

Колледж космического машиностроения и технологий

Отчет

Производственной практике

Тема: «Разработка приложения «TimeManadge» на Java»

Выполнил студент

Безбородов Константин Владимирович

Группа П2-18

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_(Подпись)

\_\_\_\_\_\_\_\_\_\_\_\_ (Дата сдачи работы)

Проверил преподаватель

Попов Вячеслав Николаевич

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_(Подпись)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_(Оценка)

Королёв 2021 г.

Оглавление

[**Введение** 3](#_Toc75379030)

[**1 Теоретическая часть** 3](#_Toc75379031)

[**Описание предметной области** 3](#_Toc75379032)

[**1.2 Описание существующих разработок** 5](#_Toc75379033)

[**2** **Проектная часть** 7](#_Toc75379034)

[**2.1** **Диаграмма прецедентов** 7](#_Toc75379035)

[**2.2 Выбор инструментов** 7](#_Toc75379036)

[**2.3 Проектирование сценария** 8](#_Toc75379037)

[**2.4 Диаграмма классов** 9](#_Toc75379038)

[**2.5** **Описание главного модуля** 12](#_Toc75379039)

[**2.6** **Описание спецификаций к модулям** 35](#_Toc75379040)

[**2.7** **Описание модулей** 35](#_Toc75379041)

[**2.8** **Описание применения средств отладки** 36](#_Toc75379042)

[**3** **Эксплуатационная часть** 36](#_Toc75379043)

[**3.1** **Руководство оператора** 36](#_Toc75379044)

[**1.1.** **Выполнение программы** 38](#_Toc75379045)

[**1.1.1.** **Выполнение функции открытия окна таблиц.** 38](#_Toc75379046)

[**1.1.2.** **Выполнение функции смены таблицы.** 38](#_Toc75379047)

[**1.1.3.** **Выполнение функции добавления элемента в таблицу.** 39](#_Toc75379048)

[**1.1.4.** **Выполнение функции удаления элемента из таблицы.** 39](#_Toc75379049)

[**Заключение** 40](#_Toc75379050)

[**Дневник практики** 41](#_Toc75379051)

[**Список литературы и интернет-источников** 41](#_Toc75379052)

# **Введение**

Целью данного проекта является написание серверной части программы, для формирования отчетов. Тема серверной части является актуальной на данный момент потому что в наше время все приложения веб-приложения имеют как front-end часть, так и back-end часть.

Данный проект позволит получить информацию с сервера, на базе которой можно разработать гибкое взаимодействие с данными

В первой части будет рассмотрена предметная область данной темы, а также несколько продуктов по данной теме.

Во второй части будут рассмотрены инструменты и модули, которые были разработаны, структура программной части и листинги ключевых частей программных модулей.

В третьей части будет рассмотрено руководство для пользователей.

В заключительной части будет приведен To-do лист с планами по доработки программы, а также сделаны общие выводы о получившемся проекте.

# **1 Теоретическая часть**

## **Описание предметной области**

Подготовка отчетов - это распространенная задача каждого предприятия, владелец или директор обязан вести отчетность отработанных часов его сотрудников. На многих предприятиях имеется работник, выполняющий данную задачу, в большинстве случаем они предпочитают использовать программы, для их составления. Она позволяет грамотно распланировать совершение всех операций на предприятии и обеспечивает удобное хранение информации. Это открывает новые перспективы для развития отчетности предприятия.

С помощью программ отслеживающих рабочее время сотрудников, можно легко получить выводы о эффективности их работы, всегда иметь отчет о рабочих часах сотрудников.

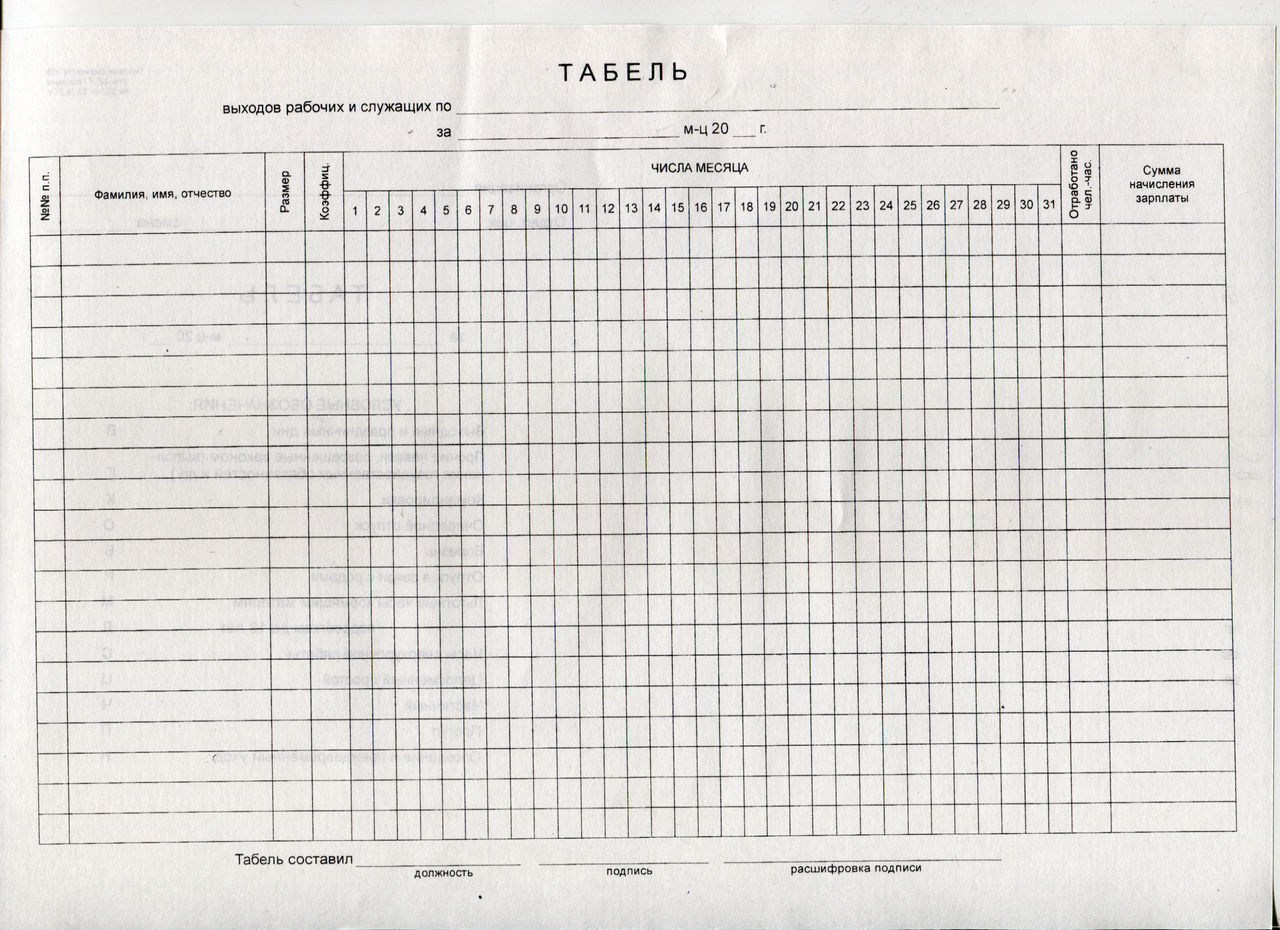


Рисунок 1. Табель о рабочих часах(бумажная версия)

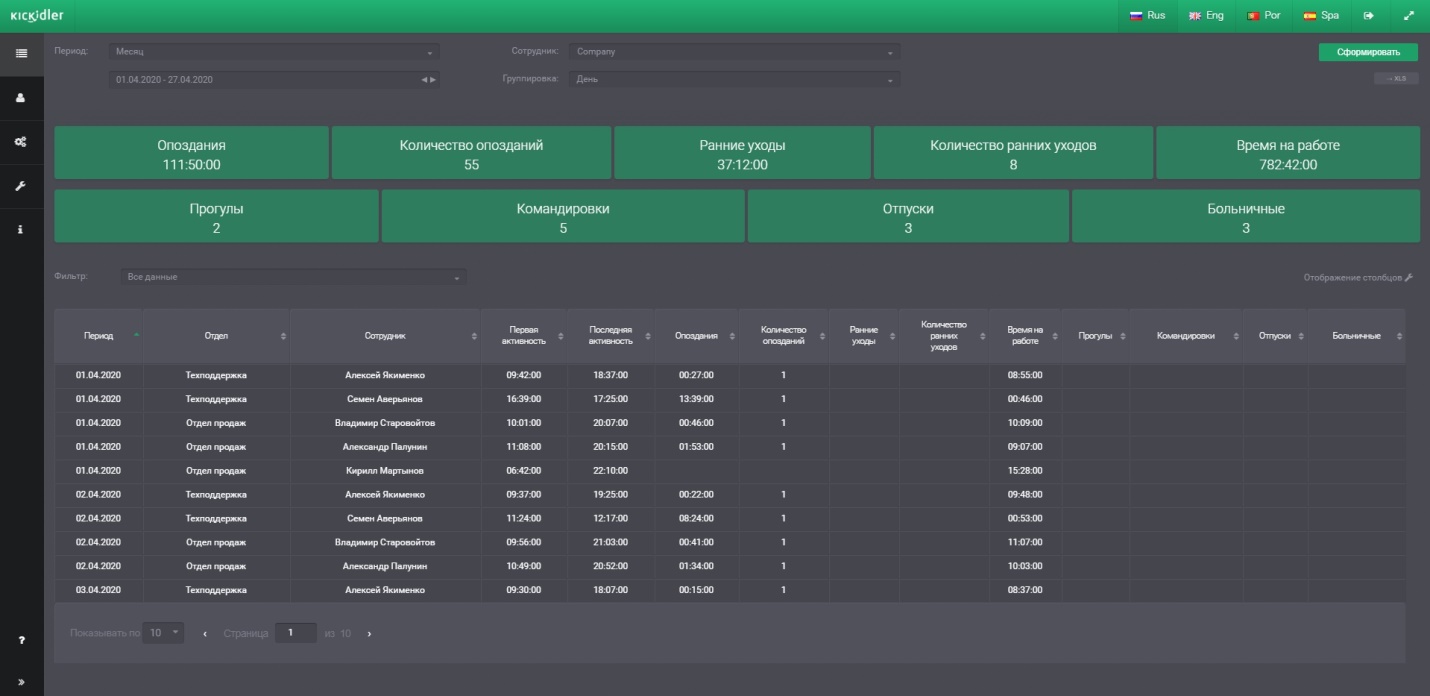


Рисунок 1. Табель о рабочих часах(электронная версия)

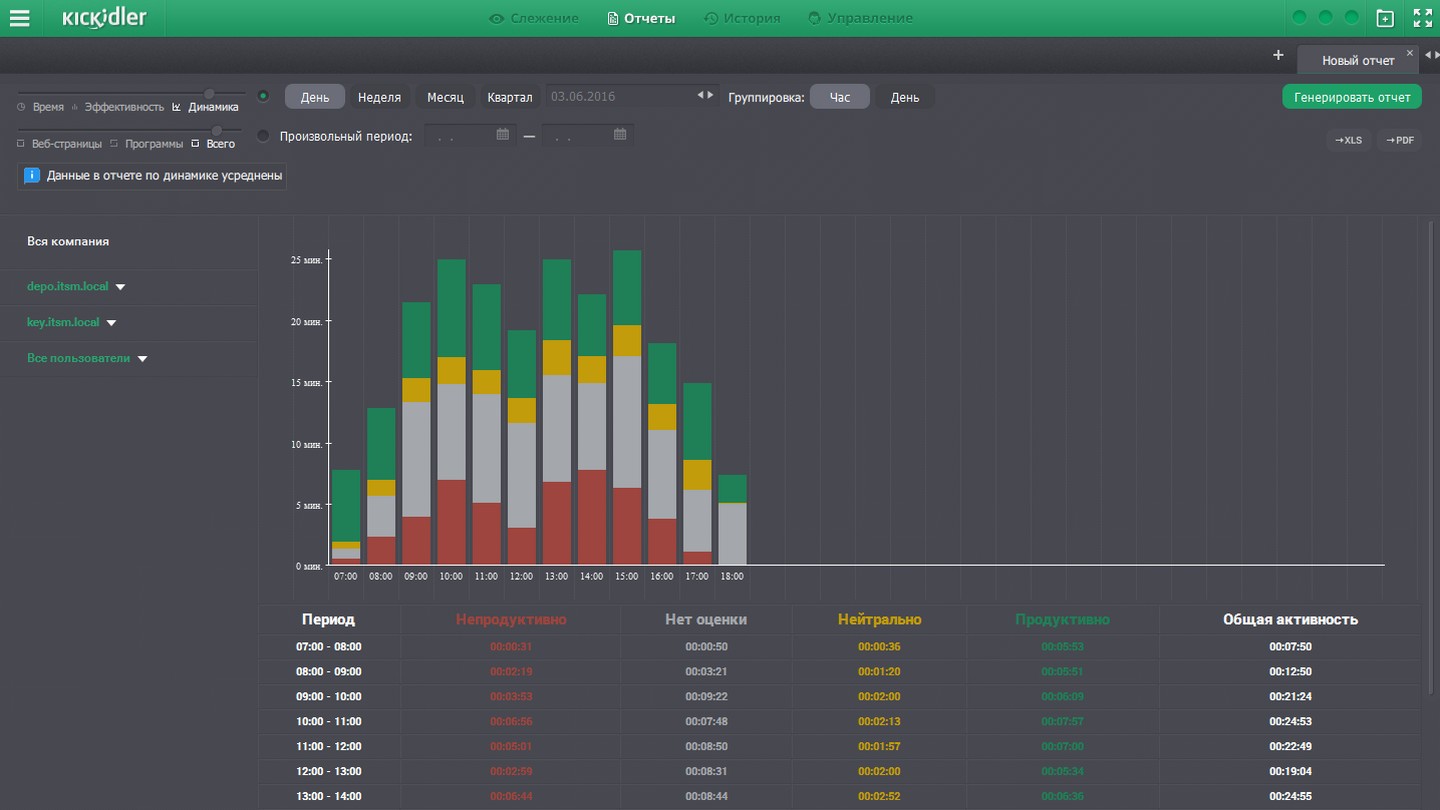


Рисунок 3. Динамика рабочего времени

## **1.2 Описание существующих разработок**

В это разделе рассмотрены некоторые уже имеющиеся программы для управления отчетами и их составлением, а также цели таких программ и требования.

Интерфейс таких программ чаще всего простой и понятный для обычных пользователей.

Цели использования систем составления отчетов:

* Автоматизация работы сотрудников
* Синхронизация отчетности
* Ведение бухгалтерии
* Оформление исходящих документов

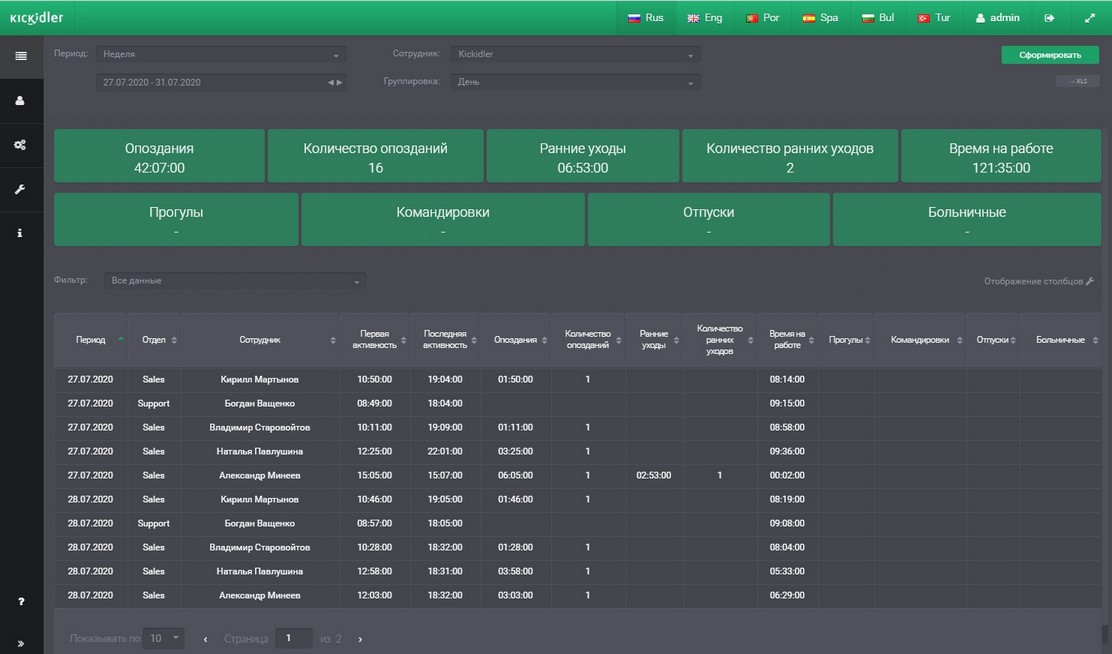
1.2 Kickidler 

Рисунок 4. Учет рабочего времени сотрудников kickidler

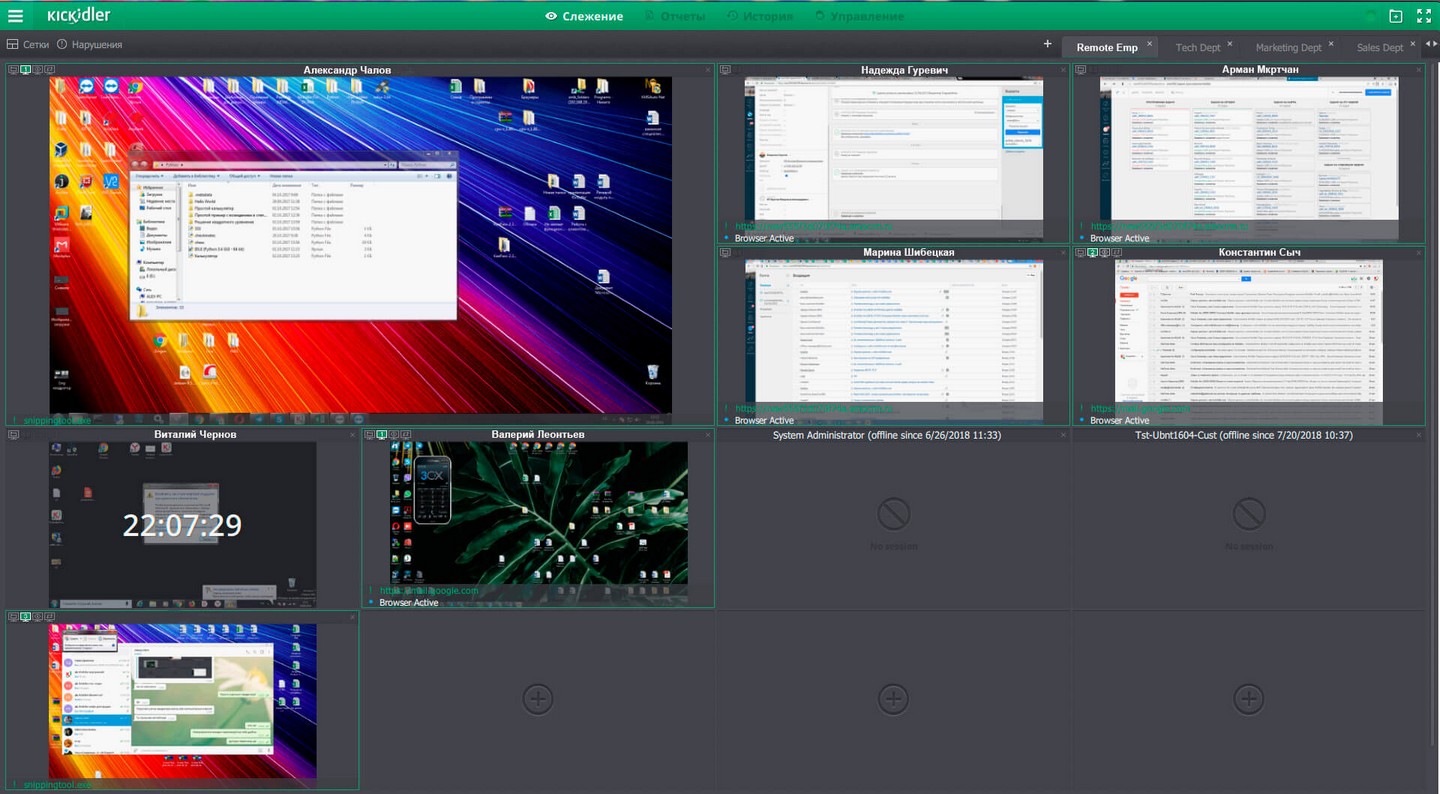


Рисунок 5. Контроль удаленных сотрудников

# **Проектная часть**

## **Диаграмма прецедентов**

В этом разделе представлены две диаграммы прецедентов. На первой диаграмме показаны все возможные функциональные и поведенческие отношения.

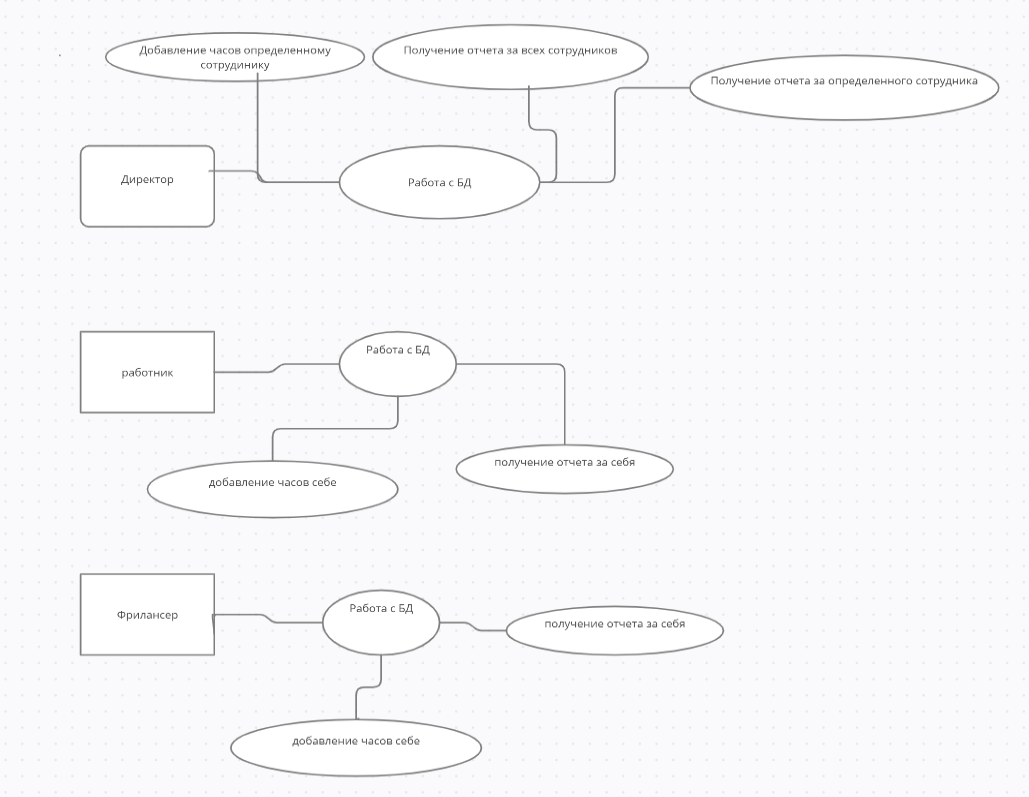


Рисунок 6. Диаграмма прецедентов работы

## **2.2 Выбор инструментов**

При выборе инструментов было проведено сравнение по критериям, представленных в таблице 1.

Степень важности критерия выбиралась из: низкая, ниже средней, средняя, ниже высокой, высокая.

Таблица 1. Критерии выбора инструмента.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Критерий | Участие в корпоративном проекте | Простота сопровождения | Наличие фреймворков | Наличие документации на русском языке | Скорость разработки |
| Важность критерия | Высокая | Средняя | Высокая | Ниже средней | Ниже высокой |

Исходя из этих критериев, я сравнил 3 языка программирования от 0 до 10 баллов за критерий.

Таблица 2. Оценка языков программирования.

|  |  |  |  |
| --- | --- | --- | --- |
|  | C++ | Python | Java |
| Участие в корпоративном проекте | 10 | 8 | 10 |
| Простота сопровождения | 7 | 10 | 9 |
| Наличие фреймворка | 6 | 10 | 10 |
| Наличие документации на русском языке | 8 | 5 | 5 |
| Скорость разработки | 6 | 7 | 9 |
| Итого баллов | 37 | 41 | 43 |

По результатам сравнения был выбран язык программирования java.

## **2.3 Проектирование сценария**

Функционал роли руководитель: (mapping localhost:8080/Director/ Может добавлять новых сотрудников, (/addNewWorker с параметрами name, passowrd, role

Может добавлять сотрудникам отработанные часы (/addHoursToWorkker с параметрами name, date, hours)

Может получать отчет по отработанным часам по датам для определенного сотрудника (/makeReportForWorker с параметрами name, date, daysValue может быть day,week,month)

Может получать отчет по отработанным часам по датам для всех сотрудников (/makeReportForAll с параметрами начала date , и интервала, который может быть day,week,month)

Может удалять работника (/removeWorker с парамтрами name)

Функционал роли штатный работник: (mapping localhost:8080/Worker/

Может добавлять отработанные часы только за себя (/selfHoursAdd с параметрами date, и hours)

Может получать отчет только за себя (/selfReport с параметрами date, daysValue может быть day,week,month) Функционал роли фрилансер(mapping localhost:8080/Freelance

Может добавлять часы себе, только если от текущей даты прошло не более двух дней(/selfHoursAdd с параметрами date, hours)

Может получать отчет только за себя (/selfReport с параметрами date, daysValue может быть day,week,month).

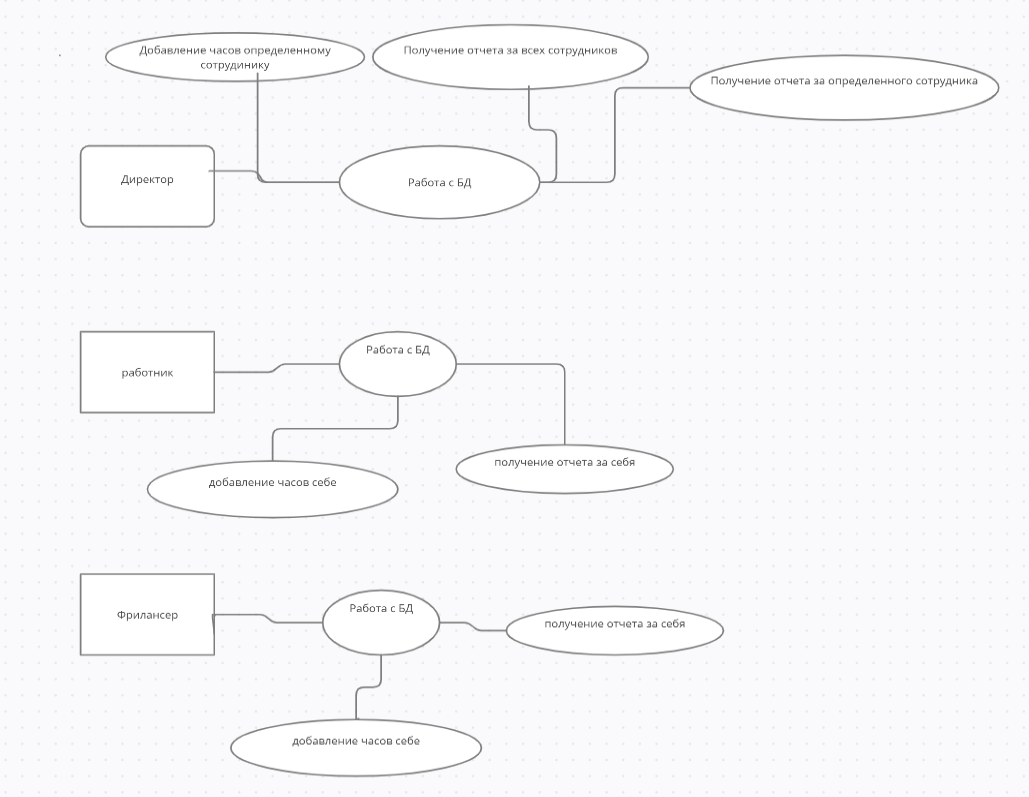


Рисунок 7. Диаграмма сценария

## **2.4 Диаграмма классов**

В данном разделе представлены все классы, использующиеся в проекте, а также их отношения между собой.

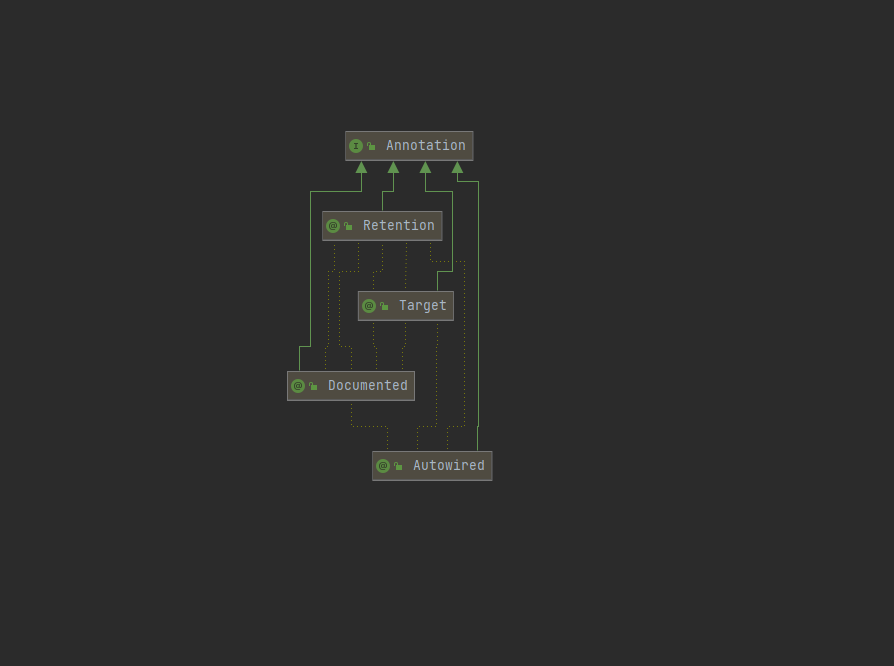


Рисунок 7. Диаграмма классов

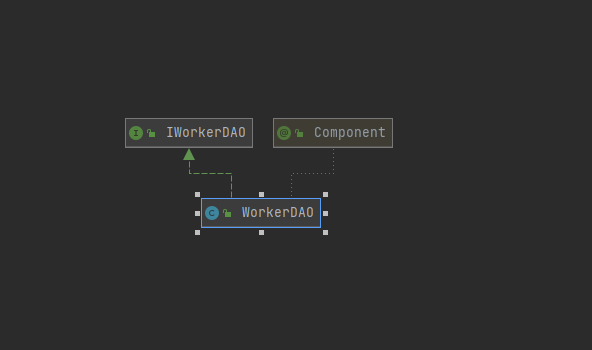


Рисунок 8. Диаграмма классов

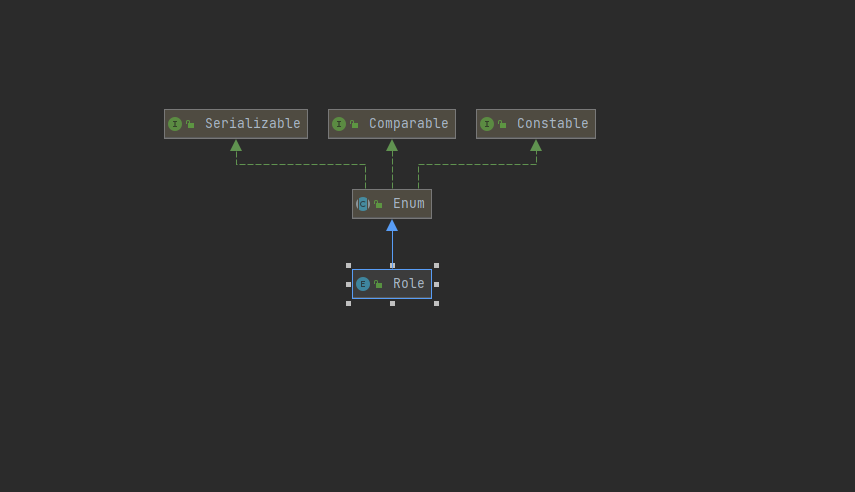


Рисунок 9. Диаграмма классов

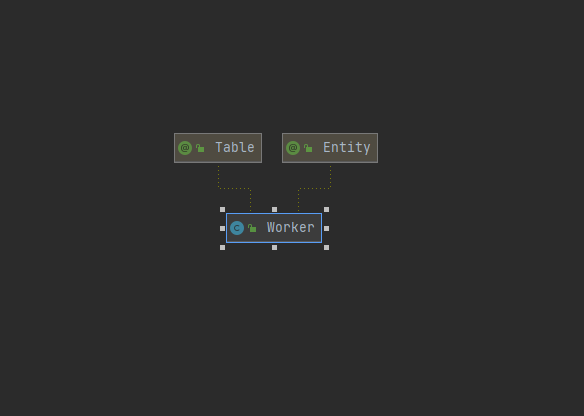


Рисунок 10. Диаграмма классов

* 1. **Описание главного модуля**

В главный модуль входит класс SoftDevTimeSheetApplication который отвечает за запуск сервера.

В главный модуль также импортируется еще 4 модуля один из который отвечает за связь с базой данных, один за работу авторизации, один для запуска сервера, и один для обработки запросов.

Также в главный модуль входит код, который запускает сам сервер при ее запуске. Полный код главного модуля находится в приложении 1.

**Листинг 1. Главный класс запуска сервера**

**package com.SoftDevTimeSheet;**

**import org.springframework.boot.SpringApplication;**

**import org.springframework.boot.autoconfigure.SpringBootApplication;**

**//main spring boot method**

**@SpringBootApplication**

**public class SoftDevTimeSheetApplication {**

**public static void main(String[] args) {**

**SpringApplication.run(SoftDevTimeSheetApplication.class, args);**

**}  
}**

**Листинг 2. интерфейс для работы с БД**

**package com.SoftDevTimeSheet.dao.DAOInterfaces;**

**import com.SoftDevTimeSheet.dao.ReportDaysValue;**

**import com.SoftDevTimeSheet.entity.WorkedDay;**

**import java.time.LocalDate;**

**import java.util.List;**

**public interface IWorkerDAO {**

**public boolean addHoursToWorker(String workerName, LocalDate date, int hoursToAdd);**

**public List<WorkedDay>getReportForAllWorkers(LocalDate date, ReportDaysValue reportDaysValue);**

**public List<WorkedDay> getReportForWorkerByName(String workerName, LocalDate date, ReportDaysValue reportDaysValue);**

**public int getTotalHoursWorkedByWorkerName(String workerName);**

**public boolean addNewWorkerWithRole(String workerName, String password, String role);**

**}**

**Листинг 3. модуль соединения сервера с базой данных**

**package com.SoftDevTimeSheet.dao;**

**import java.sql.Connection;**

**import java.sql.DriverManager;**

**import java.sql.SQLException;**

**public class DAOConnectionFactory {**

**// static method to get DAO connection to postgres**

**public static Connection getConnection() throws SQLException {//connector creater**

**final String userLogin = "postgres";**

**final String pass = "admin";**

**final String url = "jdbc:postgresql://localhost:5432/practice";**

**return DriverManager.getConnection(url,userLogin,pass);**

**}**

**}**

**Листинг 4. модуль взаимодействия с базой данных**

**package com.SoftDevTimeSheet.dao;**

**import com.SoftDevTimeSheet.dao.DAOInterfaces.IWorkerDAO;**

**import com.SoftDevTimeSheet.entity.WorkedDay;**

**import java.sql.Connection;**

**import java.sql.PreparedStatement;**

**import java.sql.ResultSet;**

**import java.sql.SQLException;**

**import java.time.LocalDate;**

**import java.util.ArrayList;**

**import java.util.List;**

**public class WorkerDAO implements IWorkerDAO {**

**@Override**

**public boolean addHoursToWorker(String workerName, LocalDate date, int hoursToAdd) { // timesheet method, that adds hours to worker by name**

**Connection connection = null;**

**PreparedStatement statement = null;**

**try{**

**connection = DAOConnectionFactory.getConnection();**

**try{**

**statement = connection.prepareStatement("INSERT INTO timesheet (name, date, hours) values ((?),(?),(?))");**

**statement.setString(1,workerName);**

**statement.setString(2, String.valueOf(date));**

**statement.setInt(3, hoursToAdd);**

**if(statement.executeUpdate() > 0){**

**return true;**

**}**

**}catch (SQLException e){**

**System.out.println(e);**

**}finally {**

**if(statement != null){**

**statement.close();**

**}**

**}**

**}catch (SQLException e){**

**System.out.println(e);**

**}finally {**

**if(connection != null){**

**try {**

**connection.close();**

**}catch (SQLException e){**

**System.out.println(e);**

**}**

**}**

**}**

**return false;**

**}**

**@Override**

**public List<WorkedDay> getReportForAllWorkers(LocalDate date, ReportDaysValue reportDaysValue) { //Director report maker DAO for all workers**

**Connection connection = null;**

**PreparedStatement statement = null;**

**ResultSet resultSet = null;**

**List<WorkedDay> allWorkersReport = null;**

**try{**

**connection = DAOConnectionFactory.getConnection();**

**try{**

**statement = connection.prepareStatement("SELECT name, date, hours from timesheet where date between (?) and (?)");**

**statement.setString(1,String.valueOf(date.minusDays(reportDaysValue.getValue())));**

**statement.setString(2,String.valueOf(date));**

**allWorkersReport = new ArrayList<>(10);**

**resultSet = statement.executeQuery();**

**while(resultSet.next()){**

**WorkedDay workedDay = new WorkedDay();**

**workedDay.setWorkerName(resultSet.getString("name"));**

**workedDay.setWorkDate(LocalDate.parse(resultSet.getString("date")));**

**workedDay.setHoursWorked(resultSet.getInt("hours"));**

**allWorkersReport.add(workedDay);**

**}**

**}catch (SQLException e){**

**System.out.println(e);**

**}finally {**

**if(statement != null){**

**statement.close();**

**}**

**if(resultSet != null){**

**resultSet.close();**

**}**

**}**

**}catch(SQLException e){**

**System.out.println(e);**

**}finally {**

**if(connection != null){**

**try {**

**connection.close();**

**}catch (SQLException e){**

**System.out.println(e);**

**}**

**}**

**}**

**return allWorkersReport;**

**}**

**// method that getting a report for worker by**

**//worker name**

**@Override**

**public List<WorkedDay> getReportForWorkerByName(String workerName, LocalDate date, ReportDaysValue reportDaysValue) {**

**Connection connection = null;**

**PreparedStatement statement = null;**

**ResultSet resultSet = null;**

**List<WorkedDay> reportForWorker = null;**

**try{**

**connection = DAOConnectionFactory.getConnection();**

**try{**

**statement = connection.prepareStatement("SELECT name, date, hours FROM timesheet where name = (?) AND date BETWEEN (?) AND (?)");**

**statement.setString(1, workerName);**

**statement.setString(2, String.valueOf(date.minusDays(reportDaysValue.getValue())));**

**statement.setString(3, String.valueOf(date));**

**reportForWorker = new ArrayList<>(10);**

**resultSet = statement.executeQuery();**

**while(resultSet.next()){**

**WorkedDay workedDay = new WorkedDay();**

**workedDay.setWorkerName(resultSet.getString("name"));**

**workedDay.setWorkDate(LocalDate.parse(resultSet.getString("date")));**

**workedDay.setHoursWorked(resultSet.getInt("hours"));**

**reportForWorker.add(workedDay);**

**}**

**}catch (SQLException e){**

**System.out.println(e);**

**}finally {**

**if(statement != null){**

**statement.close();**

**}**

**if(resultSet != null){**

**resultSet.close();**

**}**

**}**

**}catch (SQLException e){**

**System.out.println(e);**

**}finally {**

**try {**

**if (connection != null) {**

**connection.close();**

**}**

**}catch (SQLException e){**

**System.out.println(e);**

**}**

**}**

**return reportForWorker;**

**}**

**//DAO method that returns number of worked hours by worker name**

**@Override**

**public int getTotalHoursWorkedByWorkerName(String workerName) {**

**Connection connection = null;**

**PreparedStatement statement = null;**

**ResultSet resultSet = null;**

**int totalHoursWorked = 0;**

**try{**

**connection = DAOConnectionFactory.getConnection();**

**try{**

**statement = connection.prepareStatement("SELECT hours from timesheet where name = (?)");**

**statement.setString(1,workerName);**

**resultSet = statement.executeQuery();**

**while (resultSet.next()){**

**totalHoursWorked += resultSet.getInt("hours");**

**}**

**}catch (SQLException e){**

**System.out.println(e);**

**}finally {**

**if(statement != null){**

**statement.close();**

**}**

**if(resultSet != null){**

**resultSet.close();**

**}**

**}**

**}catch (SQLException e){**

**System.out.println(e);**

**}finally {**

**try {**

**if (connection != null) {**

**connection.close();**

**}**

**}catch (SQLException e) {**

**System.out.println(e);**

**}**

**}**

**return totalHoursWorked;**

**}**

**@Override**

**public boolean addNewWorkerWithRole(String workerName, String password, String role) {**

**Connection connection = null;**

**PreparedStatement statement = null;**

**try{**

**connection = DAOConnectionFactory.getConnection();**

**try{**

**statement = connection.prepareStatement("INSERT INTO roles (name, password, role)values ((?),(?),(?))");**

**statement.setString(1,workerName);**

**statement.setString(2,password);**

**statement.setString(3, role);**

**if(statement.executeUpdate() > 0){**

**return true;**

**}**

**}catch(SQLException e){**

**if(statement !=null){**

**statement.close();**

**}**

**}**

**}catch (SQLException e){**

**System.out.println(e);**

**}finally {**

**try{**

**if(connection != null) {**

**connection.close();**

**}**

**}catch (SQLException e){**

**System.out.println(e);**

**}**

**}**

**return false;**

**}**

**}**

**Листинг 5. Класс хранящий виды периодов данных**

**package com.SoftDevTimeSheet.dao;**

**public enum ReportDaysValue {**

**//month = 31 day, week = 7 day, day = 1day**

**MONTH(31), WEEK(7), DAY(1);**

**private int value;**

**ReportDaysValue(int value){**

**this.value = value;**

**}**

**public int getValue() {**

**return value;**

**}**

**public void setValue(int value) {**

**this.value = value;**

**}**

**}**

**Листинг 6. класс сущности прав**

**package com.SoftDevTimeSheet.entity;**

**public enum Permission {**

**WORKER\_SELF\_HOURS\_ADD("worker:addHours"),**

**WORKER\_SELF\_REPORT\_MAKE("worker:selfReport"),**

**DIRECTOR\_MAKE\_REPORT\_FOR\_ALL("director:allWorkerReport"),**

**DIRECTOR\_MAKE\_REPORT\_FOR\_WORKER("director:workerReport"),**

**DIRECTOR\_ADD\_HOURS\_TO\_WORKER("director:addHoursToWorker"),**

**DIRECTOR\_ADD\_NEW\_WORKER("director:addNewWorker"),**

**FREELANCER\_SELF\_HOURS\_ADD("freelancer:addHours"),**

**FREELANCER\_SELF\_REPORT\_MAKE("freelancer:selfReport");**

**private final String permission;**

**Permission(String permission) {**

**this.permission = permission;**

**}**

**public String getPermission() {**

**return permission;**

**}**

**}**

**Листинг 7. Класс сущности ролей**

**package com.SoftDevTimeSheet.entity;**

**import org.springframework.security.core.authority.SimpleGrantedAuthority;**

**import java.util.Set;**

**import java.util.stream.Collectors;**

**public enum Role {**

**WORKER(Set.of(Permission.WORKER\_SELF\_REPORT\_MAKE,Permission.WORKER\_SELF\_HOURS\_ADD)),**

**DIRECTOR(Set.of(Permission.DIRECTOR\_ADD\_HOURS\_TO\_WORKER, Permission.DIRECTOR\_MAKE\_REPORT\_FOR\_WORKER, Permission.DIRECTOR\_MAKE\_REPORT\_FOR\_ALL,Permission.DIRECTOR\_ADD\_NEW\_WORKER)),**

**FREELANCER(Set.of(Permission.FREELANCER\_SELF\_REPORT\_MAKE,Permission.FREELANCER\_SELF\_HOURS\_ADD));**

**Role(Set<Permission> permissions) {**

**this.permissions = permissions;**

**}**

**public Set<Permission> getPermissions() {**

**return permissions;**

**}**

**private final Set<Permission> permissions;**

**public Set<SimpleGrantedAuthority> getAuthorities(){**

**return getPermissions().stream().map(permission -> new SimpleGrantedAuthority(permission.getPermission()))**

**.collect(Collectors.toSet());**

**}**

**}**

**Листинг 8. Класс сущности рабочего дня**

**package com.SoftDevTimeSheet.entity;**

**import java.time.LocalDate;**

**//worked day entity**

**public class WorkedDay {**

**private String workerName;**

**private int hoursWorked;**

**private LocalDate workDate;**

**public String getWorkerName() {**

**return workerName;**

**}**

**public void setWorkerName(String workerName) {**

**this.workerName = workerName;**

**}**

**public int getHoursWorked() {**

**return hoursWorked;**

**}**

**public void setHoursWorked(int hoursWorked) {**

**this.hoursWorked = hoursWorked;**

**}**

**public LocalDate getWorkDate() {**

**return workDate;**

**}**

**public void setWorkDate(LocalDate workDate) {**

**this.workDate = workDate;**

**}**

**@Override**

**public String toString() {**

**return "workerName = " + workerName + " workDate=" + workDate + " hoursWorked= " + hoursWorked;**

**}**

**}**

**Листинг 9. Класс сущности работника**

**package com.SoftDevTimeSheet.entity;**

**import javax.persistence.\*;**

**//Worker entity**

**@Entity**

**@Table(name = "roles")**

**public class Worker {**

**@Id**

**@GeneratedValue(strategy = GenerationType.IDENTITY)**

**private Long id;**

**@Column(name = "name")**

**private String name;**

**@Column(name = "password")**

**private String password;**

**@Enumerated(value = EnumType.STRING)**

**@Column(name = "role")**

**private Role role;**

**public Worker() {**

**}**

**public Worker(Long id, String name, String password, Role role) {**

**this.id = id;**

**this.name = name;**

**this.password = password;**

**this.role = role;**

**}**

**public Long getId() {**

**return id;**

**}**

**public void setId(Long id) {**

**this.id = id;**

**}**

**public String getName() {**

**return name;**

**}**

**public void setName(String name) {**

**this.name = name;**

**}**

**public String getPassword() {**

**return password;**

**}**

**public void setPassword(String password) {**

**this.password = password;**

**}**

**public Role getRole() {**

**return role;**

**}**

**public void setRole(Role role) {**

**this.role = role;**

**}**

**}**

**Листинг 10. Класс парсер строки в период времени**

**package com.SoftDevTimeSheet.parsers;**

**import com.SoftDevTimeSheet.dao.ReportDaysValue;**

**//browser String to days report value parser**

**public class StringToReportDaysValueParser {**

**public static ReportDaysValue parseStingToDaysValue(String reportDaysStringValue) {**

**if(reportDaysStringValue.toUpperCase().equals("MONTH")){**

**return ReportDaysValue.MONTH;**

**}else if(reportDaysStringValue.toUpperCase().equals("WEEK")){**

**return ReportDaysValue.WEEK;**

**}else if(reportDaysStringValue.toUpperCase().equals("DAY")){**

**return ReportDaysValue.DAY;**

**}**

**return ReportDaysValue.DAY;//default value**

**}**

**}**

**Листинг 11. класс-интерфейс для работы с получением авторизации работника**

**package com.SoftDevTimeSheet.repository;**

**import com.SoftDevTimeSheet.entity.Worker;**

**import org.springframework.data.jpa.repository.JpaRepository;**

**import java.util.Optional;**

**public interface UserRepository extends JpaRepository<Worker, Long> {**

**Optional<Worker> findByName(String name);**

**}**

**Листинг 12. Главный сервис работы логики**

**package com.SoftDevTimeSheet.restService;**

**import com.SoftDevTimeSheet.dao.DAOInterfaces.IWorkerDAO;**

**import com.SoftDevTimeSheet.dao.ReportDaysValue;**

**import com.SoftDevTimeSheet.dao.WorkerDAO;**

**import com.SoftDevTimeSheet.entity.WorkedDay;**

**import com.SoftDevTimeSheet.parsers.StringToReportDaysValueParser;**

**import org.springframework.security.core.Authentication;**

**import org.springframework.security.core.context.SecurityContextHolder;**

**import java.time.LocalDate;**

**import java.time.format.DateTimeFormatter;**

**import java.util.List;**

**import java.util.Locale;**

**public class TimesheetService {**

**IWorkerDAO dao = new WorkerDAO();**

**//director make report for all workers by date**

**public List<WorkedDay> directorMakeReportForAllWorkers(String date, String reportDays){**

**//string date to local date**

**DateTimeFormatter dtf = DateTimeFormatter.ofPattern("dd-MM-yyyy");**

**LocalDate localDate = LocalDate.parse(date, dtf);**

**//string report days value to report days value**

**ReportDaysValue reportDaysValue = StringToReportDaysValueParser.parseStingToDaysValue(reportDays);**

**return dao.getReportForAllWorkers(localDate,reportDaysValue);**

**}**

**//director make report for worker by name**

**public List<WorkedDay> directorMakeReportByWorkerName(String workerName, String stringDate, String stringReportDaysValue){**

**//string date to local date**

**DateTimeFormatter dtf = DateTimeFormatter.ofPattern("dd-MM-yyyy");**

**LocalDate date = LocalDate.parse(stringDate,dtf);**

**//string report days value to report days value**

**ReportDaysValue reportDaysValue = StringToReportDaysValueParser.parseStingToDaysValue(stringReportDaysValue);**

**List<WorkedDay> reportByName = dao.getReportForWorkerByName(workerName,date,reportDaysValue);**

**if(!reportByName.isEmpty()){**

**return reportByName;**

**}**

**return reportByName;**

**}**

**//director add hours to worker by name**

**public boolean directorAddHoursToWorker(String workerName, String stringDate, String stringHoursToAdd){**

**//string date to local date**

**DateTimeFormatter dtf = DateTimeFormatter.ofPattern("dd-MM-yyyy");**

**LocalDate date = LocalDate.parse(stringDate,dtf);**

**//string hours to int hours**

**int hoursToAdd = Integer.parseInt(stringHoursToAdd);**

**if(dao.addHoursToWorker(workerName,date,hoursToAdd)){**

**return true;**

**}else{**

**return false;**

**}**

**}**

**public boolean directorAddNewWorker(String workerName, String password,String role){**

**role = role.toUpperCase(Locale.ROOT);**

**if(dao.addNewWorkerWithRole(workerName,password,role)){**

**return true;**

**}else {**

**return false;**

**}**

**}**

**public boolean workerAddSelfHours(String stringDate, String stringHoursToAdd){**

**String workerName = getCurrentUsername();**

**//string date to local date**

**DateTimeFormatter dtf = DateTimeFormatter.ofPattern("dd-MM-yyyy");**

**LocalDate date = LocalDate.parse(stringDate,dtf);**

**//string hours to int hours**

**int hoursToAdd = Integer.parseInt(stringHoursToAdd);**

**if(dao.addHoursToWorker(workerName,date,hoursToAdd)){**

**return true;**

**}else{**

**return false;**

**}**

**}**

**public List<WorkedDay> workerMakeSelfReport(String stringDate, String stringReportDaysValue){**

**String workerName = getCurrentUsername();**

**//string date to local date**

**DateTimeFormatter dtf = DateTimeFormatter.ofPattern("dd-MM-yyyy");**

**LocalDate date = LocalDate.parse(stringDate,dtf);**

**//string report days value to report days value**

**ReportDaysValue reportDaysValue = StringToReportDaysValueParser.parseStingToDaysValue(stringReportDaysValue);**

**List<WorkedDay> reportByName = dao.getReportForWorkerByName(workerName,date,reportDaysValue);**

**if(!reportByName.isEmpty()){**

**return reportByName;**

**}**

**return reportByName;**

**}**

**public boolean freelancerAddSelfHours(String stringDate, String stringHoursToAdd){**

**String workerName = getCurrentUsername();**

**//string date to local date**

**DateTimeFormatter dtf = DateTimeFormatter.ofPattern("dd-MM-yyyy");**

**LocalDate date = LocalDate.parse(stringDate,dtf);**

**LocalDate nowLocalDate = LocalDate.now();**

**//string hours to int hours**

**int hoursToAdd = Integer.parseInt(stringHoursToAdd);**

**if(date.equals(nowLocalDate) || date.equals(nowLocalDate.minusDays(1)) || date.equals(nowLocalDate.minusDays(2))){**

**//well date**

**if(dao.addHoursToWorker(workerName,date,hoursToAdd)){**

**return true;**

**}else {**

**return false;**

**}**

**}else {**

**//if date longer that 2 days from now**

**return false;**

**}**

**}**

**public List<WorkedDay> freelanerMakeSelfReport(String stringDate, String stringReportDaysValue){**

**String workerName = getCurrentUsername();**

**//string date to local date**

**DateTimeFormatter dtf = DateTimeFormatter.ofPattern("dd-MM-yyyy");**

**LocalDate date = LocalDate.parse(stringDate,dtf);**

**//string report days value to report days value**

**ReportDaysValue reportDaysValue = StringToReportDaysValueParser.parseStingToDaysValue(stringReportDaysValue);**

**List<WorkedDay> reportByName = dao.getReportForWorkerByName(workerName,date,reportDaysValue);**

**if(!reportByName.isEmpty()){**

**return reportByName;**

**}**

**return reportByName;**

**}**

**private String getCurrentUsername() {**

**Authentication auth = SecurityContextHolder.getContext().getAuthentication();**

**return auth.getName();**

**}**

**}**

**Листинг 13. класс защищенного пользователя**

**package com.SoftDevTimeSheet.security;**

**import com.SoftDevTimeSheet.entity.Worker;**

**import lombok.Data;**

**import org.springframework.security.core.GrantedAuthority;**

**import org.springframework.security.core.authority.SimpleGrantedAuthority;**

**import org.springframework.security.core.userdetails.UserDetails;**

**import java.util.Collection;**

**import java.util.List;**

**@Data**

**public class SecurityUser implements UserDetails {**

**private final String name;**

**private final String password;**

**private final List<SimpleGrantedAuthority> authorities;**

**private final boolean isActive;**

**@Override**

**public Collection<? extends GrantedAuthority> getAuthorities() {**

**return authorities;**

**}**

**@Override**

**public String getPassword() {**

**return password;**

**}**

**@Override**

**public String getUsername() {**

**return name;**

**}**

**@Override**

**public boolean isAccountNonExpired() {**

**return isActive;**

**}**

**@Override**

**public boolean isAccountNonLocked() {**

**return isActive;**

**}**

**@Override**

**public boolean isCredentialsNonExpired() {**

**return isActive;**

**}**

**@Override**

**public boolean isEnabled() {**

**return isActive;**

**}**

**public static UserDetails fromUser(Worker worker){**

**return new org.springframework.security.core.userdetails.User(**

**worker.getName(), worker.getPassword(),worker.getRole().getAuthorities()**

**);**

**}**

**}**

**Листинг 14. класс получения авторизованного пользователя**

**package com.SoftDevTimeSheet.security;**

**import com.SoftDevTimeSheet.entity.Worker;**

**import com.SoftDevTimeSheet.repository.UserRepository;**

**import org.springframework.beans.factory.annotation.Autowired;**

**import org.springframework.security.core.userdetails.UserDetails;**

**import org.springframework.security.core.userdetails.UserDetailsService;**

**import org.springframework.security.core.userdetails.UsernameNotFoundException;**

**import org.springframework.stereotype.Service;**

**@Service("userDetailsServiceImpl")**

**public class UserDetailsServiceImpl implements UserDetailsService {**

**private final UserRepository userRepository;**

**@Autowired**

**public UserDetailsServiceImpl(UserRepository userRepository) {**

**this.userRepository = userRepository;**

**}**

**@Override**

**public UserDetails loadUserByUsername(String name) throws UsernameNotFoundException {**

**Worker worker = userRepository.findByName(name).orElseThrow(() -> new UsernameNotFoundException("User doesn't exist"));**

**return SecurityUser.fromUser(worker);**

**}**

**}**

**Листинг 15. класс контроллер авторизации**

**package com.SoftDevTimeSheet.controllers;**

**import org.springframework.stereotype.Controller;**

**import org.springframework.web.bind.annotation.GetMapping;**

**import org.springframework.web.bind.annotation.RequestMapping;**

**@Controller**

**@RequestMapping("/auth")**

**public class AuthController {**

**@GetMapping("/login")**

**public String getLoginPage(){**

**return "login";**

**}**

**@GetMapping("success")**

**public String getSuccessPage(){**

**return "success";**

**}**

**}**

**Листинг 16. Класс контроллер для пользователя директор**

**package com.SoftDevTimeSheet.controllers;**

**import com.SoftDevTimeSheet.entity.WorkedDay;**

**import com.SoftDevTimeSheet.restService.TimesheetService;**

**import org.springframework.security.access.prepost.PreAuthorize;**

**import org.springframework.web.bind.annotation.GetMapping;**

**import org.springframework.web.bind.annotation.RequestMapping;**

**import org.springframework.web.bind.annotation.RequestParam;**

**import org.springframework.web.bind.annotation.RestController;**

**import java.util.List;**

**@RestController**

**@RequestMapping("/Director")**

**public class DirectorController {**

**@PreAuthorize("hasAuthority('director:allWorkerReport')")**

**@GetMapping("/makeReportForAll")**

**public List<WorkedDay> allWorkersReport(@RequestParam(value = "date", defaultValue = "03-01-2021")String date,**

**@RequestParam(value = "daysValue", defaultValue = "week") String reportDays){**

**TimesheetService timesheetService = new TimesheetService();**

**return timesheetService.directorMakeReportForAllWorkers(date,reportDays);**

**}**

**@PreAuthorize("hasAuthority('director:addHoursToWorker')")**

**@GetMapping("/addHoursToWorker")**

**public boolean addHours(@RequestParam(value = "name",defaultValue = "default") String name,**

**@RequestParam(value = "date",defaultValue = "01-01-2001") String date,**

**@RequestParam(value = "hours",defaultValue = "88") String hours){**

**TimesheetService timesheetService = new TimesheetService();**

**return timesheetService.directorAddHoursToWorker(name, date, hours);**

**}**

**@PreAuthorize("hasAuthority('director:workerReport')")**

**@GetMapping("/makeReportForWorker")**

**public List<WorkedDay> allWorkersReport(@RequestParam(value = "name",defaultValue = "kostya")String name,**

**@RequestParam(value = "date", defaultValue = "03-01-2021")String date,**

**@RequestParam(value = "daysValue", defaultValue = "week") String reportDays){**

**TimesheetService timesheetService = new TimesheetService();**

**return timesheetService.directorMakeReportByWorkerName(name, date, reportDays);**

**}**

**@PreAuthorize("hasAuthority('director:addNewWorker')")**

**@GetMapping("/addNewWorker")**

**public boolean addNewWorker(@RequestParam(value = "name",defaultValue = "default") String name,**

**@RequestParam(value = "password",defaultValue = "default") String password,**

**@RequestParam(value = "role",defaultValue = "worker") String role){**

**TimesheetService timesheetService = new TimesheetService();**

**return timesheetService.directorAddNewWorker(name,password,role);**

**}**

**}**

**Листинг 17. класс контроллер для работника с ролью фрилансер**

**package com.SoftDevTimeSheet.controllers;**

**import com.SoftDevTimeSheet.entity.WorkedDay;**

**import com.SoftDevTimeSheet.restService.TimesheetService;**

**import org.springframework.security.access.prepost.PreAuthorize;**

**import org.springframework.web.bind.annotation.GetMapping;**

**import org.springframework.web.bind.annotation.RequestMapping;**

**import org.springframework.web.bind.annotation.RequestParam;**

**import org.springframework.web.bind.annotation.RestController;**

**import java.util.List;**

**@RestController**

**@RequestMapping("/Freelancer")**

**public class FreelancerController {**

**@PreAuthorize("hasAuthority('freelancer:selfReport')")**

**@GetMapping("/selfReport")**

**public List<WorkedDay> makeSelfReport(@RequestParam(value = "date", defaultValue = "03-01-2021") String date,**

**@RequestParam(value = "daysValue", defaultValue = "week") String reportDays){**

**TimesheetService timesheetService = new TimesheetService();**

**return timesheetService.freelanerMakeSelfReport(date,reportDays);**

**}**

**@PreAuthorize("hasAuthority('freelancer:addHours')")**

**@GetMapping("/selfHoursAdd")**

**public boolean addHours(@RequestParam(value = "date", defaultValue = "16-05-2021") String date,**

**@RequestParam(value = "hours",defaultValue = "88") String hours){**

**TimesheetService timesheetService = new TimesheetService();**

**return timesheetService.freelancerAddSelfHours(date,hours);**

**}**

**}**

**Листинг 18 класс контролер для штатного работника**

**package com.SoftDevTimeSheet.controllers;**

**import com.SoftDevTimeSheet.dao.DAOInterfaces.IWorkerDAO;**

**import com.SoftDevTimeSheet.dao.WorkerDAO;**

**import com.SoftDevTimeSheet.entity.WorkedDay;**

**import com.SoftDevTimeSheet.restService.TimesheetService;**

**import org.springframework.security.access.prepost.PreAuthorize;**

**import org.springframework.web.bind.annotation.GetMapping;**

**import org.springframework.web.bind.annotation.RequestMapping;**

**import org.springframework.web.bind.annotation.RequestParam;**

**import org.springframework.web.bind.annotation.RestController;**

**import java.util.List;**

**@RestController**

**@RequestMapping("Worker")**

**public class WorkerController {**

**@PreAuthorize("hasAuthority('worker:selfReport')")**

**@GetMapping("/selfReport")**

**public List<WorkedDay> makeSelfReport(@RequestParam(value = "date", defaultValue = "03-01-2021") String date,**

**@RequestParam(value = "daysValue", defaultValue = "week") String reportDays){**

**TimesheetService timesheetService = new TimesheetService();**

**return timesheetService.workerMakeSelfReport(date,reportDays);**

**}**

**@PreAuthorize("hasAuthority('worker:addHours')")**

**@GetMapping("/selfHoursAdd")**

**public boolean addHours(@RequestParam(value = "date", defaultValue = "03-01-2021") String date,**

**@RequestParam(value = "hours",defaultValue = "88") String hours){**

**TimesheetService timesheetService = new TimesheetService();**

**return timesheetService.workerAddSelfHours(date,hours);**

**}**

## **Описание спецификаций к модулям**

В данном разделе описаны публичные члены модулей проекта.

1. app – объект, который создает системное окно и объект самого приложения.
2. UserRepository – объект интерфейса для обращений к БД.
3. ui – объект который представляет собой сам интерфейс.
4. Permission – сущность прав.
5. Role – сущность роли.
6. WorkedDay – сущность рабочего дня.
7. Worker – сущность работника.
8. SecurityUser – пользователь из Spring.security
9. DAO package - классы для доступа к БД.
10. Controller package – классы контроллеров.
11. Service – классы бизнес логики

## 

## **Описание модулей**

Кроме главного модуля программа содержит еще один модуль который реализует интерфейс тесирования.

import org.junit.jupiter.api.Test;

import org.springframework.boot.test.context.SpringBootTest;

@SpringBootTest

class SoftDevTimeSheetApplicationTests {

@Test

void contextLoads() {

}

}

Который отвечает за проверку удачной загрузки контекста.

## **Описание применения средств отладки**

В этом разделе показано умение применять средства отладки.

В ходе написания проекта при попытке запустить jvm было получено данное сообщение:

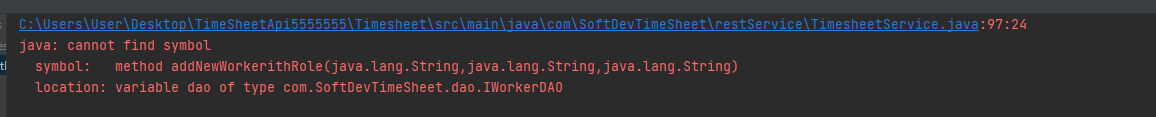


Рисунок 11. До применения средств отладки

После получения данного сообщения были просмотрены 97:24 строки модуля и была обнаружена ошибка, которая впоследствии была устранена, а после попытки запуска скрипта получено данное сообщение:

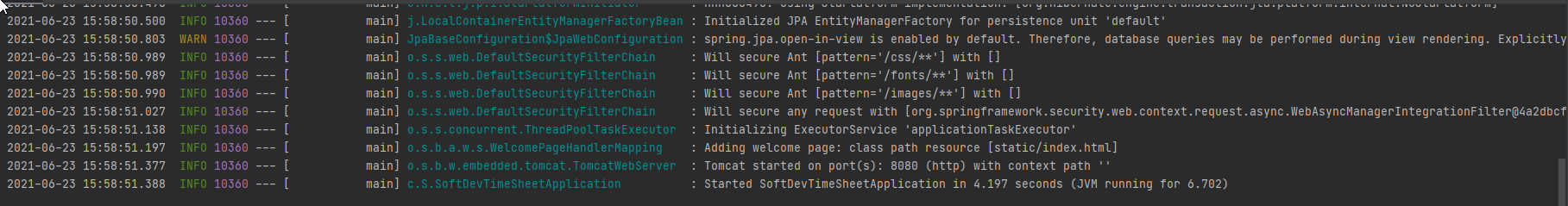


Рисунок 12. После применения средств отладки

Это означает что ошибка была устранена и jvm запустился.

# **Эксплуатационная часть**

## **Руководство оператора**

**АННОТАЦИЯ**

Функционал роли руководитель: (mapping localhost:8080/Director/ Может добавлять новых сотрудников, (/addNewWorker с параметрами name, passowrd, role

Может добавлять сотрудникам отработанные часы (/addHoursToWorkker с параметрами name, date, hours)

Может получать отчет по отработанным часам по датам для определенного сотрудника (/makeReportForWorker с параметрами name, date, daysValue может быть day,week,month)

Может получать отчет по отработанным часам по датам для всех сотрудников (/makeReportForAll с параметрами начала date , и интервала, который может быть day,week,month)

Может удалять работника (/removeWorker с парамтрами name)

Функционал роли штатный работник: (mapping localhost:8080/Worker/

Может добавлять отработанные часы только за себя (/selfHoursAdd с параметрами date, и hours)

Может получать отчет только за себя (/selfReport с параметрами date, daysValue может быть day,week,month) Функционал роли фрилансер(mapping localhost:8080/Freelance

Может добавлять часы себе, только если от текущей даты прошло не более двух дней(/selfHoursAdd с параметрами date, hours)

Может получать отчет только за себя (/selfReport с параметрами date, daysValue может быть day,week,month).



Рисунок.13 Авторизация

Кнопки меню имеют следующие функции:

Tables – открывает окно для работы с таблицами.

Exit – Выход из программы.

* 1. **Выполнение программы**
     1. **Выполнение функции открытия окна таблиц.**

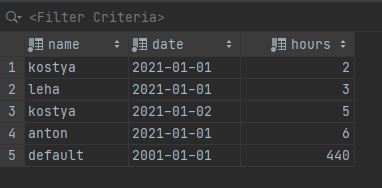


Рисунок 15. Окно для работы с таблицами

* + 1. **Выполнение функции смены таблицы.**

Нажмите на поле находящееся выше таблицы. Выберите новую таблицу, в данном случае таблицу «Guests».

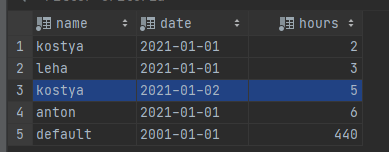


Рисунок 16. Выбор таблицы

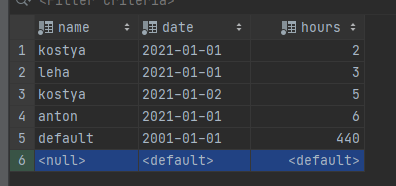


Рисунок 16. Новая таблица

* + 1. **Выполнение функции добавления элемента в таблицу.**

При выполнении добавления нового работника, будет добавлена запись в БД.

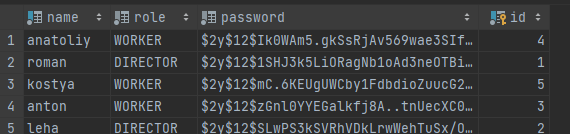


Рисунок 17. Таблица работников

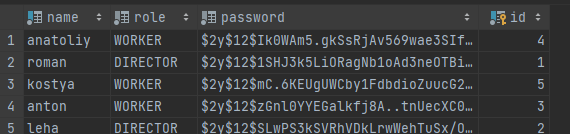


Рисунок18. Таблица работников

* + 1. **Выполнение функции удаления элемента из таблицы.**

При получении запроса на удаление будет удалена запись

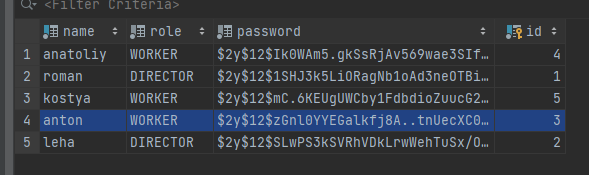


Рисунок 19. Элемент для удаления

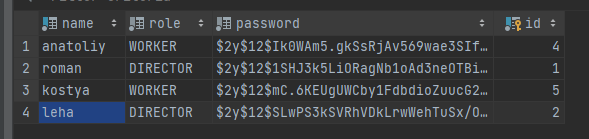


Рисунок 20. Таблица после удаления элемента

# **Заключение**

В результате выполнения проекта была написана программа «timesheet rest api» для упрощения ведения бизнеса в области отчетов, а именно для отслеживания отработанных часов работника.

В ходе работы были проанализированы предметная область, существующие разработки, посвященные данному направлению, получены практические навыки по созданию REST при помощи Java/spring

Также планируется продолжать работу над данным проектом с целью расширения возможностей и удобства приложения для пользователей. Планы по доработкам представлены ниже.

To-do лист:

1. Доработка всех пунктов главного меню,
2. Добавления расчета зарплаты.
3. Доработка интерфейса с целью упрощения работы с программой.
4. Перенос всех данный в БД.
5. Улучшение дизайна интерфейса.

# **Дневник практики**

Таблица 1. Дневник практики.

|  |  |  |  |
| --- | --- | --- | --- |
| Дата | Содержание работы | Отметка о выполнении  работы | Подпись руководителя практики |
| 26.02 – 31.02 | Составление плана разработки |  |  |
| 2.03 – 13.03 | Выбор технологий |  |  |
| 25.03 – 2.04 | Разработка программного содержания. |  |  |
| 12.04 – 15.04 | Тестирование |  |  |
| 15.04 – 23.04 | Оформление отчёта. |  |  |

# **Список литературы и интернет-источников**

1. Гагарина, Л. Г. Технология разработки программного обеспечения: учеб. пособие / Л.Г. Гагарина, Е.В. Кокорева, Б.Д. Сидорова-Виснадул; под ред. Л.Г. Гагариной. — Москва: ИД «ФОРУМ»: ИНФРА-М, 2019. — 400 с. — (Высшее образование: Бакалавриат). - ISBN 978-5-16-104071-3. - Текст: электронный. - URL:

**Статья про отчеты**

1. <https://ru.wiktionary.org/wiki/%D0%BE%D1%82%D1%87%D1%91%D1%82>

Официальная документация Spring

1. [**https://spring.io/**](https://spring.io/)

Документация по Java

1. **https://metanit.com/java/**