Министерство науки и высшего образования Российской Федерации

федеральное государственное автономное   
образовательное учреждение высшего образования   
«Самарский национальный исследовательский университет   
имени академика С.П. Королева»

Институт информатики, математики и электроники

Факультет информатики

Кафедра технической кибернетики

Отчет по лабораторной работе №1

Дисциплина: «Технологии сетевого программирования»

Тема «Проектирование приложения»

Выполнили: Жиляев М.И.

Прохоров С.А

Группа: 6301-010302D

Самара, 2025

**СОДЕРЖАНИЕ**

[1 Настройка базы данных 3](#_Toc192008267)

[1.1 Развернуть PostgeSQL в Docker 3](#_Toc192008268)

[1.2 Подключиться к БД через DBeaver 3](#_Toc192008269)

[2 Разработка ORM-моделей 4](#_Toc192008270)

[2.1 Определить сущности и связи между ними 4](#_Toc192008271)

[2.2 Реализовать модели с использованием SQLAlchemy/Django ORM; 4](#_Toc192008272)

[2.3 Настроить миграции 6](#_Toc192008273)

[3 Разработка модели пользователя 7](#_Toc192008274)

[4 Работа с моделями 8](#_Toc192008275)

[4.1 Написаны скрипты для заполнения БД 8](#_Toc192008276)

1. Настройка базы данных
   1. Развернуть PostgeSQL в Docker

Был написан докер композ, где описывалась бд

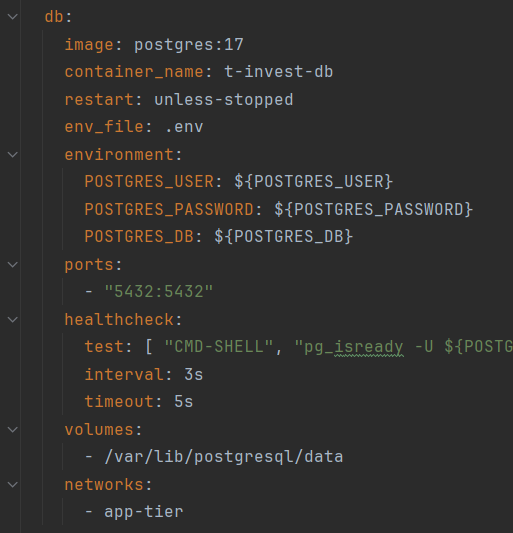


Рисунок 1 – Docker-compose

* 1. Подключиться к БД через Pg Admin 4

После создания образа и контейнера было выполнено подключение к базе данных через программу Pg Admin 4.

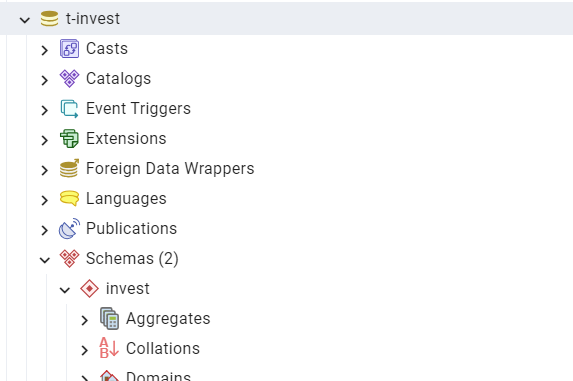


Рисунок 2 – Подключение к базе данных

1. Разработка ORM-моделей
   1. Определить сущности и связи между ними

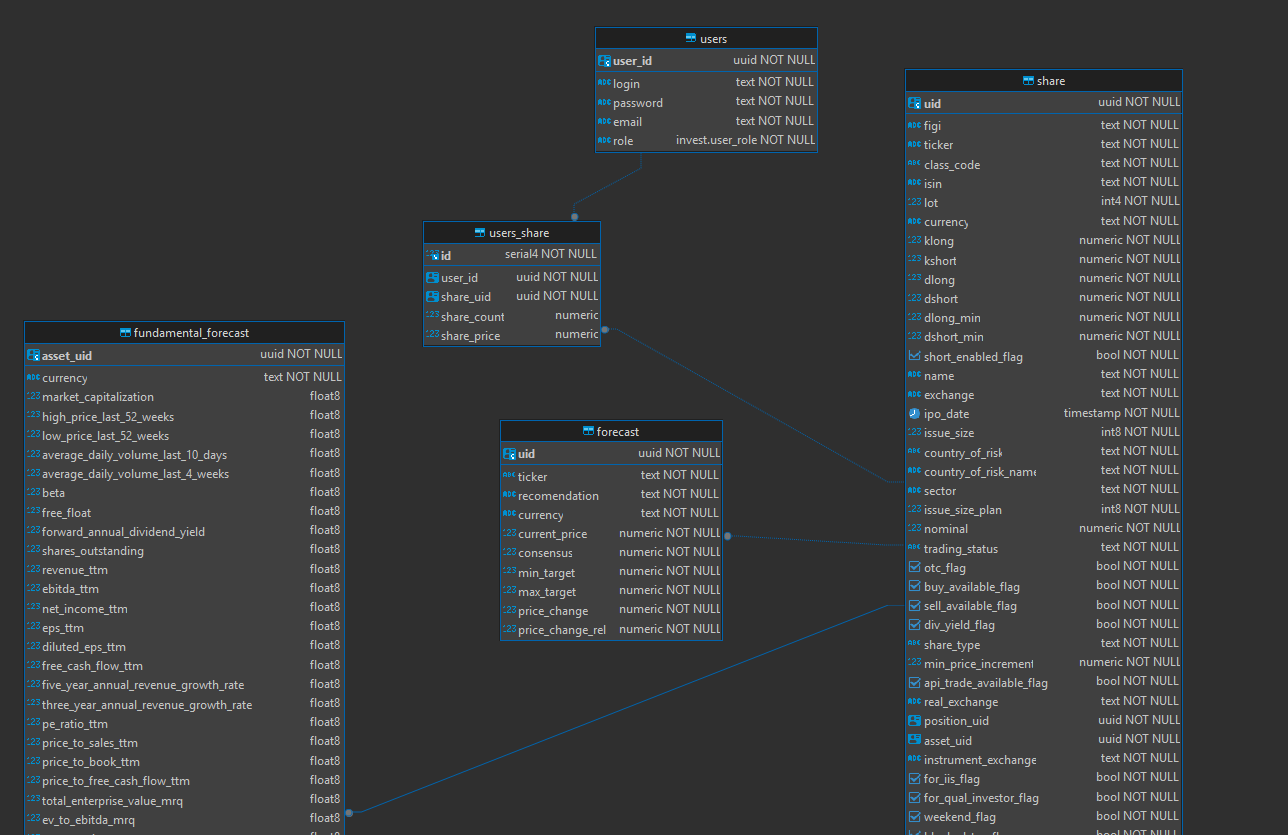


Рисунок 3 – Сущности и связи между ними

* 1. Реализовать модели с использованием Jooq ORM;

Для реализации моделей был выбран Jooq ORM. Модели сгенерированы при помощи JOOQ по sql миграциям. Код генератора Jooq:

jooq {

configuration {

generator {

database {

name = "org.jooq.meta.extensions.liquibase.LiquibaseDatabase"

properties {

property {

key = "rootPath"

value = "${project.projectDir}/src/main/resources"

}

// Specify the classpath location of your XML, YAML, or JSON script.

property {

key = "scripts"

value = "/migrations/changelog-master.xml"

}

// Whether you want to include liquibase tables in generated output

//

// - false (default)

// - true: includes DATABASECHANGELOG and DATABASECHANGELOGLOCK tables

property {

key = "includeLiquibaseTables"

value = false.toString()

}

// Whether you want to use jOOQ's translating ParsingConnection to translate

// between your dialect (e.g. Oracle), and jOOQ's in-memory H2 dialect

//

// - false (default)

// - true: translates e.g. from VARCHAR2(100) to VARCHAR(100)

property {

key = "useParsingConnection"

value = false.toString()

}

// Properties prefixed "database." will be passed on to the liquibase.database.Database class

// if a matching setter is found

property {

key = "database.liquibaseSchemaName"

value = "lb"

}

// The property "changeLogParameters.contexts" will be passed on to the

// liquibase.database.Database.update() call (jOOQ 3.13.2+).

// See https://www.liquibase.org/documentation/contexts.html

property {

key = "changeLogParameters.contexts"

value = "!test"

}

}

target {

directory = "${project.projectDir}/src/main/java/dev/invest/db/jooq"

packageName = "dev.invest.db.jooq.org.jooq.generated"

}

}

}

}

}

* 1. Настроить миграции

Миграции были написаны на sql при помощи Liquibase.

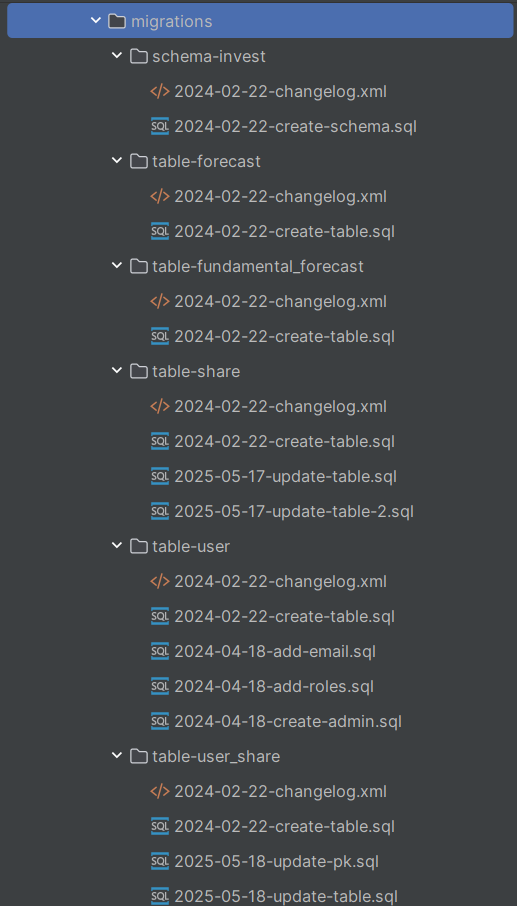


Рисунок 4 – Созданы миграции

1. Разработка модели пользователя

Данное приложение не требует много информации о пользователе, необходимы лишь адрес электронной почты, выступающий в роли логина, пароль и имя пользователя, было решено остановиться на выборе в качестве модели пользователя стандартной модели из библиотеки Django.

Код миграциий для пользователя:

CREATE TABLE IF NOT EXISTS invest.users (

user\_id UUID NOT NULL,

login text NOT NULL UNIQUE,

password text NOT NULL,

CONSTRAINT users\_pk PRIMARY KEY (user\_id)

);

COMMENT ON TABLE invest.users IS 'Таблица для хранения пользователей';

COMMENT ON COLUMN invest.users.user\_id IS 'Уникальный идентификатор пользователя';

COMMENT ON COLUMN invest.users.login IS 'Логин пользователя';

COMMENT ON COLUMN invest.users.password IS 'Пароль пользователя';

ALTER TABLE invest.users ADD COLUMN email text NOT NULL UNIQUE;

create type invest.user\_role as ENUM ('ROLE\_USER', 'ROLE\_ADMIN');

ALTER TABLE invest.users ADD COLUMN role invest.user\_role NOT NULL DEFAULT 'ROLE\_USER';

insert into invest.users (user\_id, login, password, email, role)

values (

'550e8400-e29b-41d4-a716-446655440000',

'admin',

'$2a$10$I3Eic6lzladxLnEq1.jhj.3978TlnkNQNlbzeR3hNTTRNQ6lM9bn.',

'admin@example.com',

'ROLE\_ADMIN'

)

Далее было настроено хэширование паролей:

@Bean

public PasswordEncoder passwordEncoder() {

return new BCryptPasswordEncoder();

}

1. Работа с моделями
   1. Написаны скрипты для заполнения БД

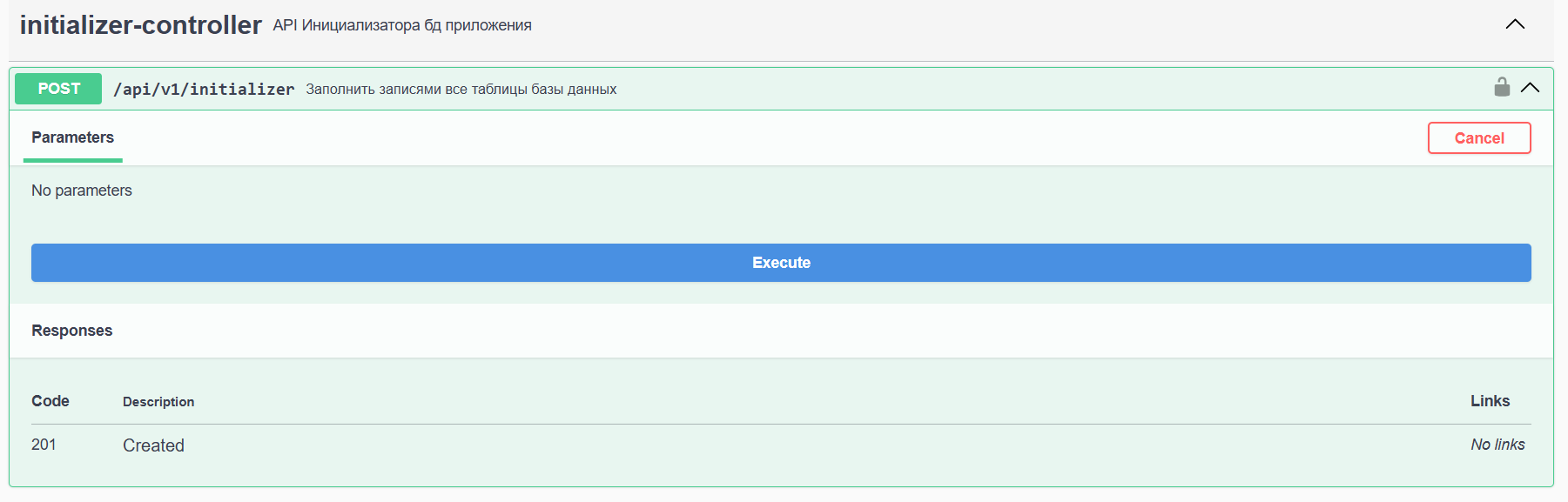
Для заполения бд данными был создан REST контроллер с ручкой, которая запускала процесс генерации данных в бд.  


Рисунок 5 – Эндпоинт для генерации данных в бд

Код генератора:

package dev.invest.utils;

import dev.invest.db.jooq.org.jooq.generated.invest.tables.records.UsersShareRecord;

import dev.invest.model.user.CreateUserRequest;

import java.math.BigInteger;

import java.util.List;

import java.util.Random;

import java.util.UUID;

import lombok.experimental.UtilityClass;

import org.apache.commons.lang3.RandomStringUtils;

import ru.tinkoff.piapi.contract.v1.GetAssetFundamentalsResponse;

import ru.tinkoff.piapi.contract.v1.GetForecastResponse;

import ru.tinkoff.piapi.contract.v1.Quotation;

import ru.tinkoff.piapi.contract.v1.Recommendation;

@UtilityClass

public class GenerateUtils {

private static final Random random = new Random();

public static Quotation buildRandomQuotation() {

return Quotation.newBuilder()

.setUnits(random.nextInt(1, 10000))

.setNano(random.nextInt(1, 10000))

.build();

}

public static GetForecastResponse.ConsensusItem buildRandomConsensusItem(List<String> tickers) {

return GetForecastResponse.ConsensusItem.newBuilder()

.setUid(UUID.randomUUID().toString())

.setTicker(tickers.get(random.nextInt(1, 10000) % tickers.size()))

.setRecommendation(Recommendation.RECOMMENDATION\_BUY)

.setCurrency("RUB")

.setCurrentPrice(buildRandomQuotation())

.setConsensus(buildRandomQuotation())

.setMinTarget(buildRandomQuotation())

.setMaxTarget(buildRandomQuotation())

.setPriceChange(buildRandomQuotation())

.setPriceChangeRel(buildRandomQuotation())

.build();

}

public static List<GetForecastResponse.ConsensusItem> buildConsensusList(List<String> tickers) {

return List.of(

buildRandomConsensusItem(tickers),

buildRandomConsensusItem(tickers),

buildRandomConsensusItem(tickers)

);

}

public static GetAssetFundamentalsResponse.StatisticResponse buildRandomFundamental(List<UUID> assetUuids) {

return GetAssetFundamentalsResponse.StatisticResponse.newBuilder()

.setAssetUid(assetUuids.get(random.nextInt(0, assetUuids.size())).toString())

.setCurrency("USD")

.setMarketCapitalization(random.nextDouble(1, 10000))

.setHighPriceLast52Weeks(random.nextDouble(1, 10000))

.setLowPriceLast52Weeks(random.nextDouble(1, 10000))

.setAverageDailyVolumeLast10Days(random.nextDouble(1, 10000))

.setAverageDailyVolumeLast4Weeks(random.nextDouble(1, 10000))

.setBeta(random.nextDouble(1, 10000))

.setFreeFloat(random.nextDouble(1, 10000))

.setForwardAnnualDividendYield(random.nextDouble(1, 10000))

.setSharesOutstanding(random.nextDouble(1, 10000))

.setRevenueTtm(random.nextDouble(1, 10000))

.setEbitdaTtm(random.nextDouble(1, 10000))

.setNetIncomeTtm(random.nextDouble(1, 10000))

.setEpsTtm(random.nextDouble(1, 10000))

.setDilutedEpsTtm(random.nextDouble(1, 10000))

.setFreeCashFlowTtm(random.nextDouble(1, 10000))

.setFiveYearAnnualRevenueGrowthRate(random.nextDouble(1, 10000))

.setThreeYearAnnualRevenueGrowthRate(random.nextDouble(1, 10000))

.setPeRatioTtm(random.nextDouble(1, 10000))

.setPriceToSalesTtm(random.nextDouble(1, 10000))

.setPriceToBookTtm(random.nextDouble(1, 10000))

.setPriceToFreeCashFlowTtm(random.nextDouble(1, 10000))

.setTotalEnterpriseValueMrq(random.nextDouble(1, 10000))

.setEvToEbitdaMrq(random.nextDouble(1, 10000))

.setNetMarginMrq(random.nextDouble(1, 10000))

.setNetInterestMarginMrq(random.nextDouble(1, 10000))

.setRoe(random.nextDouble(1, 10000))

.setRoa(random.nextDouble(1, 10000))

.setRoic(random.nextDouble(1, 10000))

.setTotalDebtMrq(random.nextDouble(1, 10000))

.setTotalDebtToEquityMrq(random.nextDouble(1, 10000))

.setTotalDebtToEbitdaMrq(random.nextDouble(1, 10000))

.setFreeCashFlowToPrice(random.nextDouble(1, 10000))

.setNetDebtToEbitda(random.nextDouble(1, 10000))

.setCurrentRatioMrq(random.nextDouble(1, 10000))

.setFixedChargeCoverageRatioFy(random.nextDouble(1, 10000))

.setDividendYieldDailyTtm(random.nextDouble(1, 10000))

.setDividendRateTtm(random.nextDouble(1, 10000))

.setDividendsPerShare(random.nextDouble(1, 10000))

.setFiveYearsAverageDividendYield(random.nextDouble(1, 10000))

.setFiveYearAnnualDividendGrowthRate(random.nextDouble(1, 10000))

.setDividendPayoutRatioFy(random.nextDouble(1, 10000))

.setBuyBackTtm(random.nextDouble(1, 10000))

.setOneYearAnnualRevenueGrowthRate(random.nextDouble(1, 10000))

.setDomicileIndicatorCode("US")

.setAdrToCommonShareRatio(random.nextDouble(1, 10000))

.setNumberOfEmployees(random.nextDouble(1, 10000))

.setExDividendDate(DateUtils.getCurrentProtoTimestamp())

.setFiscalPeriodStartDate(DateUtils.getCurrentProtoTimestamp())

.setFiscalPeriodEndDate(DateUtils.getCurrentProtoTimestamp())

.setRevenueChangeFiveYears(random.nextDouble(1, 10000))

.setEpsChangeFiveYears(random.nextDouble(1, 10000))

.setEbitdaChangeFiveYears(random.nextDouble(1, 10000))

.setTotalDebtChangeFiveYears(random.nextDouble(1, 10000))

.setEvToSales(random.nextDouble(1, 10000))

.build();

}

public static List<GetAssetFundamentalsResponse.StatisticResponse> buildFundamentalList(List<UUID> assetUuids) {

return List.of(

buildRandomFundamental(assetUuids),

buildRandomFundamental(assetUuids),

buildRandomFundamental(assetUuids)

);

}

public static CreateUserRequest buildCreateUserRequest() {

return new CreateUserRequest(

RandomStringUtils.randomAlphabetic(20) + "@test.com",

RandomStringUtils.randomAlphabetic(20),

RandomStringUtils.randomAlphabetic(20));

}

public static List<CreateUserRequest> buildCreateUserRequestList() {

return List.of(

buildCreateUserRequest(),

buildCreateUserRequest(),

buildCreateUserRequest()

);

}

public static UsersShareRecord buildUserShare(UUID userId, UUID shareUid) {

return new UsersShareRecord(userId,

shareUid,

BigInteger.valueOf(random.nextLong(1, 1000)),

BigInteger.valueOf(random.nextLong(1, 10000)),

random.nextInt(1, 2000000)

);

}

public static List<UsersShareRecord> buildUserShareList(List<UUID> userIds, List<UUID> shareIds) {

return List.of(

buildUserShare(userIds.get(0), shareIds.get(0)),

buildUserShare(userIds.get(1), shareIds.get(1)),

buildUserShare(userIds.get(2), shareIds.get(2))

);

}

}