody Company companyDetails) {
                return companyService.updateCompanyById(id,
                   companyDetails);
        }

        @PostMapping("/company/add")
        public Company addCompany(@Valid @RequestBody Company
           company) {
                return companyService.addCompany(company);
        }

        @ResponseStatus(value = HttpStatus.OK)
        @DeleteMapping("/company/{id}")
        public Company deleteCompany(@PathVariable Long id) {
                return companyService.deleteCompany(id);
        }
}
```

В листинге 3.9 представлена инициализация таблиц базы данных.

Листинг 3.9 – Создание таблиц

```
CREATE TABLE IF NOT EXISTS Indicators (
        id serial primary key ,
```

```
        price real,
        market_cap int,
        income int,
        revenue int
);

CREATE TABLE IF NOT EXISTS Company (
        id serial primary key,
        name varchar,
        logo varchar,
        description varchar,
        ticker varchar,
        indicators_id int,
        FOREIGN KEY (indicators_id) REFERENCES Indicators(id) ON
           DELETE CASCADE
);

CREATE TABLE IF NOT EXISTS Forecast (
        id serial primary key ,
        invest_house varchar,
        date_publishing timestamp,
        date_end timestamp,
        date_update timestamp,
        goal_price int,
        forecast varchar,
        description varchar,
        company_id int,
        FOREIGN KEY (company_id) REFERENCES Company(id) ON DELETE
            CASCADE
);

CREATE TABLE IF NOT EXISTS News (
        id serial primary key,
        title varchar,
        date_public timestamp,
        content varchar,
        url varchar,
        author varchar
);

CREATE TABLE IF NOT EXISTS Users (
```

```
                id bigserial primary key,
                username varchar not null ,
                password varchar,
                email varchar,
                name varchar,
                surname varchar
);

CREATE TABLE IF NOT EXISTS Roles (
                id bigserial primary key,
                name varchar not null
);
```

3.4 Взаимодействие с приложением

На рисунках 3.1 – 3.3 представлены примеры запросов к приложению и ответы.

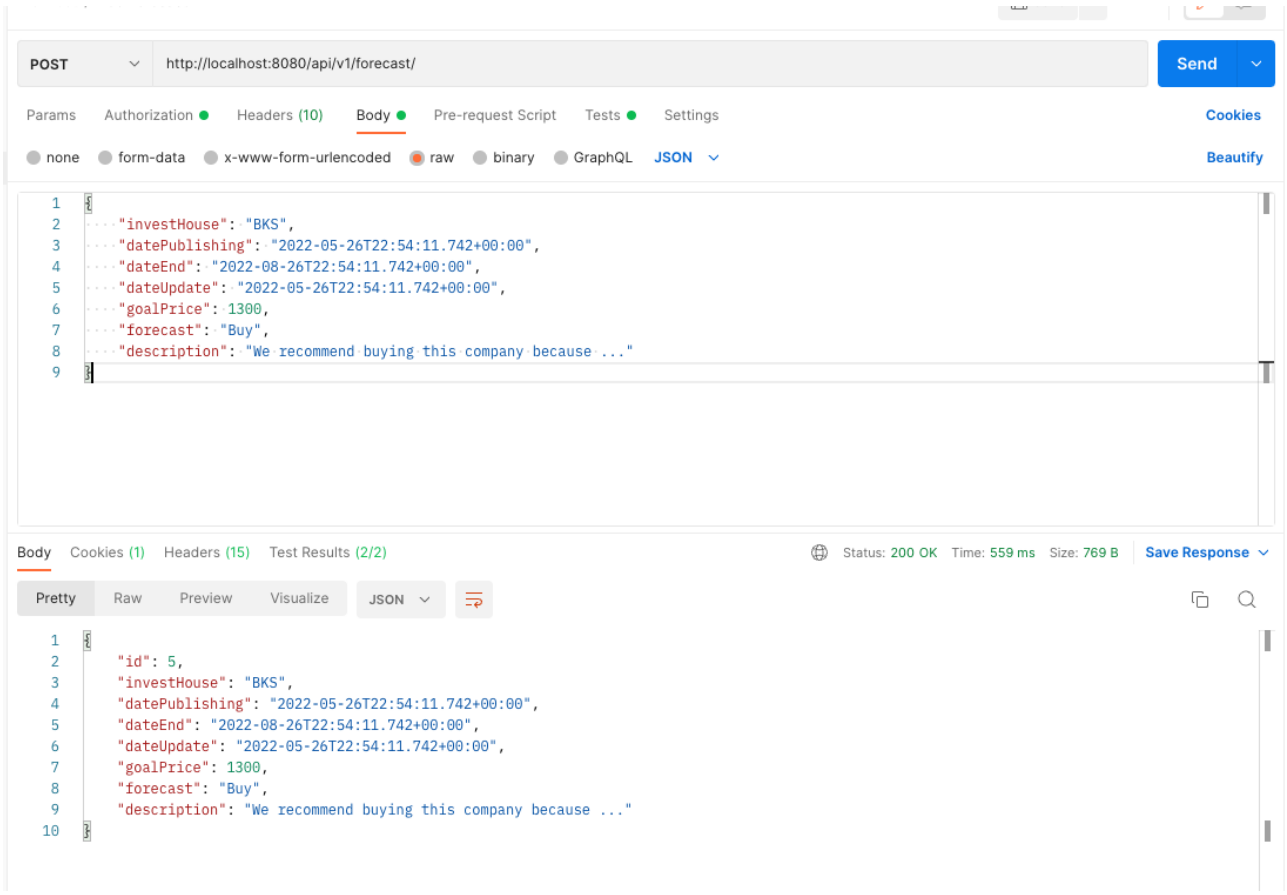

Рисунок 3.1 – Пример добавления прогноза с помощью POST запроса.

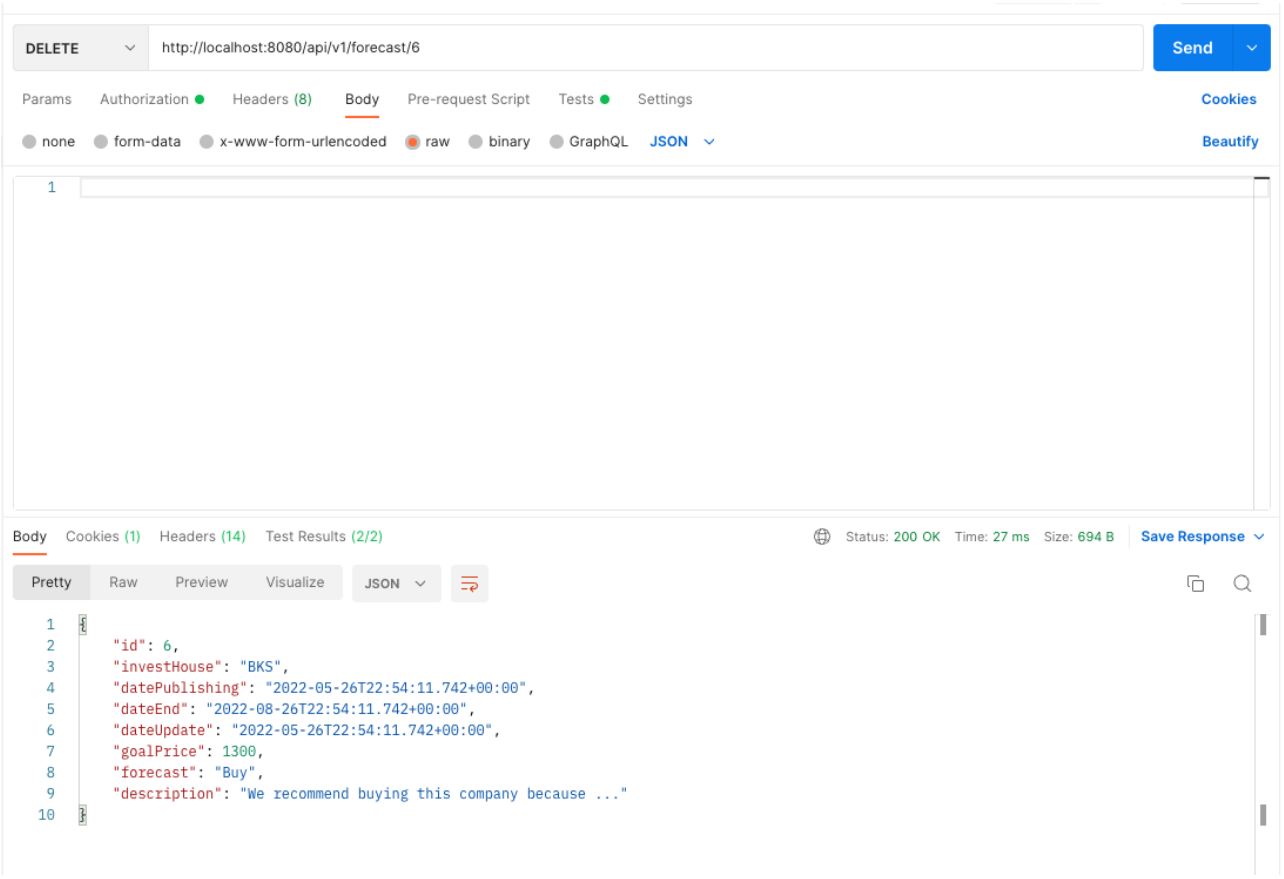

Рисунок 3.2 – Пример удаления прогноза с помощью DELETE запроса.

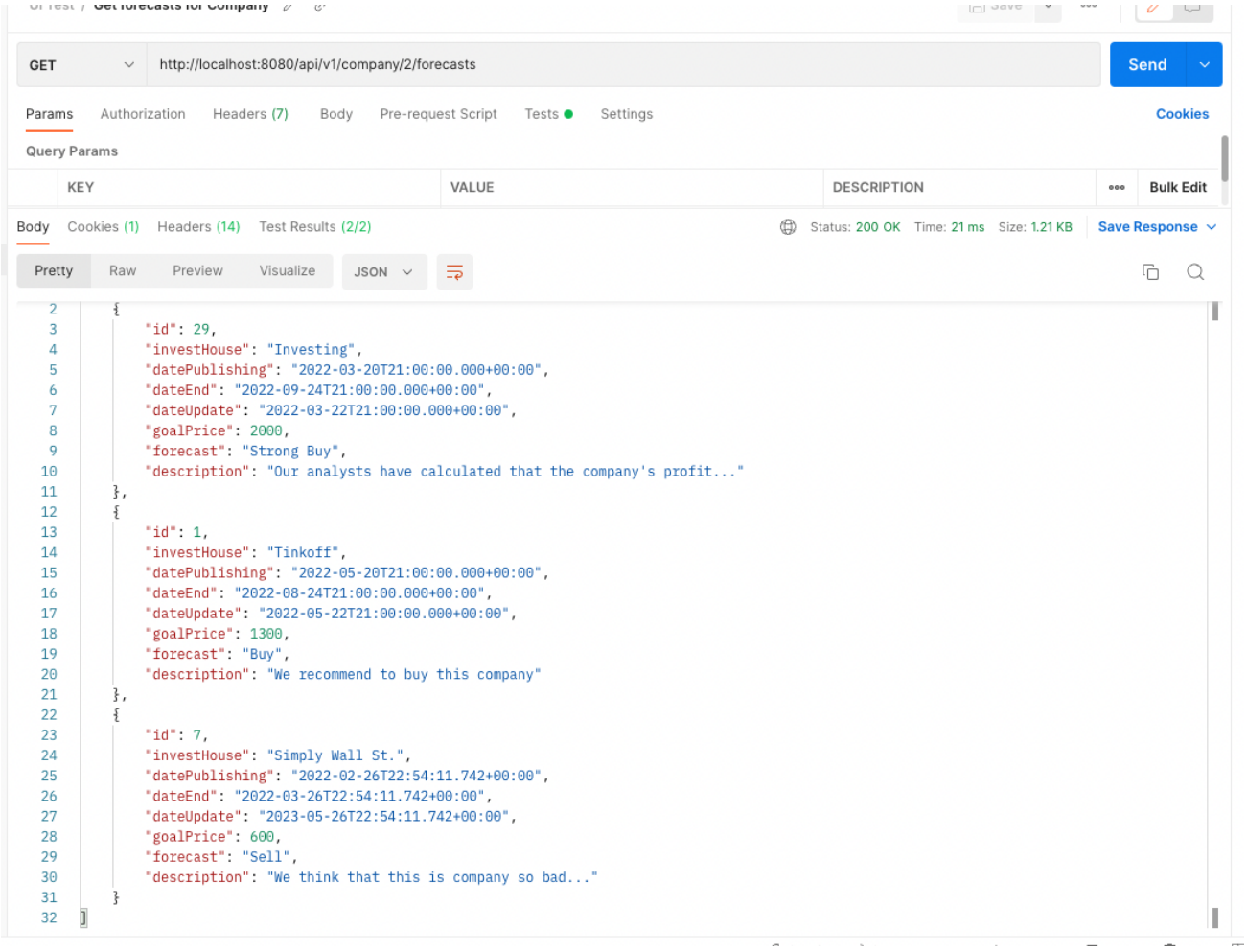

Рисунок 3.3 – Получение всех прогнозов на компанию с `id = 2`

Вывод

В данном разделе были представлены средства реализации программного обеспечения, листинги ключевых компонентов системы, а также была продемонстрирована работоспособность программного обеспечения.

4 Исследовательская часть

В данном разделе проводится эксперимент, в котором сравниваются временные характеристики получения данных из таблицы прогнозов с помощью GET запроса из базы данных с использованим механизма кэширования и без использования механизма кэширования.

Цель эксперимента — сравнить время получения данных с помощью GET запроса о всех прогназах компании с использованием кэширования и без использования кэширования.

4.1 Описание эксперимента

Для сравнения времени запросов с использованием кэширования и без использования кэширования будут выполнены следующие действия:

1. Замерим время выполнения запроса к базе данных, отключив механизм кэширования. При этом количество прогнозов в базе данных равняется n элементам.

2. Замерим время выполнения запроса к базе данных, включив механизм кэширования. При этом количество прогнозов в базе данных равняется n элементам.

3. Повторим пункты 1. и 2. для n = 10, 50, 100, 500, 1000.

4.2 Результат эксперимента

В таблице 4.1 представлены результаты проведенного эксперимента.

Таблица 4.1 – Результаты сравнения времени выполнения запроса при включенном и отключенном механизме кэширования

Количество прогнозов	**Время без кэширования, мс**	**Время с кэшированием, мс**
10	43.643	7.72
50	203.153	8.41
100	310.921	7.84
500	1636.152	9.26
1000	3152.685	10.63

Вывод

На основе результата сравнения времени запроса при включенном и отключенном механизме кэширования можно сделать вывод о том, что приложение с кэшированием данных работает приблизительно в 40 раз быстрее. Данный результат обусловлен тем, что база данных с кэшем уже хранит данные для ответа на запрос. За счет того, что асимптотическая сложность поиска по индексу равна $O(1)$ при любом количестве элементов будет получен одинаковый результат замеренного времени. Однако такой результат можно получить только тогда, когда выборочные данные находятся в кэше, что является не всегда выполнимым условием.

ЗАКЛЮЧЕНИЕ

Во время выполнения курсового проекта были рассмотрены существующие СУБД, спроектирована и реализована база данных для хранения, редактирования и удаления данных о компаниях и прогнозах. Для решения задачи был выбран язык программирования и фреймворк для создания веб-приложения.

В ходе выполнения исследовательской части были получены результаты, с помощью которых можно сказать, что приложение с включенным механизмом кэширования данных работает быстрее чем приложение без механизма кэширования данных.

Также был получен опыт работы с PostgreSQL, изучены возможности языка Java и получены знания в области баз данных.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. What is a REST API? - Red Hat [Электронный ресурс]. — Режим доступа: `https://www.redhat.com/en/topics/api/what-is-a-rest-api` (дата обращения: 07.05.2022).
2. *Левонисова С. В.* Базы данных //. — Издательский дом Москва, 2005.
3. *Лазицкас Е. А.* Базы данных и системы управления базами данных. //. — Издательский дом Москва, 2016.
4. *Г.А. Петров С. Т.* Базы данных, учебное пособие. //. — Издательский дом Москва, 2008.
5. PostgreSQL: Документация. [Электронный ресурс]. — Режим доступа: `https://postgrespro.ru/docs/postgresql/` (дата обращения: 07.05.2022).
6. Redis is an open source (BSD licensed), in-memory data structure store, used as a database, cache, and message broker [Электронный ресурс]. — Режим доступа: `https://redis.io/` (дата обращения: 07.05.2022).