Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования «Удмуртский государственный университет»

Институт математики информационных технологий и физики

Кафедра теоретических основ информатики

Курсовая работа на тему

“Проектирование и разработка базы данных и приложения автоматизированной системы тестирования программ”

По дисциплине “Базы данных”

Выполнил: студент   
группы 02.03.02-21

Новожилов Константин Алексеевич

Научный руководитель:

старший преподаватель

Анисимов Андрей Евгеньевич

Ижевск, 2019

# Аннотация

Курсовая работа посвящена проектированию и разработке базы данных и разработке приложения автоматизированной системы тестирования программ.

Работа включает в себя изучение предметной области, инструментальных средств и обзор аналогичных систем и проектирование базы данных.

Приложение написано на языке Java с использованием фрэймворка Spring MVC, также использовалась СУБД MySQL.

# Содержание

Аннотация 1

Содержание 2

Введение 2

1. Постановка задачи 3

1.1 Общие требования 3

1.2 Техническое задание 3

2. Обзор инструментальных средств 4

2.1 Java 4

2.2 MySQL 4

3. Архитектура приложения 5

3.1 Анализ аналогичных приложений 5

3.2 Разработка архитектуры приложения 5

4. Проектирование базы данных 7

4.1 Концептуальная модель предметной области 7

4.2 Даталогическая модель базы данных 8

4.3 Физическая модель базы данных 9

5. Разработка приложения 12

5.1 Классы и интерфейсы 12

5.2 Графический интерфейс 15

5.3 Тестирование приложения 17

Заключение 18

Список Литературы 19

# Введение

Информатизация и автоматизация учебного процесса требует создания нового программного обеспечения предназначеного для этой задачи. Проверка решений задач студентов создает дополнительную нагрузку на преподавателей.

В настоящее время достаточно мало доступных систем, которые предоставляют возможность автоматизированног тестирования компьютерных программ, поддержки состава задач, путем их добавления и быстрой проверки большого количества задач.

Поэтому разработанная система должна предоставлять возможность студентам быстро сдавать решенные задачи, а также облегчить преподавателям их проверку.

# 1. Постановка задачи

## 1.1 Общие требования

Требуется создать приложение для тестирования программ написанных студентами на языке Java с использованием стандартных потоков ввода и вывода System.In и System.Out и состоящих из одного класса. Приложение должно реализовывать добавление программ студентами и их проверку преподавателем.

## 1.2 Техническое задание

Приложение “Автоматизированная система тестирования программ” должно удовлетворять следующим техническим требованиям:

1. Функционал авторизации и регистрации пользователей.
2. Возможность добавления программ обучающимися.
3. Автоматизированное тестирование программ с использованием автономной компиляции и анализа входных и выходных данных.
4. Возможность проверки программ преподавателем.
5. Иметь интуитивно понятный интерфейс

# 2. Обзор инструментальных средств

## 2.1 Java

Java - объектно-ориентированный язык программирования, разрабатываемый компанией Oracle. Язык является кросс-платформенным, при условии наличия JVM (Виртуальной машины Джава).

Существует большое количество Java-фреймворков для разработки Web-приложений: Spring MVC, PrimeFaces, Blade, Google web Toolkit и другие. Наиболее популярным является фреймворк Spring MVC.

Spring реализует схему разделения данных MVC (Model-View-Controller) — использующую такие паттерны проектирования как: “Наблюдатель”, “Стратегия”, “Компоновщик”, “Команда” и т.д., а также разделяет программу на три отдельных компонента:

* + - Модель предоставляющую данные и методы работы с этими данными.
    - Представление отвечающую за отображение информации.
    - Контроллер обеспечивающую связь между пользователем и системой.

## 2.2 MySQL

MySQL - реляционная система управления базами данных. Данные хранятся в виде связанных между собой таблиц. Доступ осуществляется с помощью SQL запросов.

Основные преимущества:

Быстродействие (использование многопоточности, система распределения памяти)

Открытость для некоммерческого использования

Безопасность (Управление правами пользователей)

Доступность API для Java

# 3. Архитектура приложения

## 3.1 Анализ аналогичных приложений

Stepik: Платформа онлайн-курсов предоставляющая уже готовые сборники задач и тестов, имеет встроенную IDE с возможностью проверки прямо на сайте. Не предусмотрено: добавление задач в уже существующий курс, одновременная проверка большого количества программ, запускаемая пользователем, назначение задач пользователям.

ACMP: Сборник задач с возможностью их проверки. Решения должны быть в виде одного файла исходного кода языка С++. Программы должны использовать определенные файловые потоки. Не предусмотрено: добавление задач в уже существующий курс, одновременная проверка большого количества программ, запускаемая пользователем, назначение задач пользователям.

Расмотренные системы предоставляют собой удобный инструмент для решения и проверки задач для индивидуальных пользователей, с возможностью тестирования программ.

Однако можно заметить, что в данных примерах аналогичных приложений не предусмотрено: назначение задач пользователям, проверка большого количества программ, запускаемая пользователем. Так что данные системы не подходят для использования в учебном процессе.

## 3.2 Разработка архитектуры приложения

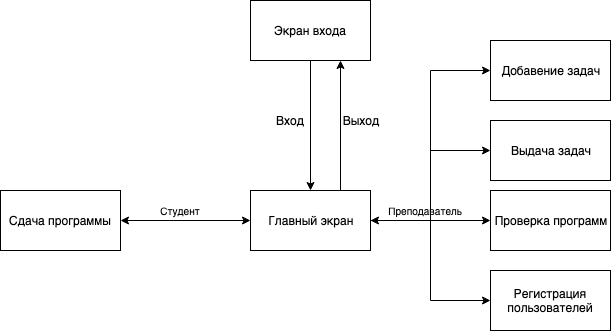
Приложение АСТ позволяет:

1. Регистрация новых пользователей;
2. Возможность сдачи программ пользователями;
3. Возможность проверки программ преподавателем;

Экран входа в приложение представляет собой два поля для ввода ID пользователя и его пароля и кнопку входа. Главный экран приложения содержит кнопки для сдачи и выхода, а также кнопку для проверки работ если пользователь принадлежит группе “admin”.

Схема интеракции пользователя с интерфейсом представлена на рис. 1

Рис. 1 Схема интеракции пользователя с интерфейсм



# 4. Проектирование базы данных

## 4.1 Концептуальная модель предметной области

При построении информационно-логической модели было выделено 5 основных сущностей:

* + 1. Пользователь (имя, электронная почта, пароль, режим);
    2. Задача (Текст)
    3. Тест (Ввод, ожидаемый результат)
    4. Программа (Путь)
    5. Результат работы (Количество пройденных тестов)

Сущность “пользователь” определяет необходимую информацию представляющую объект “пользователь”, то есть студент или преподаватель, и имеет поля имя, пароль, электронная почта и группа пользователей.

Сущность “задача” определяет необходимую информацию представляющую объект “задача” и имеет единственное поле — текст задачи.

Сущность “тест” определяет необходимую информацию представляющую объект “тест” и имеет поля ввод и ожидаемый результат.

Сущность “программа” определяет необходимую информацию представляющую объект “программа” и имеет единственное поле путь к программе.

Сущность “результат работы программы” определяет необходимую информацию представляющую объект “результат работы программы” и имеет единственное поле количество пройденных тестов.

Сама информационно-логическая модель представлена на рис. 2

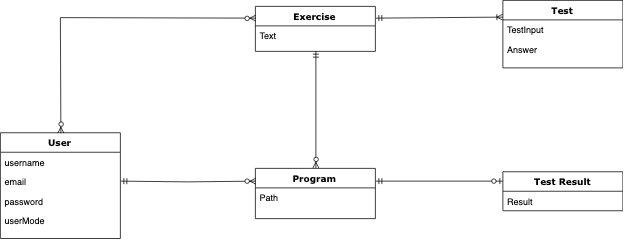


Рис. 2 информационно-логическая модель

## 4.2 Даталогическая модель базы данных

Построение даталогической модели базы данных сводится к реализации логических связей между сущностями.

Отношение многие-ко-многим между сущностями “пользователь” и “задача”; один-ко-многим между сущностями “задача” и “программа”; один-ко-многим между сущностями “пользователь” и “программа” привели к образованию таблицы связи. Остальные связи реализованы с помощью миграции первичного ключа.

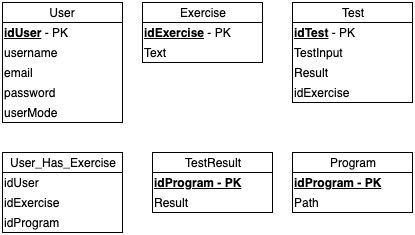
Даталогическая модель представлена на рис. 3

Рис. 3 даталогическая модель

## 4.3 Физическая модель базы данных

Физическая модель базы данных требует приведение полей сущностей и таблиц связей к определенным типам, существующих в СУБД.

Поля сущности “User” имеют следующие типы:

* 1. idUser - INT
  2. Username - VARCHAR(255)
  3. Email - VARCHAR(255)
  4. Password - VARCHAR(32)
  5. userMode - VARCHAR(32)

Поля сущности “Exercise”:

* 1. idExercise - INT
  2. Text - VARCHAR(255)

Поля сущности “Test”:

* 1. idTest - INT
  2. TestInput - VARCHAR(255)
  3. Answer - VARCHAR(255)
  4. Exercise\_idExercise - INT

Поля сущности “Program”:

* 1. idProgram - INT
  2. Path - VARCHAR(255)

Поля сущности “TestResult”:

* 1. Result - VARCHAR(255)
  2. Program\_idProgram

Поля сущности “User\_Has\_Exercise”:

* 1. User\_idUser - INT
  2. Exercise\_idExercise - INT
  3. idProgram - INT

Физическая модель базы данных, разработанная с помощью программы MySQL Workbench, представлена на рис. 4

Для создания базы данных используется скрипт сгенерированный с помощью MySQL Workbench. Разберем создание одной из таблиц — таблицы “User”, остальной текст запроса находится в приложении 1.

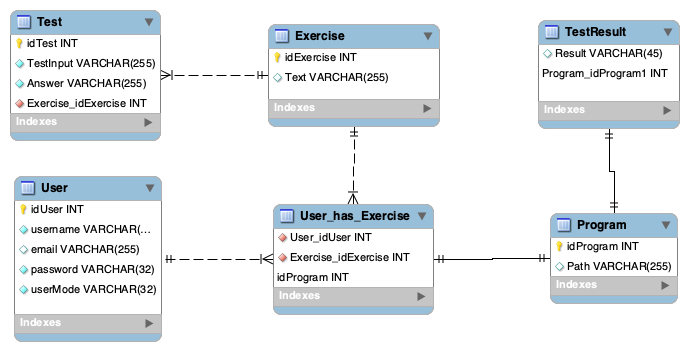


Рис. 4 физическая модель

CREATE TABLE IF NOT EXISTS `mydb`.`User` (

`idUser` INT NOT NULL,

`username` VARCHAR(45) NOT NULL,

`email` VARCHAR(255) NULL,

`password` VARCHAR(32) NOT NULL,

`userMode` VARCHAR(32) NOT NULL,

INDEX `username` (`username` ASC),

PRIMARY KEY (`idUser`),

UNIQUE INDEX `UserId\_UNIQUE` (`idUser` ASC));

Таким образом создается таблица “User” с атрибутами “idUser”, “username”, “email”, “password”, “userMode”, в которой первичным ключом является атрибут idUser, он же и является индексом для сортировки при необходимости, также индексом является атрибут “username”.

В создаваемой программе для обращения к базе данных используются выражения: SELECT, UPDATE, INSERT, DELETE. Эти выражения реализованы в соответсвующих классах, которые служат для обращения к базе данных.

# 5. Разработка приложения

## 5.1 Классы и интерфейсы

Приложение АСТ состоит из следующих групп классов и интерфейсов:

* + - DТO (Data Transfer Object) — классы модели, представляющие сущности базы данных.
    - DAO (Data Access Object) — классы модели, предоставляющие доступ и реализующие необходимые запросы к базе данных и реализующие паттерн “Репозиторий”.
    - Services — классы модели, интерфейс взаимодействия между моделью и контроллером.
    - Controllers — классы контроллера, получающие данные из представления и интерпретирующие их.
    - Interceptors — классы отвечающие за управление доступом пользователя.
    - Config — класс отвечающий за предоставление необходимых зависимостей другим классам.
    - JSP — классы, представляющие собой сервлеты Java, ответственные за представление приложения и графический интерфейс.

Исходный код классов DAO находится в приожении 2.

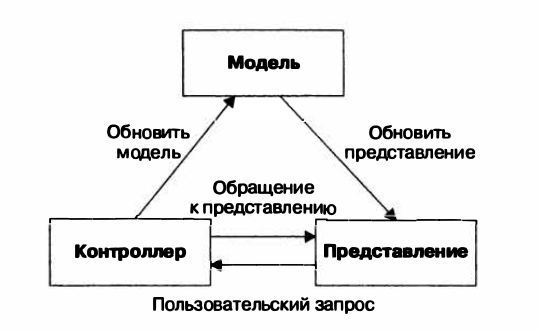


Рис. 5 представление модели MVC

Структура программы следует модели MVC(Model-View-Controller, Модель-Представление-Контроллер), представленной на рис. 5.

Классы ответственные за обращение к базе данных принадлежат группе DAO. К каждой сущности базы данных обращается определенный класс, реализующий соответсвующий ему интерфейс. Использование интерфейсов позволяет в дальнейшем упростить поддержку и расширяемость программы, например при миграции на другую СУБД.

В качестве примера разберем класс TestRepository и соответсвующий ему интерфейс ITestRepository:

Интерфейс ITestRepository предпологает реализацию методов: create, getTest, getTestsByExerciseId, deleteTest - в имплементирующем его классе, где методы предпологают следующее:

create - создание новой записи в базе данных;

getTest - получение единственного экземпляра сущности Test;

getTestsByExerciseId - получение списка тестов по номеру задачи;

deleteTest - удаление записи из базы данных;

Класс TestRepository, реализующий интерфейс ITestRepository, обеспечивает связь с базой данных, использующей СУБД MySQL.

Методы класса:

* + - * create - метод для добавления записи в базу данных с помощью SQL запроса: "insert into `mydb`.`exercise` (`Exercise\_idExercise`, `testInput`, `answer`) VALUES(idExercise, data, expectedResult)”
      * getTest - метод для получения записи по первичному ключу с помощью SQL запроса: "select \* from test where idTest = ?”
      * getTestsByExerciseId - метод для получения списка тестов с помощью SQL запрос: “select \* from test where Exercise\_idExercise = ?”
      * delete - метод для удаления записи с помощью SQL запроса: "delete test from test where idTest = ?"

Как видно из приведенного выше кода, класс содержит необходимые запросы для обращения к базе данных, которые он направляет в синглтон JDBCTemplate, содержащий драйвер для обращения к базе данных на СУБД MySQL. Остальные классы и интерфейсы DAO реализованы подобным образом .

## 5.2 Графический интерфейс

Графический интерфейс программы представляет собой интернет-сайт, который состоит из страниц: логина, регистрации, сдачи задания, проверки заданий, главной страницы. Каждая страница генерируется с помощью технологии JSP(JavaServer Pages). Каждая страница имеет статические компоненты, оформленные в формате HTML, и динамические компоненты, в формате JSP-Элементов. Код страницы транслируется в код сервлета с помощью компилятора Jasper.

Страница логина состоит из 2 полей ввода (ID и пароль) и кнопки входа.

Главная страница представляет собой простой список ссылок на страницу сдачи задания, а также кнопку выхода. Если пользователь принадлежит группе “admin” страница также содержит ссылки на страницы проверки заданий, назначения задания, регистрации пользователя.

Страница проверки заданий содержит полее ввода(Номер задачи), кнопку вывода и поле вывода, куда выводятся результаты тестов программ.

Остальные страницы подобны между собой и содержат одно или несколько полей ввода и кнопку совершения действия.

Общий вид страниц можно увидеть на рис. 5-8.

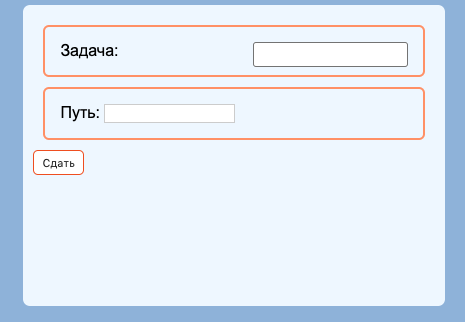


Рис. 5 страница сдачи задачи

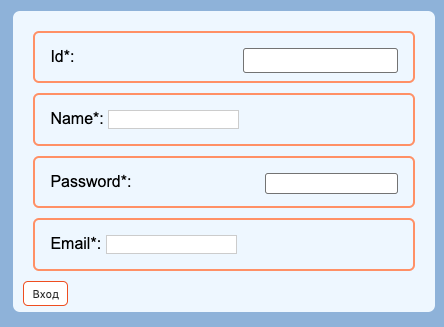


Рис. 6 страница регистрации пользователя

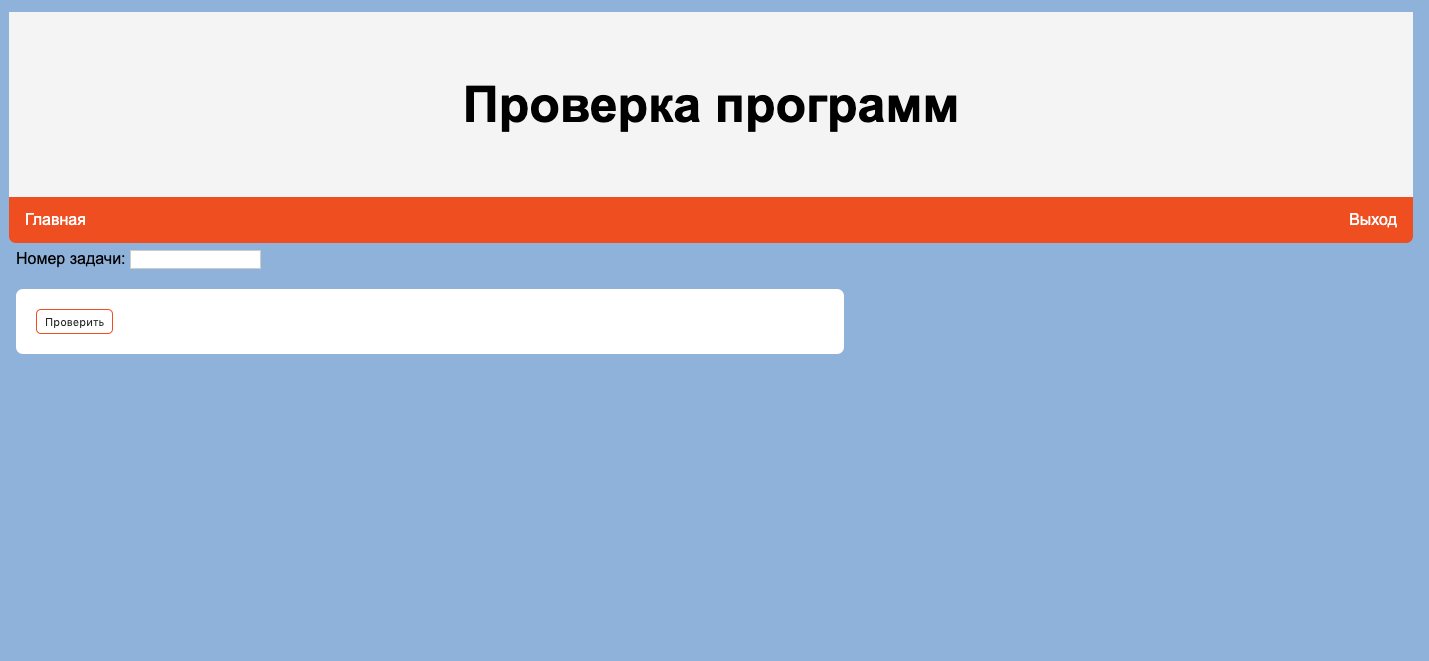


Рис. 7 Проверка программ преподавателем

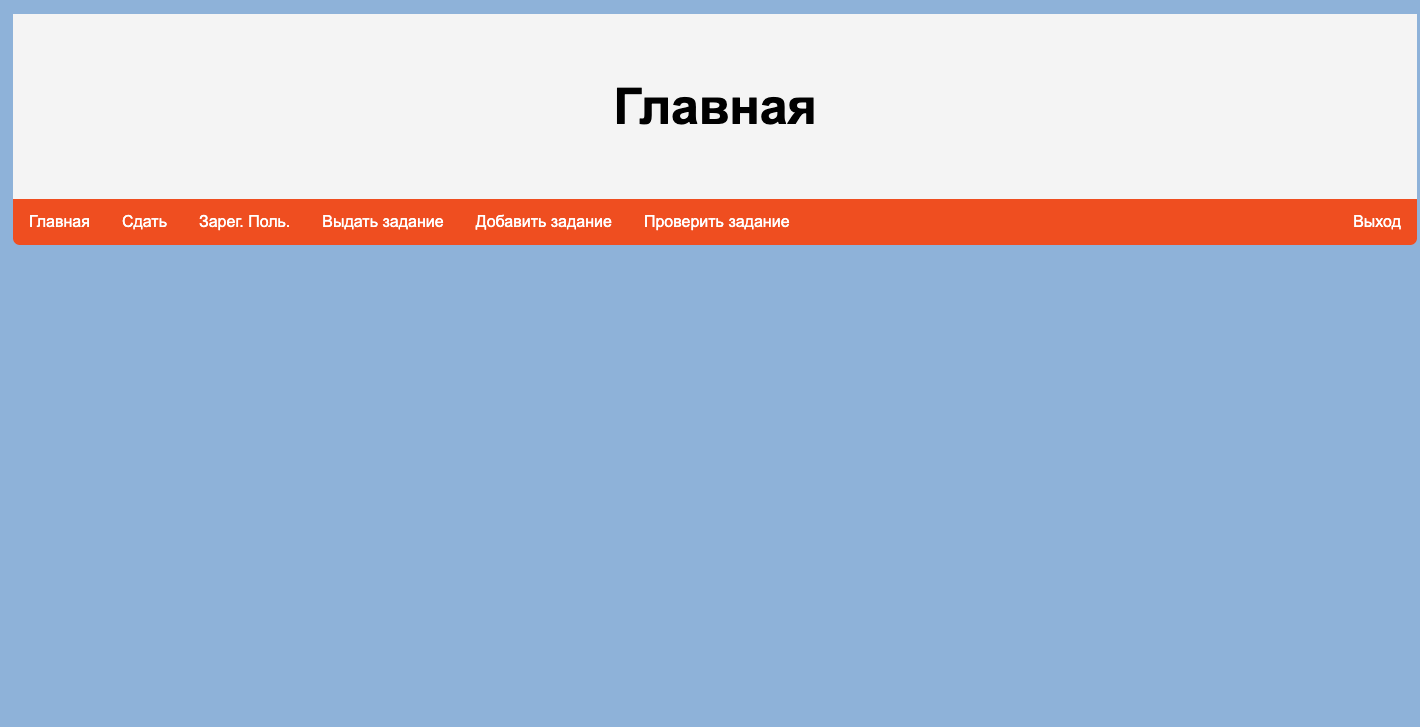


Рис. 8 Главная страница если пользователь - преподаватель

## 5.3 Тестирование приложения

Для проверки работы приложения было проведено:

1. Проверка работоспособности базы данных:
   1. Запрос списка пользователей select \* from ‘mydb’.’user’
   2. Запрос списка задач select \* from ‘mydb’.’exercise’
   3. Запрос списка назначенных задач select idProgram from ‘mydb’.’user\_has\_exercise’
   4. Проверка работы программы путем использования интерфейса:
      1. Добавление нового пользователя
      2. Добавление новой задачи
      3. Назначение задачи пользователю
      4. Сдача программы
      5. Проверка задачи

Перед проведением тестов в таблицу были добавлены тестовые данные.

В ходе выполнения SQL-запросов возвращаемые данные соответствовали ожидаемым.

В ходе использования интерфейса программы ход работы программы не прерывался, ошибок выявлено не было.

Тестовые данные и результаты тестирования представлены в приложении 3

# Заключение

В данной курсовой работе было разработано приложение “Автоматизированная система тестирования программ” на языке Java с помощью фреймворка Spring MVC, а также спроектирована база данных этого приложения на языке SQL с использованием СУБД MySQL. Было изучено:

* Модули фреймворка Spring MVC.
* Возможности языка запросов SQL
* Возможности СУБД MySQL и клиент MySQL Workbench
* Паттерны проектирования: “Фабрика”, “Инъекция зависимостей”, “Репозиторий”.

Также возможно дальнейшее развитие проекта: поддержка многофайловых проектов, поддержка иных потоков ввода/вывода, использование многопоточности, генерация тестов с помощью программы-примера.

# Список Литературы

1. Уоллс К. Spring in Action / К. Уоллс , 2018 — М.: Издательство Manning Publications — 498 стр.
2. Диго С. М. Базы данных, проектирование и создание / С. М. Диго — М.: Издательский центр ЕАОИ, 2008. — 173 стр.
3. Oracle [Электронный ресурс]: The Java™ Tutorials / Oracle Corporation — M.: Oracle Corporation, 2019 — Режим доступа: <https://docs.oracle.com/javase/tutorial/>
4. Шварц Б. High Performance MySQL: Optimization, Backups , and Replication / Б. Шварц, В. Зайцев, В. Ткаченко — М.: O’Reilly 2012. — 793 стр.
5. Ульман Д. Системы баз данных: Полный курс / Д. Ульман, Г. Гарсиа-Молина, Д. Уидом — М.: Издательский дом “Вильямс”, 2003 — 1088 стр.

Приложение 1.

-- MySQL Script generated by MySQL Workbench

-- Fri May 17 13:34:45 2019

-- Model: New Model Version: 1.0

-- MySQL Workbench Forward Engineering

SET @OLD\_UNIQUE\_CHECKS=@@UNIQUE\_CHECKS, UNIQUE\_CHECKS=0;

SET @OLD\_FOREIGN\_KEY\_CHECKS=@@FOREIGN\_KEY\_CHECKS, FOREIGN\_KEY\_CHECKS=0;

SET @OLD\_SQL\_MODE=@@SQL\_MODE, SQL\_MODE='ONLY\_FULL\_GROUP\_BY,STRICT\_TRANS\_TABLES,NO\_ZERO\_IN\_DATE,NO\_ZERO\_DATE,ERROR\_FOR\_DIVISION\_BY\_ZERO,NO\_ENGINE\_SUBSTITUTION';

-- -----------------------------------------------------

-- Schema mydb

-- -----------------------------------------------------

DROP SCHEMA IF EXISTS `mydb` ;

-- -----------------------------------------------------

-- Schema mydb

-- -----------------------------------------------------

CREATE SCHEMA IF NOT EXISTS `mydb` DEFAULT CHARACTER SET utf8 ;

USE `mydb` ;

-- -----------------------------------------------------

-- Table `mydb`.`User`

-- -----------------------------------------------------

DROP TABLE IF EXISTS `mydb`.`User` ;

CREATE TABLE IF NOT EXISTS `mydb`.`User` (

`idUser` INT NOT NULL,

`username` VARCHAR(45) NOT NULL,

`email` VARCHAR(255) NULL,

`password` VARCHAR(32) NOT NULL,

`userMode` VARCHAR(32) NOT NULL,

INDEX `username` (`username` ASC),

PRIMARY KEY (`idUser`),

UNIQUE INDEX `UserId\_UNIQUE` (`idUser` ASC));

-- -----------------------------------------------------

-- Table `mydb`.`Exercise`

-- -----------------------------------------------------

DROP TABLE IF EXISTS `mydb`.`Exercise` ;

CREATE TABLE IF NOT EXISTS `mydb`.`Exercise` (

`idExercise` INT NOT NULL,

`Text` VARCHAR(255) NULL,

PRIMARY KEY (`idExercise`),

UNIQUE INDEX `idExercise\_UNIQUE` (`idExercise` ASC))

ENGINE = InnoDB;

-- -----------------------------------------------------

-- Table `mydb`.`Program`

-- -----------------------------------------------------

DROP TABLE IF EXISTS `mydb`.`Program` ;

CREATE TABLE IF NOT EXISTS `mydb`.`Program` (

`idProgram` INT NOT NULL,

`Path` VARCHAR(255) NULL,

UNIQUE INDEX `ProgramId\_UNIQUE` (`idProgram` ASC),

PRIMARY KEY (`idProgram`))

ENGINE = InnoDB;

-- -----------------------------------------------------

-- Table `mydb`.`TestResult`

-- -----------------------------------------------------

DROP TABLE IF EXISTS `mydb`.`TestResult` ;

CREATE TABLE IF NOT EXISTS `mydb`.`TestResult` (

`Result` VARCHAR(45) NULL,

`Program\_idProgram1` INT NOT NULL,

INDEX `fk\_TestResult\_Program1\_idx` (`Program\_idProgram1` ASC),

PRIMARY KEY (`Program\_idProgram1`),

CONSTRAINT `fk\_TestResult\_Program1`

FOREIGN KEY (`Program\_idProgram1`)

REFERENCES `mydb`.`Program` (`idProgram`)

ON DELETE NO ACTION

ON UPDATE NO ACTION)

ENGINE = InnoDB;

-- -----------------------------------------------------

-- Table `mydb`.`Test`

-- -----------------------------------------------------

DROP TABLE IF EXISTS `mydb`.`Test` ;

CREATE TABLE IF NOT EXISTS `mydb`.`Test` (

`idTest` INT NOT NULL AUTO\_INCREMENT,

`TestInput` VARCHAR(255) NOT NULL,

`Answer` VARCHAR(45) NOT NULL,

`Exercise\_idExercise` INT NOT NULL,

PRIMARY KEY (`idTest`),

INDEX `fk\_Test\_Exercise1\_idx` (`Exercise\_idExercise` ASC),

UNIQUE INDEX `idTest\_UNIQUE` (`idTest` ASC),

CONSTRAINT `fk\_Test\_Exercise1`

FOREIGN KEY (`Exercise\_idExercise`)

REFERENCES `mydb`.`Exercise` (`idExercise`)

ON DELETE NO ACTION

ON UPDATE NO ACTION)

ENGINE = InnoDB;

-- -----------------------------------------------------

-- Table `mydb`.`User\_has\_Exercise`

-- -----------------------------------------------------

DROP TABLE IF EXISTS `mydb`.`User\_has\_Exercise` ;

CREATE TABLE IF NOT EXISTS `mydb`.`User\_has\_Exercise` (

`User\_idUser` INT NOT NULL,

`Exercise\_idExercise` INT NOT NULL,

`idProgram` INT NOT NULL AUTO\_INCREMENT,

INDEX `fk\_User\_has\_Exercise\_Exercise1\_idx` (`Exercise\_idExercise` ASC, `User\_idUser` ASC),

INDEX `fk\_User\_has\_Exercise\_Program1\_idx` (`idProgram` ASC),

PRIMARY KEY (`idProgram`),

UNIQUE INDEX `idUser\_Has\_Exercise\_UNIQUE` (`idProgram` ASC));

SET SQL\_MODE=@OLD\_SQL\_MODE;

SET FOREIGN\_KEY\_CHECKS=@OLD\_FOREIGN\_KEY\_CHECKS;

SET UNIQUE\_CHECKS=@OLD\_UNIQUE\_CHECKS;

Приложение 2

Классы DAO:

С сущностью Exercise:

@Repository

public class ExerciseRepository implements IExerciseRepository {

@Autowired

private JdbcTemplate jdbcTemplate;

private RowMapper<Exercise> RowMap = (rowStr, rowNum) -> new Exercise(

rowStr.getString("text"),

rowStr.getLong("idExercise")

);

public List<Exercise> getAllExercises() {

String sql = "select \* from `mydb`.`Exercise`";

return jdbcTemplate.query(sql, new Object[]{}, RowMap);

}

public Exercise getExercise(Long idExercise) {

String sql = "select \* from `mydb`.`Exercise` where idExercise = ?";

List<Exercise> exercises = jdbcTemplate.query(sql, new Object[]{idExercise},RowMap);

if(exercises.isEmpty()){

return null;

}

else{

return exercises.get(0);

}

}

public void deleteExercise(Long idExercise) {

String sql = "delete Exercise from `mydb`.`Exercise` where idExercise = ?";

jdbcTemplate.update(sql, new Object[]{idExercise});

}

@Override

public void create(Exercise exercise) {

String sql = "insert into `mydb`.`exercise` (`idExercise`,`Text`) VALUES(idExercise, text)";

jdbcTemplate.update(sql, exercise.getIdExercise(), exercise.getText());

}

}

С сущностью Program:

@Repository

public class ProgramRepository implements IProgramRepository {

@Autowired

private JdbcTemplate jdbcTemplate;

private RowMapper<Program> RowMap = (rowStr, rowNum) -> new Program(

rowStr.getLong("idProgram"),

rowStr.getString("Path")

);

public Program getProgram(Long idProgram) {

String sql = "select \* from program where idProgram = ?";

List<Program> programs = jdbcTemplate.query(sql, new Object[]{idProgram}, RowMap);

if (programs.isEmpty()){

return null;

}

else{

return programs.get(0);

}

}

public Long getIdProgram(Long idUser, Long idExercise){

String sql = "select idProgram from user\_Has\_Exercise where User\_idUser = ? and Exercise\_idExercise = ?";

try {

return jdbcTemplate.queryForObject(sql, new Object[]{idUser, idExercise}, Long.*TYPE*);

}

catch (EmptyResultDataAccessException exc){

return null;

}

}

public void create(Program program){

String sql = "insert into `mydb`.`program` (`idProgram`,`path`) VALUES(?, ?)";

jdbcTemplate.update(sql, program.getIdProgram(), program.getPath());

}

@Override

public void createConnection(Long idUser, Long idExercise){

String sql = "insert into `mydb`.`User\_has\_Exercise` (`User\_idUser`,`Exercise\_idExercise`) VALUES(?, ?)";

jdbcTemplate.update(sql, idUser, idExercise);

}

public void updateProgram(Program program) {

String sql = "update program set path = ? where idProgram = ?";

jdbcTemplate.update(sql, program.getPath(), program.getIdProgram());

}

public void deleteProgram(Long idProgram) {

String sql = "delete program from program where idProgram = ?";

jdbcTemplate.update(sql, new Object[]{idProgram});

}

public List<Long> getIdPrograms(Long idExercise){

String sql = "select idProgram from user\_Has\_Exercise where Exercise\_idExercise = ?";

try {

return jdbcTemplate.queryForList(sql,new Object[]{idExercise} , Long.*TYPE*);

}

catch (EmptyResultDataAccessException exc){

return null;

}

}

public Long getIdUserByIdProgram(Long idProgram){

String sql = "select User\_idUser from user\_Has\_Exercise where idProgram = ?";

try {

return jdbcTemplate.queryForObject(sql, new Object[]{idProgram}, Long.*TYPE*);

}

catch (EmptyResultDataAccessException exc){

return null;

}

}

}

С сущностью Test:

@Repository

public class TestRepository implements ITestRepository {

@Autowired

private JdbcTemplate jdbcTemplate;

private RowMapper<UTest> RowMap = (rowStr, rowNum) -> new UTest(

rowStr.getInt("idTest"),

rowStr.getInt("idExercise"),

rowStr.getString("data"),

rowStr.getString("expectedResult")

);

@Override

public void create(UTest test) {

String sql = "insert into `mydb`.`test` (`Exercise\_idExercise`,`testInput`,`answer`)" +

" VALUES(idExercise, data, expectedResult)";

jdbcTemplate.update(sql,

test.getIdExercise(),

test.getData(),

test.getExpectedResult());

}

@Override

public UTest getTest(Long idTest) {

String sql = "select \* from test where idTest = ?";

List<UTest> tests = jdbcTemplate.query(sql,new Object[]{idTest}, RowMap);

if(tests.isEmpty()){

return null;

}

else{

return tests.get(0);

}

}

public List<UTest> getTestsByExerciseId(Long idExercise){

String sql = "select \* from test where Exercise\_idExercise = ?";

return jdbcTemplate.query(sql,new Object[]{idExercise}, RowMap );

}

@Override

public void deleteTest(Long idTest) {

String sql = "delete test from test where idTest = ?";

jdbcTemplate.update(sql, new Object[]{idTest});

}

}

С сущностью TestResult:

@Repository

public class TestResultRepository implements ITestResultRepository {

@Autowired

private JdbcTemplate jdbcTemplate;

private RowMapper<TestResult> RowMap = (rowStr, rowNum) -> new TestResult(

rowStr.getLong("Program\_idProgram1"),

rowStr.getString("result")

);

@Override

public void create(TestResult testResult) {

String sql = "INSERT INTO `mydb`.`TestResult` (`result`, `Program\_idProgram1`) " +

"VALUES (?, ?)";

jdbcTemplate.update(sql, testResult.getResult(), testResult.getIdProgram());

}

@Override

public TestResult getByIdProgram(Long idProgram) {

String sql = "select \* from testResult where Program\_idProgram1 = ?";

List<TestResult> testResults = jdbcTemplate.query(sql, new Object[]{idProgram}, RowMap);

if(testResults.isEmpty()){

return null;

}

else{

return testResults.get(0);

}

}

public void updateTestResult(TestResult testResult) {

String sql = "UPDATE TestResult SET result = ? where Program\_idProgram1 = ?";

jdbcTemplate.update(sql, testResult.getResult(), testResult.getIdProgram());

}

}

С сущностью User:

@Repository

public class UserRepository implements IUserRepository {

@Autowired

private JdbcTemplate jdbcTemplate;

public List<User> getAllUsers() {

String sql = "SELECT \* FROM user";

List<User> users = jdbcTemplate.query(sql, new Object[]{}, RowMap);

return users;

}

private RowMapper<User> RowMap = (rowStr, rowNum) -> new User(

rowStr.getLong("idUser"),

rowStr.getString("username"),

rowStr.getString("email"),

rowStr.getString("password"),

rowStr.getString("userMode")

);

@Override

public void create(User user) {

String sql = "insert into `mydb`.`user` (`idUser`, `username`, `email`, `password`, `userMode`) VALUES (?, ?, ?, ?,'user')";

jdbcTemplate.update(sql, user.getName(),

user.getName(),

user.getEmail(),

user.getPassword());

}

@Override

public User getUser(Long idUser) {

String sql = "SELECT \* FROM user where idUser = ?";

List<User> users = jdbcTemplate.query(sql, new Object[]{idUser}, RowMap);

if(users.isEmpty()){

return null;

} else {

return users.get(0);

}

}

@Override

public void updateUser(User user) {

String sql = "UPDATE user SET username = ?, email = ?, password = ?, role = ?, ban = ? where idUser = ?";

jdbcTemplate.update(sql, user.getName(),

user.getEmail(),

user.getPassword(),

user.getUserMode(),

user.getIdUser());

}

@Override

public void deleteUser(Long idUser) {

String sql = "DELETE user from user where idUser = ?";

jdbcTemplate.update(sql, new Object[]{idUser});

}

@Override

public User getUserByLogin(String name) {

String sql = "SELECT \* FROM user where username = ?";

List<User> users = jdbcTemplate.query(sql, new Object[]{name}, RowMap);

if(users.isEmpty()){

return null;

} else {

return users.get(0);

}

}

}

Приложение 3

Ввод тестовых данных в таблицу:

insert into `mydb`.`user` (`idUser`,`username`, `email`, `password`, `userMode`) VALUES (1, 'admin', 1, 1, 'admin');

insert into `mydb`.`user` (`idUser`,`username`, `email`, `password`, `userMode`) VALUES (2, 'testUser1', 1, 2, 'user');

insert into `mydb`.`user` (`idUser`,`username`, `email`, `password`, `userMode`) VALUES (3, 'testUser2', 1, 3, 'user');

insert into `mydb`.`user` (`idUser`,`username`, `email`, `password`, `userMode`) VALUES (4, 'testUser2', 1, 4, 'user');

insert into `mydb`.`user` (`idUser`,`username`, `email`, `password`, `userMode`) VALUES (5, 'testUser2', 1, 5, 'user');

insert into `mydb`.`user` (`idUser`,`username`, `email`, `password`, `userMode`) VALUES (6, 'testUser2', 1, 6, 'user');

insert into `mydb`.`exercise` (`idExercise`,`Text`) VALUES(1, '1 Exerise');

insert into `mydb`.`exercise` (`idExercise`,`Text`) VALUES(2, '2 Exerise');

insert into `mydb`.`exercise` (`idExercise`,`Text`) VALUES(3, '4 Exerise');

insert into `mydb`.`exercise` (`idExercise`,`Text`) VALUES(4, '3 Exerise');

insert into `mydb`.`exercise` (`idExercise`,`Text`) VALUES(5, '5 Exerise');

insert into `mydb`.`test` (`Exercise\_idExercise`,`testInput`,`answer`) VALUES(1, 'Hello World', 'Hello');

insert into `mydb`.`test` (`Exercise\_idExercise`,`testInput`,`answer`) VALUES(1, 'Bye', 'Bye');

insert into `mydb`.`test` (`Exercise\_idExercise`,`testInput`,`answer`) VALUES(1, 'Do you wann hang out?', 'No, ur a creep');

insert into `mydb`.`test` (`Exercise\_idExercise`,`testInput`,`answer`) VALUES(2, 'Hello World', 'Hello');

insert into `mydb`.`test` (`Exercise\_idExercise`,`testInput`,`answer`) VALUES(3, 'Bye', 'Bye');

insert into `mydb`.`test` (`Exercise\_idExercise`,`testInput`,`answer`) VALUES(4, 'Do you wann hang out?', 'No, ur a creep');

insert into `mydb`.`test` (`Exercise\_idExercise`,`testInput`,`answer`) VALUES(5, '3', '1');

insert into `mydb`.`test` (`Exercise\_idExercise`,`testInput`,`answer`) VALUES(5, '2', '0');

insert into `mydb`.`test` (`Exercise\_idExercise`,`testInput`,`answer`) VALUES(5, '1', '-1');

insert into `mydb`.`User\_has\_Exercise` (`User\_idUser`,`Exercise\_idExercise`) VALUES(2, 1);

insert into `mydb`.`User\_has\_Exercise` (`User\_idUser`,`Exercise\_idExercise`) VALUES(2, 2);

insert into `mydb`.`User\_has\_Exercise` (`User\_idUser`,`Exercise\_idExercise`) VALUES(2, 3);

insert into `mydb`.`User\_has\_Exercise` (`User\_idUser`,`Exercise\_idExercise`) VALUES(3, 1);

insert into `mydb`.`User\_has\_Exercise` (`User\_idUser`,`Exercise\_idExercise`) VALUES(3, 4);

insert into `mydb`.`User\_has\_Exercise` (`User\_idUser`,`Exercise\_idExercise`) VALUES(3, 5)

Результат запроса select \* from ‘mydb’.’user’

[

{

"idUser" : 1,

"username" : "admin",

"email" : "1",

"password" : "1",

"userMode" : "admin"

},

{

"idUser" : 2,

"username" : "testUser1",

"email" : "1",

"password" : "2",

"userMode" : "user"

},

{

"idUser" : 3,

"username" : "testUser2",

"email" : "1",

"password" : "3",

"userMode" : "user"

},

{

"idUser" : 4,

"username" : "testUser2",

"email" : "1",

"password" : "4",

"userMode" : "user"

},

{

"idUser" : 5,

"username" : "testUser2",

"email" : "1",

"password" : "5",

"userMode" : "user"

},

{

"idUser" : 6,

"username" : "testUser2",

"email" : "1",

"password" : "6",

"userMode" : "user"

},

{

"idUser" : 23,

"username" : "23",

"email" : "23",

"password" : "23",

"userMode" : "user"

}

]

Результат запроса select \* from ‘mydb’.’exercise’

[

{

"idExercise" : 1,

"Text" : "1 Exerise"

},

{

"idExercise" : 2,

"Text" : "2 Exerise"

},

{

"idExercise" : 3,

"Text" : "4 Exerise"

},

{

"idExercise" : 4,

"Text" : "3 Exerise"

},

{

"idExercise" : 5,

"Text" : "5 Exerise"

},

{

"idExercise" : 23,

"Text" : **null**

}

]

Результат запроса select idProgram from ‘mydb’.’user\_has\_exercise’

[

{

"idProgram" : 1

},

{

"idProgram" : 2

},

{

"idProgram" : 3

},

{

"idProgram" : 4

},

{

"idProgram" : 5

},

{

"idProgram" : 6

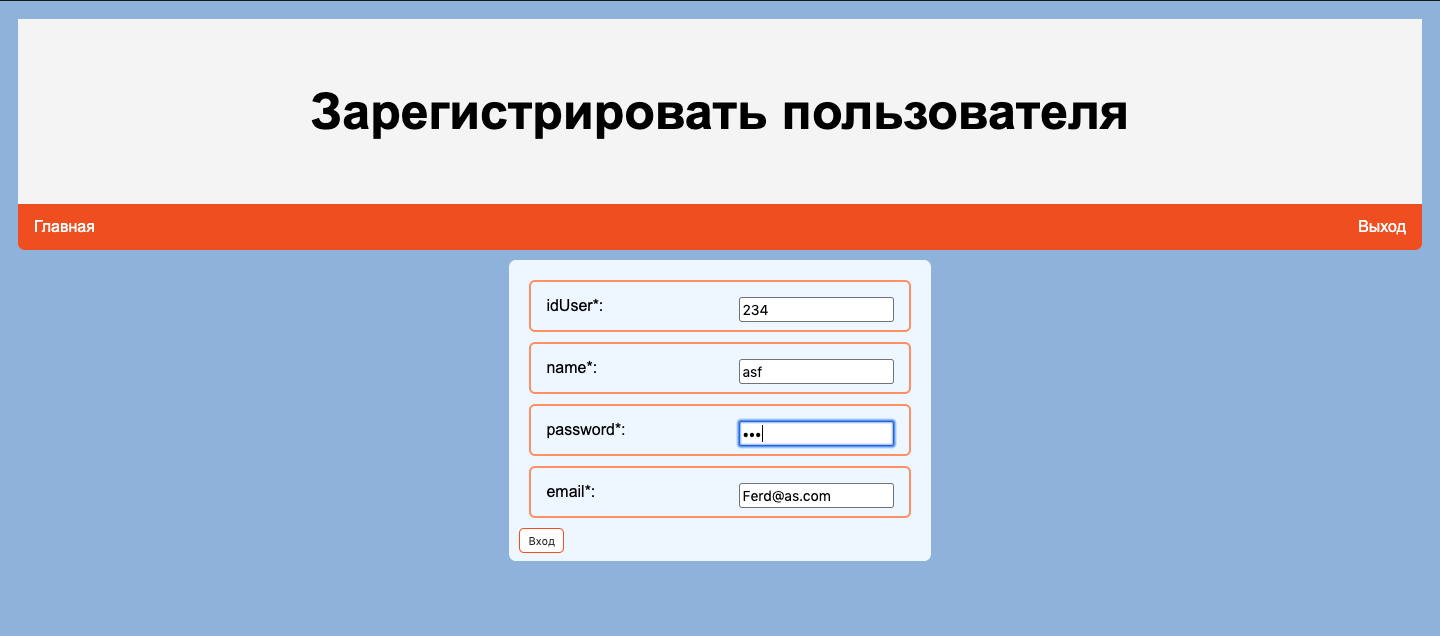
},

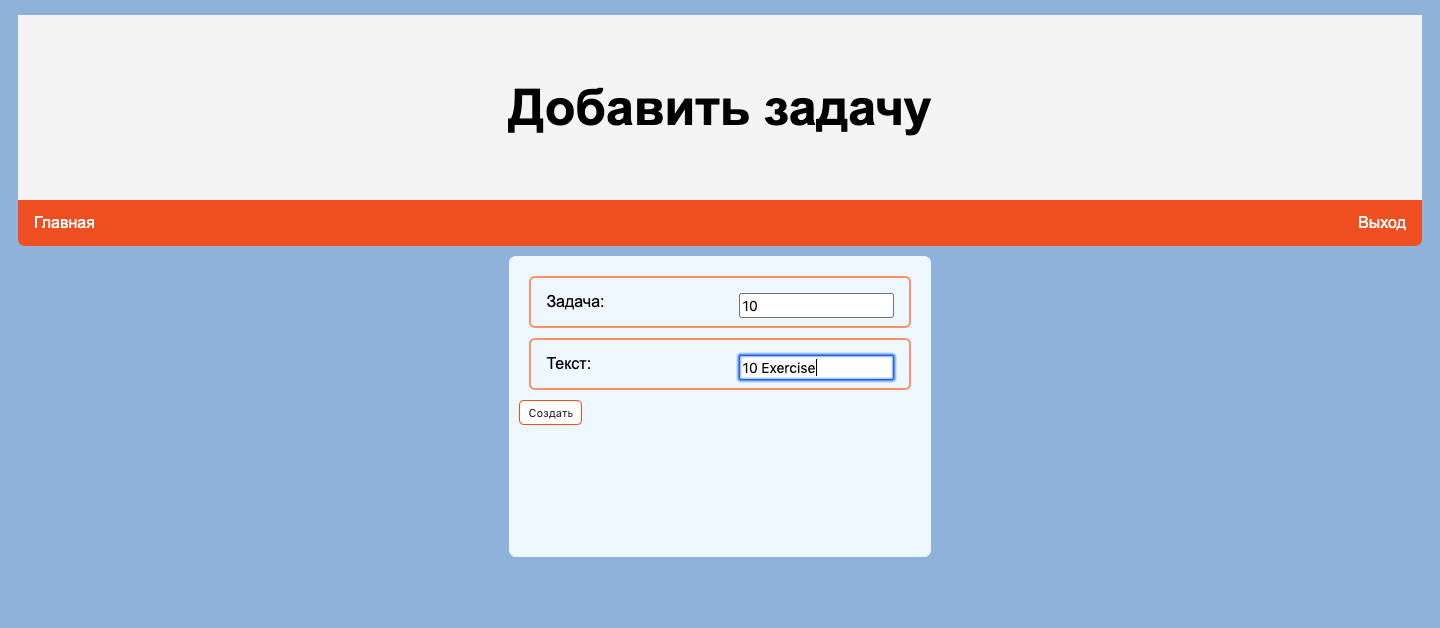
{

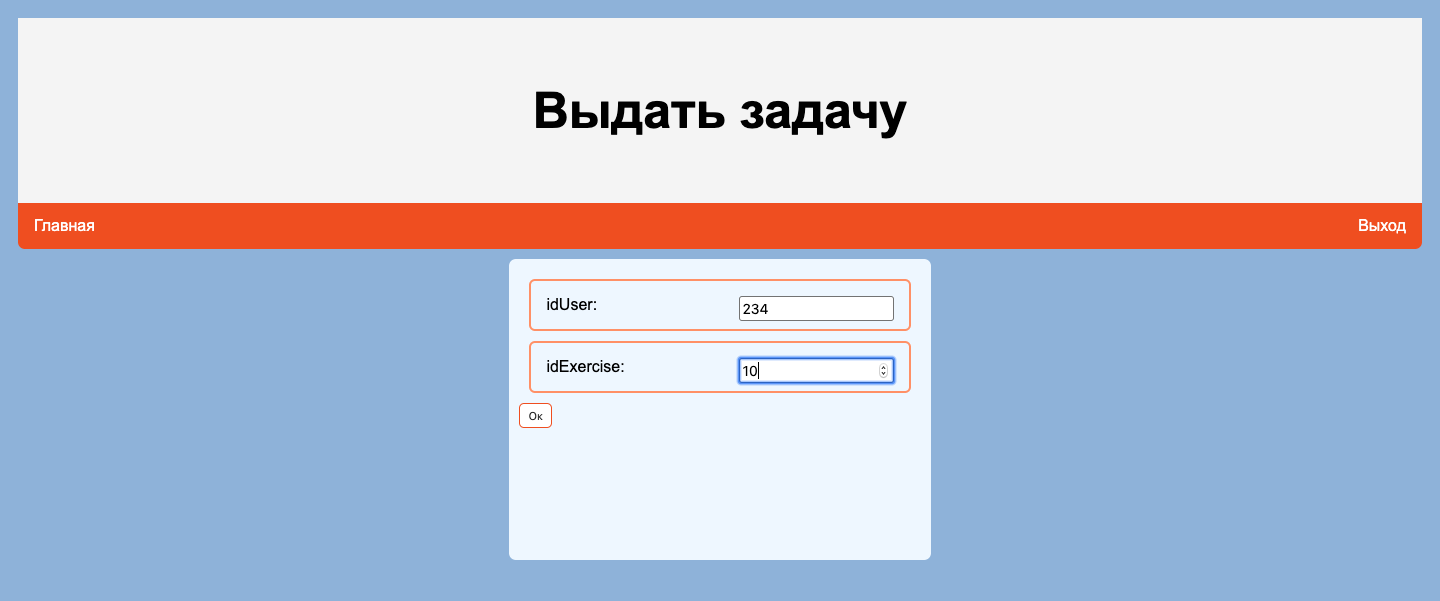
"idProgram" : 7

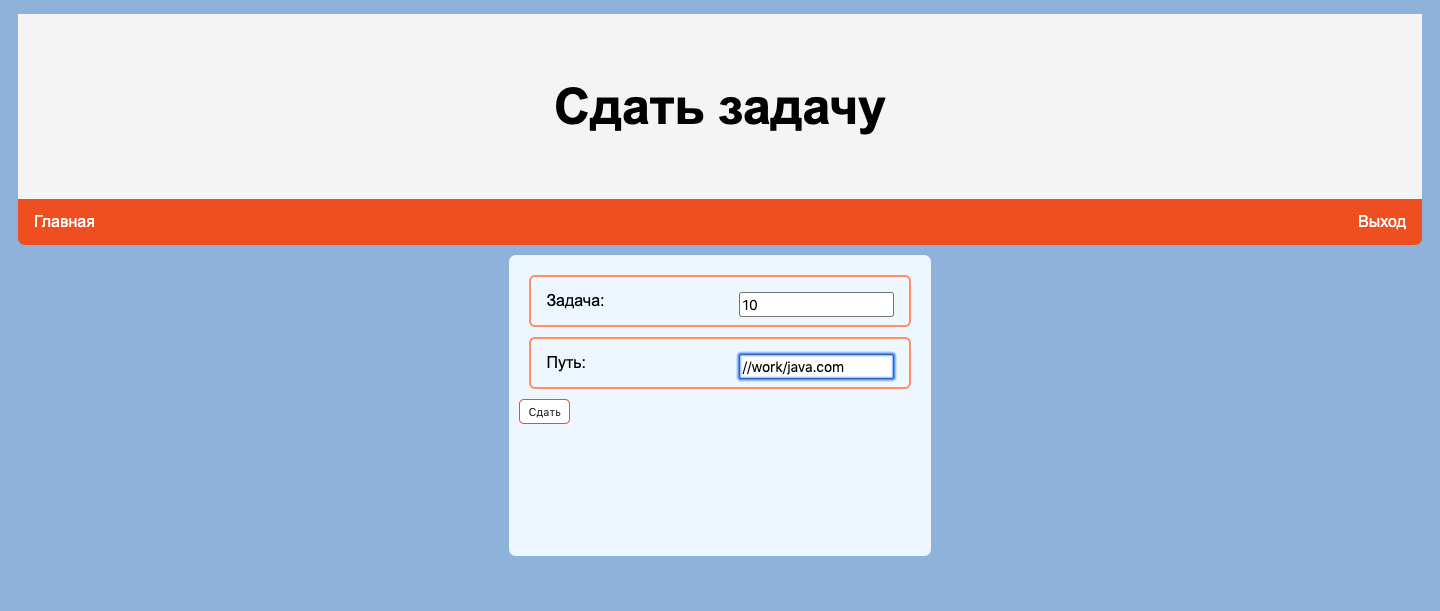
}

]

Добавление нового пользователя

Добавление новой задачи

Назначение задачи пользователю

Сдача программы

Проверка задачи