

Оглавление

Введение	2
1 Аналитическая часть	3
1.1 Формализация задачи	3
1.2 Формализация данных	3
1.3 Формализация ролей	4
1.4 Формализация сущностей системы	6
1.4.1 Выбор модели хранения данных	7
1.5 Анализ существующих решений	8
2 Конструкторская часть	10
2.1 Проектирование базы данных	10
2.2 Проектирование базы данных кэширования	13
2.3 Проектирование приложения	13
3 Технологическая часть	15
3.1 Средства реализации	15
3.1.1 Выбор СУБД	15
3.1.2 Выбор языка программирования	16
3.2 Детали реализации	16
3.3 Взаимодействие с приложением	25
Литература	29

Введение

В настоящее время количество частных инвесторов растет с каждым годом, так, согласно исследованиям [1] 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 зрения. 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 для добавления, хранения, редактирования и удаления данных 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 [2].

1 Аналитическая часть

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

1.1 Формализация задачи

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.

1.2 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

1.3 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

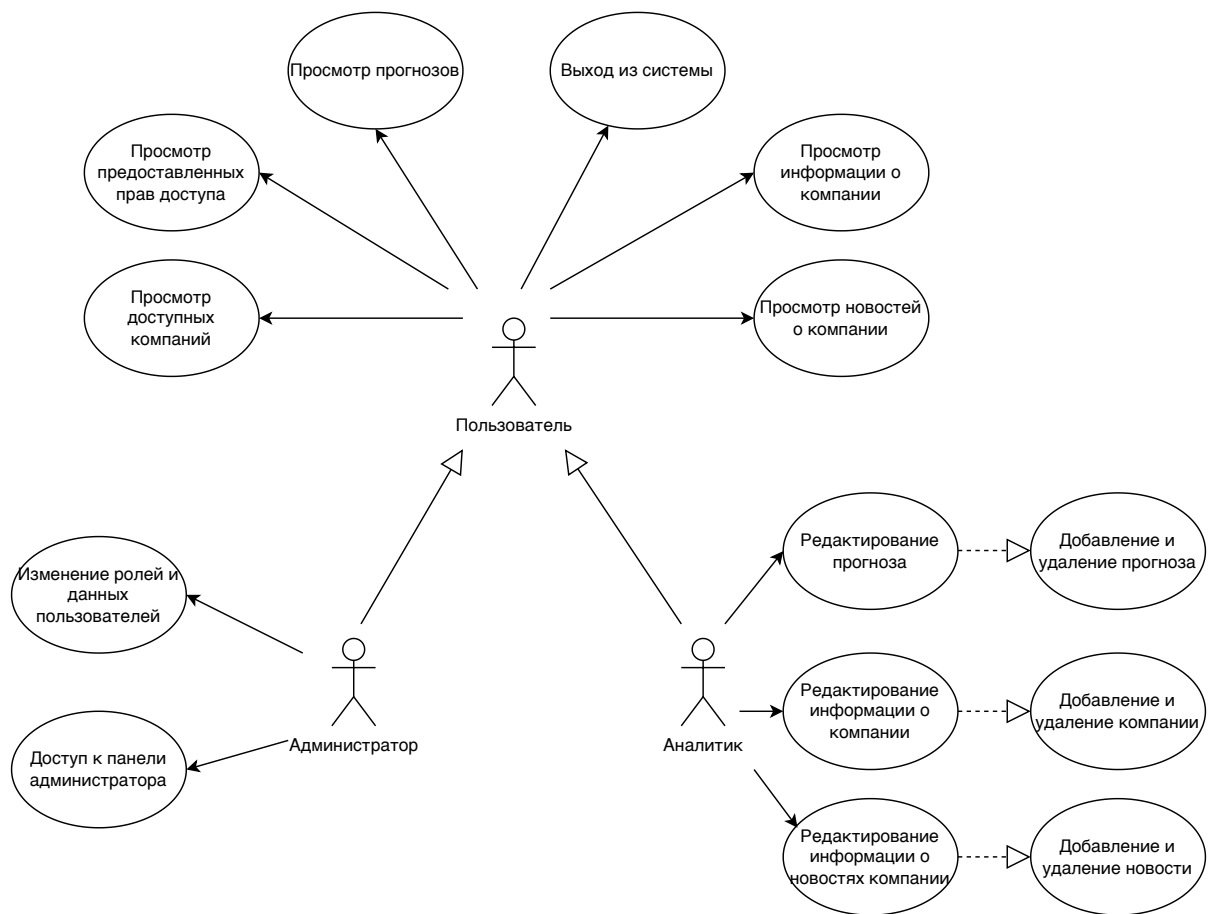


Рис. 1.1: Диаграмма сценариев для пользователей, аналитиков и администратора

1.4 Формализация сущностей системы

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

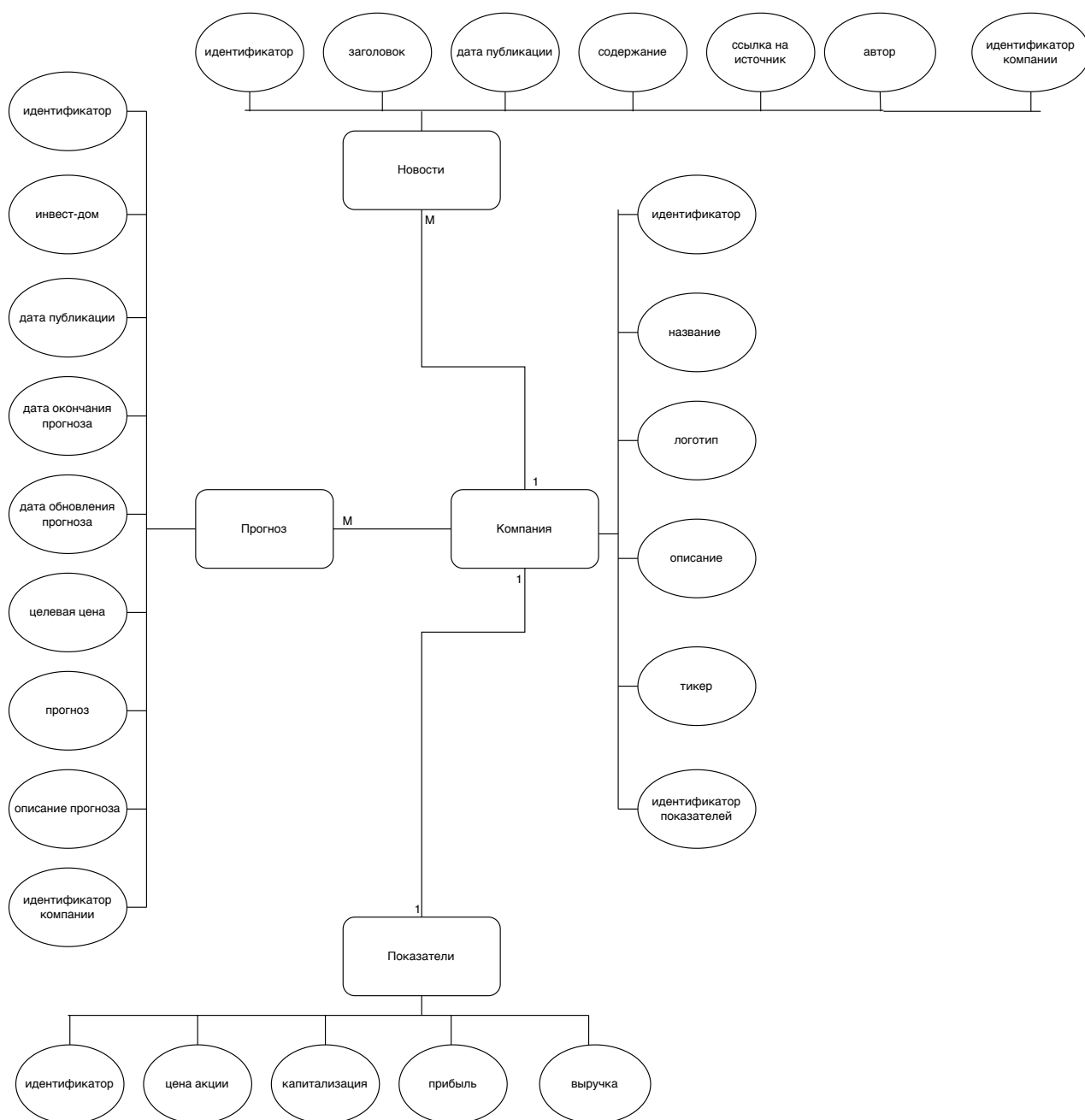


Рис. 1.2: ER-диаграмма сущностей базы данных в нотации Чена (1)

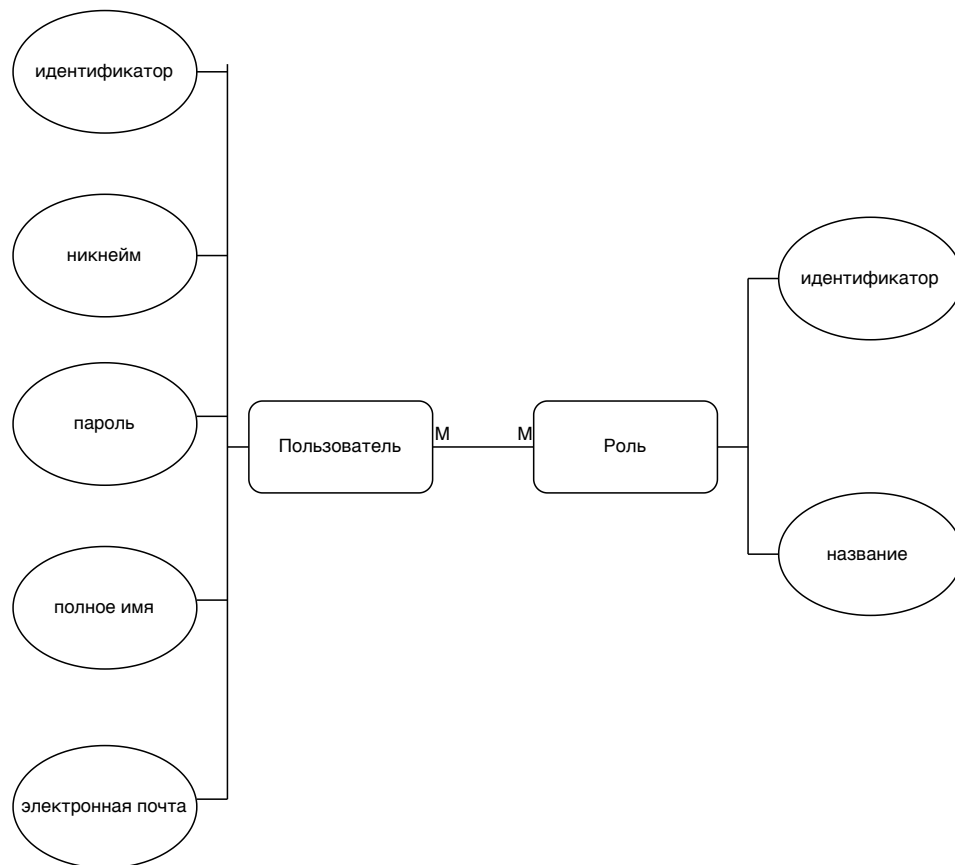


Рис. 1.3: ER-диаграмма сущностей базы данных в нотации Чена (2)

1.4.1 Выбор модели хранения данных

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

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;
- сетевая. В сетевой модели наряду с вертикальными связями допустимы и горизонтальные связи. Главный недостаток данной модели — необхо-

димось четко определять на физическом уровне связи данных и столь же четко следовать этой структуре связей 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 некоторых 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	Детализация	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, формализованы данные, используемые в системе, а также приведен анализ существующих решений.

2 Конструкторская часть

В данном разделе представлены этапы проектирования баз данных.

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, РК
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, РК
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, РК
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, РК
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, РК
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, РК
role	Строка	Роль

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

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

На рисунке 2.1 представлена диаграмма спроектированной базы данных.

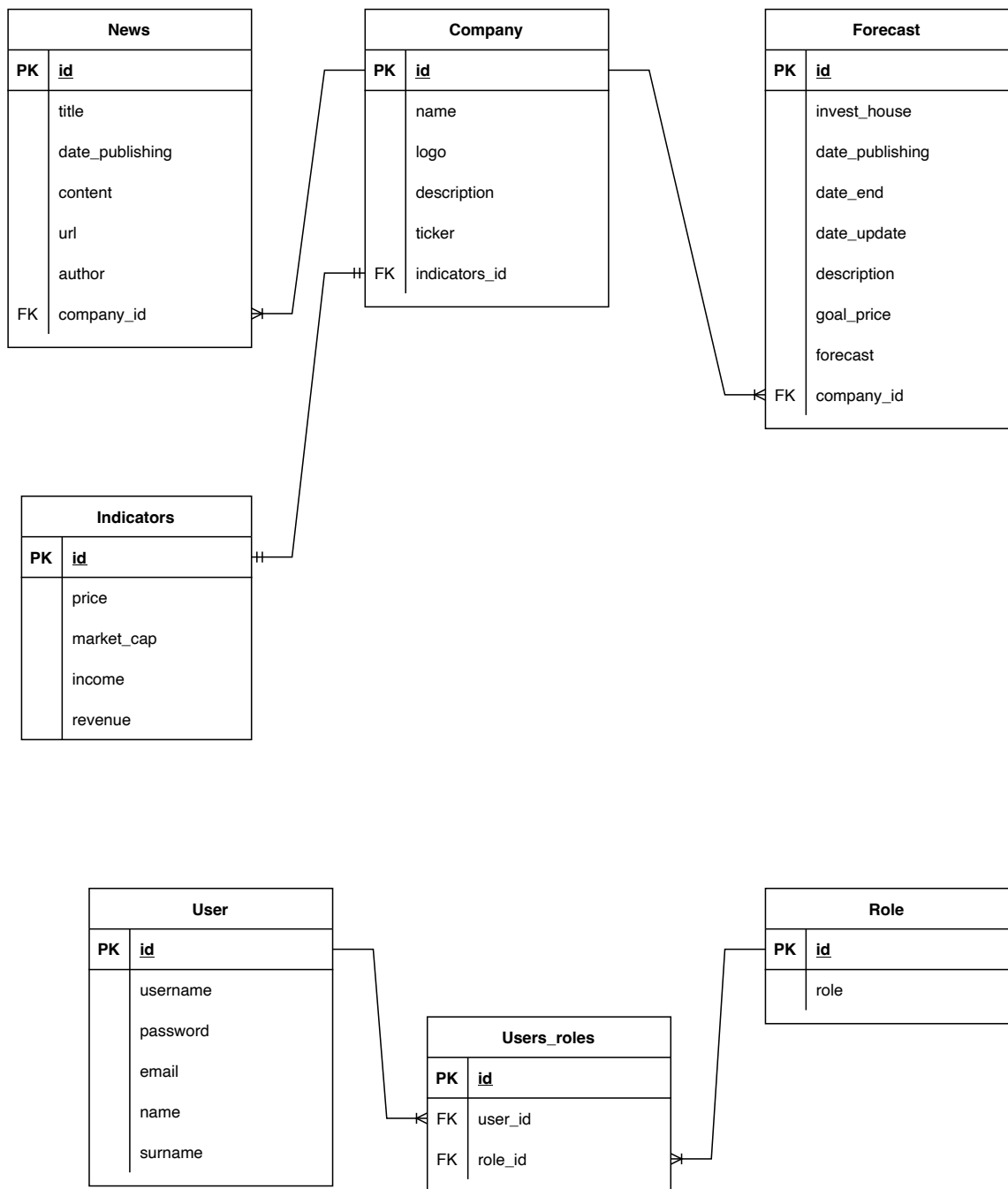


Рис. 2.1: Диаграмма спроектированной базы данных

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 продублированы 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 многокомпонентное 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.

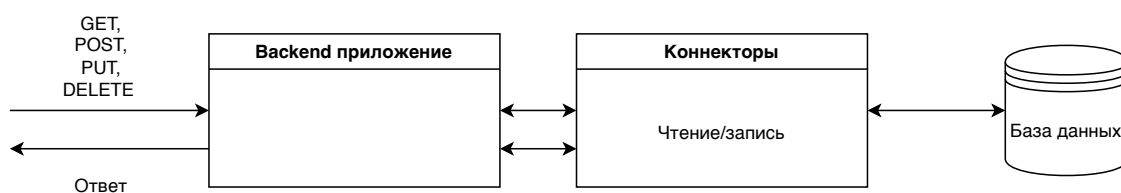


Рис. 2.2: Схема архитектуры спроектированного приложения

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.

3.1 Средства реализации

3.1.1 Выбор СУБД

СУБД — это система управления базами данных. Так называют комплекс программно-языковых средств, позволяющих создавать базы данных и манипулировать данными. 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. PostgreSQL предоставляет транзакции со свойствами согласованности, изоляции, долговечности (ACID [3]).

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

3.1.2 Выбор языка программирования

Для разработки серверной части был выбран язык программирования Java. Данный выбор обусловлен тем, что Java — кроссплатформенный, компилируемый язык программирования, поэтому разработанное 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.

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

3.2 Детали реализации

В листингах 3.1 – 3.6 представлены сущности базы данных.

```
1 package com.example.consensus.entities;
2
3 import com.fasterxml.jackson.annotation.JsonIgnore;
4 import lombok.Data;
5 import javax.persistence.*;
6 import javax.validation.constraints.NotBlank;
7 import javax.validation.constraints.Size;
8 import java.util.List;
9
10 @Table
11 @Entity
12 @Data
13 public class Company {
14     @Id
15     @GeneratedValue(strategy = GenerationType.IDENTITY)
```



```

16     private Long id;
17
18     @Column(name = "name")
19     private String name;
20
21     @Column(name = "logo")
22     private String logo;
23
24     @Column(name = "description")
25     private String description;
26
27     @Column(name = "ticker")
28     @NotBlank
29     @Size(min = 1, max = 6, message = "Ticker length may be between 1 and 6")
30     private String ticker;
31
32     @OneToMany(mappedBy = "companyForForecasts")
33     private List<Forecast> forecasts;
34
35     @OneToMany(mappedBy = "companyForNews")
36     private List<News> news;
37
38     @OneToOne(cascade = CascadeType.ALL)
39     @JoinColumn(name = "indicators_id", referencedColumnName = "id")
40     private Indicators indicators;
41
42     public void setFields(Company company) {
43         setName(company.getName());
44         setLogo(company.getLogo());
45         setDescription(company.getDescription());
46         setTicker(company.getTicker());
47         setForecasts(company.getForecasts());
48         setIndicators(company.getIndicators());
49     }
50 }

```

Листинг 3.1: Сущность Company

```

1 package com.example.consensus.entities;
2
3 import com.fasterxml.jackson.annotation.JsonIgnore;
4 import lombok.Data;
5 import javax.persistence.*;
6 import javax.validation.constraints.NotNull;
7 import java.sql.Timestamp;
8
9 @Table
10 @Entity
11 @Data

```

```

12 public class Forecast {
13     @Id
14     @GeneratedValue(strategy = GenerationType.IDENTITY)
15     @NotNull
16     private long id;
17
18     @Column(name = "invest_house")
19     private String investHouse;
20
21     @Column(name = "date_publishing")
22     private Timestamp datePublishing;
23
24     @Column(name = "date_end")
25     private Timestamp dateEnd;
26
27     @Column(name = "date_update")
28     private Timestamp dateUpdate;
29
30     @Column(name = "goal_price")
31     private int goalPrice;
32
33     @Column(name = "forecast")
34     private String forecast;
35
36     @Column(name = "description")
37     private String description;
38
39     @ManyToOne
40     @JoinColumn(name = "company_id", nullable = false)
41     @JsonIgnore
42     private Company companyForForecasts;
43
44     public void setFields(Forecast forecast) {
45         setInvestHouse(forecast.getInvestHouse());
46         setDatePublishing(forecast.getDatePublishing());
47         setDateEnd(forecast.getDateEnd());
48         setDateUpdate(forecast.getDateUpdate());
49         setGoalPrice(forecast.getGoalPrice());
50         setForecast(forecast.getForecast());
51         setDescription(forecast.getDescription());
52         setCompanyForForecasts(forecast.getCompanyForForecasts());
53     }
54 }

```

Листинг 3.2: Сущность Forecast

```

1 package com.example.consensus.entities;
2
3

```

```

4 import com.fasterxml.jackson.annotation.JsonIgnore;
5 import lombok.Data;
6 import javax.persistence.*;
7 import javax.validation.constraints.Min;
8 import javax.validation.constraints.NotNull;
9 import javax.validation.constraints.Size;
10
11 @Table
12 @Entity
13 @Data
14 public class Indicators {
15     @Id
16     @GeneratedValue(strategy = GenerationType.IDENTITY)
17     @NotNull
18     private long id;
19
20     @Column(name = "price")
21     @Min(value = 0, message = "Price can not be less than 0!")
22     private double price;
23
24     @Column(name = "market_cap")
25     @Min(value = 0, message = "Market cap can not be less than 0!")
26     private int market_cap;
27
28     @Size(min = 0, max = 100, message = "Income must be expressed in %")
29     @Column(name = "income")
30     private int income;
31
32     @Min(value = 0, message = "Revenue must be more than 0!")
33     @Column(name = "revenue")
34     private int revenue;
35
36     @OneToOne(mappedBy = "indicators")
37     @JsonIgnore
38     private Company company;
39
40     public void setFields(Indicators newIndicators) {
41         setPrice(newIndicators.getPrice());
42         setMarket_cap(newIndicators.getMarket_cap());
43         setIncome(newIndicators.getIncome());
44         setRevenue(newIndicators.getRevenue());
45         setCompany(newIndicators.getCompany());
46     }
47 }

```

ЛИСТИНГ 3.3: Сущность Indicators

```

1 package com.example.consensus.entities;
2

```

```

3
4 import com.fasterxml.jackson.annotation.JsonIgnore;
5 import lombok.Data;
6 import javax.persistence.*;
7 import java.sql.Timestamp;
8
9 @Entity
10 @Table(name = "news")
11 @Data
12 public class News {
13     @Id
14     @GeneratedValue(strategy = GenerationType.IDENTITY)
15     private Long id;
16
17     @Column(name = "title")
18     private String title;
19
20     @Column(name = "date_public")
21     private Timestamp datePublic;
22
23     @Column(name = "content")
24     private String content;
25
26     @Column(name = "url")
27     private String url;
28
29     @Column(name = "author")
30     private String author;
31
32     @ManyToOne
33     @JoinColumn(name = "company_id", nullable = false)
34     @JsonIgnore
35     private Company companyForNews;
36
37     public void setFields(News news) {
38         setAuthor(news.getAuthor());
39         setContent(news.getContent());
40         setUrl(news.getUrl());
41         setCompanyForNews(news.getCompanyForNews());
42         setDatePublic(news.getDatePublic());
43         setTitle(news.getTitle());
44     }
45 }

```

Листинг 3.4: Сущность News

```

1 package com.example.consensus.entities;
2
3 import lombok.Data;

```

```

4 import javax.persistence.*;
5 import javax.validation.constraints.Email;
6 import java.util.Collection;
7
8 @Entity
9 @Data
10 @Table(name = "users")
11 public class User {
12
13     @Id
14     @GeneratedValue(strategy = GenerationType.IDENTITY)
15     private Long id;
16
17     @Column(name = "username")
18     private String username;
19
20     @Column(name = "password")
21     private String password;
22
23     @Email
24     @Column(name = "email")
25     private String email;
26
27     @Column(name = "name")
28     private String name;
29
30     @Column(name = "surname")
31     private String surname;
32
33     @ManyToMany
34     @JoinTable(name = "users_roles", joinColumns = @JoinColumn(name = "user_id"
35         ),
36         inverseJoinColumns = @JoinColumn(name = "role_id"))
37     private Collection<Role> roles;
38 }

```

Листинг 3.5: Сущность User

```

1 package com.example.consensus.entities;
2
3 import lombok.Data;
4 import javax.persistence.*;
5
6 @Entity
7 @Table(name = "roles")
8 @Data
9 public class Role {
10
11     @Id
12     @GeneratedValue(strategy = GenerationType.IDENTITY)

```

```

12     private Long id;
13
14     @Column(name = "name")
15     private String name;
16 }

```

Листинг 3.6: Сущность Role

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

```

1 package com.example.consensus.controllers;
2
3 import com.example.consensus.entities.Forecast;
4 import com.example.consensus.services.ForecastService;
5 import org.springframework.web.bind.annotation.*;
6
7 import java.util.List;
8
9 @RestController
10 @RequestMapping("/api/v1")
11 public class ForecastController {
12     private final ForecastService forecastService;
13
14     public ForecastController(ForecastService forecastService) {
15         this.forecastService = forecastService;
16     }
17
18     @GetMapping("/company/{id}/forecasts")
19     public List<Forecast> getAllForecastsByCompanyId(@PathVariable(name="id")
20         Long id) {
21         return forecastService.getAllForecastsByCompanyId(id);
22     }
23
24     @GetMapping("/forecast/{id}")
25     public Forecast getForecastById(@PathVariable(name = "id") Long id) {
26         return forecastService.getForecastById(id);
27     }
28
29     @PutMapping("/forecast/{id}")
30     public Forecast updateForecastById(@PathVariable(name = "id") Long id,
31         @RequestBody Forecast forecastDetails) {
32         return forecastService.updateForecastById(id, forecastDetails);
33     }
34
35     @PostMapping("/forecast/")
36     public Forecast addForecast(@RequestBody Forecast forecast) {
37         return forecastService.addForecast(forecast);
38     }
39 }

```

```

37
38 @DeleteMapping("/forecast/{id}")
39 public Forecast deleteForecast(@PathVariable(name = "id") Long id) {
40     return forecastService.deleteForecast(id);
41 }
42 }

```

Листинг 3.7: Пример реализации взаимодействия клиента с сервером

```

1  package com.example.consensus.controllers;
2
3  import com.example.consensus.entities.Company;
4  import com.example.consensus.services.CompanyService;
5  import org.springframework.http.HttpStatus;
6  import org.springframework.web.bind.annotation.*;
7
8  import javax.validation.Valid;
9  import java.util.List;
10
11 @RestController
12 @RequestMapping("/api/v1")
13 public class CompanyController {
14     private final CompanyService companyService;
15
16     public CompanyController(CompanyService companyService) {
17         this.companyService = companyService;
18     }
19
20     @GetMapping("/all")
21     public List<Company> getAllCompanies() {
22         return companyService.getAllCompanies();
23     }
24
25     @GetMapping("/company/")
26     public Company getCompanyByName(@RequestParam(value = "name") String
name) {
27         return companyService.getCompanyByName(name);
28     }
29
30     @GetMapping("/company/{id}")
31     public Company getCompanyById(@PathVariable Long id) {
32         return companyService.getCompanyById(id);
33     }
34
35     @PutMapping("/company/{id}")
36     public Company updateCompany(@PathVariable Long id, @RequestBody Company
companyDetails) {
37         return companyService.updateCompanyById(id, companyDetails);
38     }

```

```

39
40     @PostMapping("/company/add")
41     public Company addCompany(@Valid @RequestBody Company company) {
42         return companyService.addCompany(company);
43     }
44
45     @ResponseStatus(value = HttpStatus.OK)
46     @DeleteMapping("/company/{id}")
47     public Company deleteCompany(@PathVariable Long id) {
48         return companyService.deleteCompany(id);
49     }
50 }

```

Листинг 3.8: Пример реализации взаимодействия клиента с сервером

В листинге 3.9 представлена инициализация таблиц базы данных.

```

1 CREATE TABLE IF NOT EXISTS Indicators (
2     id serial primary key ,
3     price real,
4     market_cap int,
5     income int,
6     revenue int
7 );
8
9 CREATE TABLE IF NOT EXISTS Company (
10     id serial primary key,
11     name varchar,
12     logo varchar,
13     description varchar,
14     ticker varchar,
15     indicators_id int,
16     FOREIGN KEY (indicators_id) REFERENCES Indicators(id) ON DELETE CASCADE
17 );
18
19 CREATE TABLE IF NOT EXISTS Forecast (
20     id serial primary key ,
21     invest_house varchar,
22     date_publishing timestamp,
23     date_end timestamp,
24     date_update timestamp,
25     goal_price int,
26     forecast varchar,
27     description varchar,
28     company_id int,
29     FOREIGN KEY (company_id) REFERENCES Company(id) ON DELETE CASCADE
30 );
31
32 CREATE TABLE IF NOT EXISTS News (

```



```

33     id serial primary key,
34     title varchar,
35     date_public timestamp,
36     content varchar,
37     url varchar,
38     author varchar
39 );
40
41 CREATE TABLE IF NOT EXISTS Users (
42     id bigserial primary key,
43     username varchar not null ,
44     password varchar,
45     email varchar,
46     name varchar,
47     surname varchar
48 );
49
50 CREATE TABLE IF NOT EXISTS Roles (
51     id bigserial primary key,
52     name varchar not null
53 );

```

Листинг 3.9: Создание таблиц

3.3 Взаимодействие с приложением

На рисунках 3.1 – 3.2 представлены примеры запросов к приложению и ответы.

```

~/repos/sem 06/db-cp main ± curl -X POST -v -H "Content-Type: application/json" -d '{"code": 200, "data": null, "message": "RPD successfully added! RPD id is 10"}' http://0.0.0.0:5000/rpd/10
Note: Unnecessary use of -X or --request, POST is already inferred
* Trying 0.0.0.0...
* TCP_NODELAY set
* Connected to 0.0.0.0 (127.0.0.1) port 5000 (#0)
> POST /rpd/save HTTP/1.1
> Host: 0.0.0.0:5000
> User-Agent: curl/7.54.0
> Accept: */*
> Content-Type: application/json
> Content-Length: 42
>
* upload completely sent off: 42 out of 42 bytes
* HTTP 1.0, assume close after body
< HTTP/1.0 200 OK
< Content-Type: application/json
< Content-Length: 89
< Server: Werkzeug/1.0.1 Python/3.8.10
< Date: Mon, 07 Jun 2021 20:49:53 GMT
<
{
  "code": 200,
  "data": null,
  "message": "RPD successfully added! RPD id is 10"
}
* Closing connection 0

```

Рис. 3.1: Пример сохранения информации о дисциплине посредством http запроса

```

~/repos/sem 06/db-cp main ± curl -X DELETE http://0.0.0.0:5000/rpd/10
{
  "code": 200,
  "data": null,
  "message": "Work program of discipline with id = 10 successfully deleted"
}

```

Рис. 3.2: Пример удаления информации о дисциплине посредством http запроса

```

[ ~/repos/sem 06/db-cp main ± curl -X GET http://0.0.0.0:5000/rpd/10
{
  "code": 200,
  "data": {
    "author": "Исаев А.Л.",
    "competency": "СУОС3++",
    "discipline_material": [
      {
        "discipline_id": 10,
        "id": 89,
        "material": "Информатика. Конспект лекций: учебное пособие / Исаев
7038-4540-0."
      },
      {
        "discipline_id": 10,
        "id": 90,
        "material": "Объектно-ориентированное программирование: учебник для
//ebooks.bmstu.ru/catalog/97/book1033.html. - ISBN 978-5-7038-3921-8."
      },
      {
        "discipline_id": 10,
        "id": 91,
        "material": "Дополнительные учебные материалы"
      },
      {
        "discipline_id": 10,
        "id": 92,
        "material": "Информатика и программирование. Основы информатики: уч
      },
      {
        "discipline_id": 10,
        "id": 93,
        "material": "Информатика и программирование. Алгоритмизация и прог
      },
      {
        "discipline_id": 10,
        "id": 94,
        "material": "Информатика. Конспект практических занятий: учебно-мет
      },
      {
        "discipline_id": 10,
        "id": 95,
        "material": "Введение в визуальное программирование на языке C в ср
      },
      {
        "discipline_id": 10,
        "id": 96,
        "material": "Обработка нечисловых типов данных в среде MS VS C++: у
bmstu.ru/catalog/97/book1618.html - ISBN 978-5-7038-4638-4."
      },
    ]
  }
}

```

Рис. 3.3: Пример получения информации о дисциплине посредством http запроса

Вывод

В данном разделе были представлена архитектура и средства реализации программного обеспечения, листинги ключевых компонентов системы и пример возвращаемых данных системой.

Литература

- [1] Исследование московской биржи. Режим доступа: <https://www.moex.com/n37253> (дата обращения: 07.06.2021).
- [2] 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: 07.06.2021).
- [3] Транзакции, ACID, CAP | GeekBrains [Электронный ресурс]. Режим доступа: https://gb.ru/posts/acid_cap_transactions (дата обращения: 07.06.2021).