

ПРИМЕРНОЕ СОДЕРЖАНИЕ ОТЧЕТА ПО ЛАБОРАТОРНОЙ РАБОТЕ №6

1. АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

1.1 Описание предметной области:

Предметная область: Задача найма сотрудников в компанию.

Компания — представлена перечнем отделов с возможностью просмотра средней заработной платы в отделе и сотрудниками, работающими в определенных отделах.

Сотрудник — представлен табельным номером, именем, отчеством, фамилией, годом рождения, электронной почтой, заработной платой, полом и наименованием отдела в котором сотрудник работает.

Требуемая функциональность программы:

- «Инициализация системы» — запуск программы. Список записей хранится в памяти компьютера в БД;
- «Авторизация» — возможность авторизации администратора.
- «Создание» — создание новых отделов, добавление сотрудников.
- «Просмотр» — просмотр списка записей, средней заработной платы в отделе;
- «Редактирование» — редактирование сведений об отделах, сотрудниках;
- «Удаление» — возможность расформирования отдела, увольнения сотрудников;
- «Выход» — завершение работы программы.

Цель и назначение разработки — упрощение процесса управления компанией и её отделами, найма сотрудников.

1.2 Варианты использования программы в виде диаграмм прецедентов:

Первичное описание прецедентов:

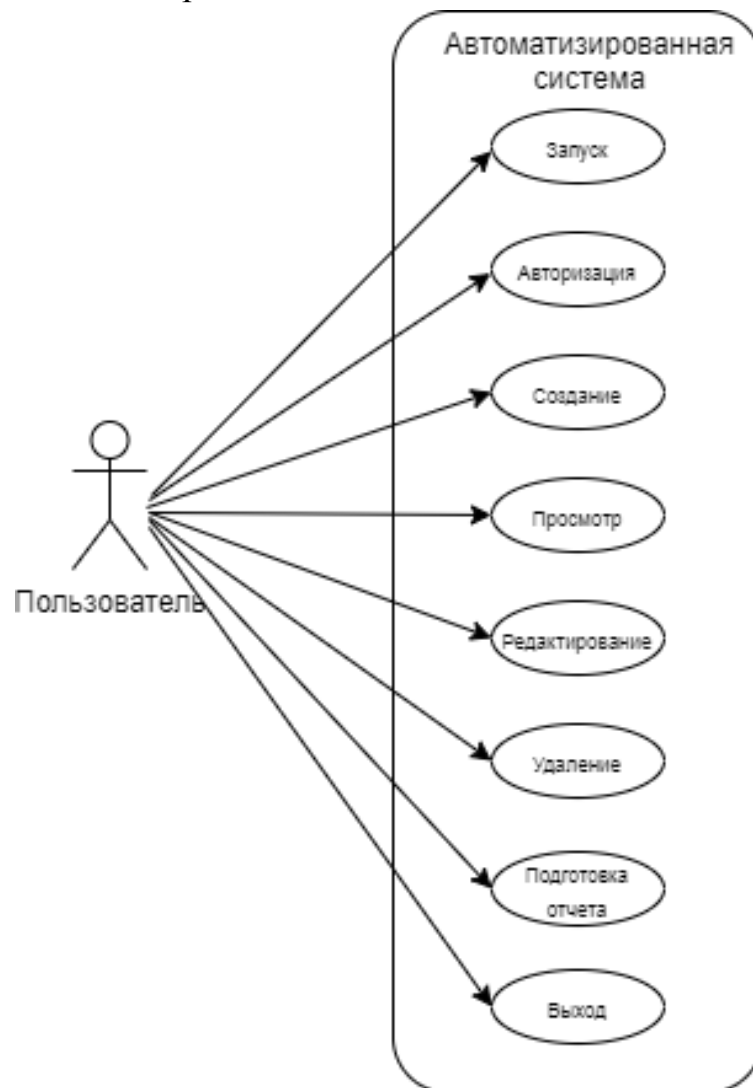


Рисунок 1.1 Первичное описание прецедентов

Описание прецедентов:

Прецедент №1 «Запуск»

Назначение: инициализация системы, визуализация приложения.

Исполнители: пользователь, система.

Предусловие: запуск программы пользователем.

Постусловие: выполняется действие в зависимости от нужд пользователя.

Основной поток событий: Происходит инициализация приложения.

В случае успешной визуализации пользователь продолжает работу с системой, иначе выполняется АПС.

Альтернативный поток событий: Аварийное завершение работы приложения.

Прецедент №2 «Авторизация»

Назначение: авторизация администратора.

Исполнители: пользователь, система.

Предусловие: система инициализирована, выбрана авторизация.

Постусловие: возможность дальнейшей работы с приложением.

Основной поток событий: Пользователь вводит логин и пароль. В случае успешной авторизации пользователь получает доступ к приложению, иначе выполняется АПС.

Альтернативный поток событий: Пользователь получает сообщение об ошибке авторизации.

Прецедент №3 «Создание»

Назначение: создание новой записи.

Исполнители: пользователь, система.

Предусловие: система инициализирована, пользователь авторизован.

Постусловие: пользователь создал запись.

Основной поток событий: В случае успешной авторизации пользователь получает доступ к созданию записей, иначе выполняется АПС.

Альтернативный поток событий: Пользователь получает сообщение об ошибке авторизации.

Прецедент №4 «Просмотр»

Назначение: просмотр необходимой информации.

Исполнители: пользователь, система.

Предусловие: система инициализирована, пользователь авторизован.

Постусловие: пользователь просмотрел информацию.

Основной поток событий: В случае успешной авторизации пользователь получает доступ к приложению, иначе выполняется АПС.

Альтернативный поток событий: Пользователь получает сообщение об ошибке авторизации.

Прецедент №5 «Редактирование»

Назначение: редактирование необходимой информации.

Исполнители: пользователь, система.

Предусловие: система инициализирована, пользователь авторизован

Постусловие: запись отредактирована.

Основной поток событий: Пользователь изменяет, сохраняет нужную запись, иначе переходим к АПС.

Альтернативный поток событий: Пользователь получает сообщение об ошибке редактирования.

Прецедент №6 «Удаление»

Назначение: удаление необходимой информации.

Исполнители: пользователь, система.

Предусловие: система инициализирована, пользователь авторизован

Постусловие: запись удалена.

Основной поток событий: Пользователь удаляет нужную запись, иначе переходим к АПС.

Альтернативный поток событий: Пользователь получает сообщение об ошибке удаления.

Прецедент №7 «Формирование отчета» //не выполняем

Назначение: генерация отчета.

Исполнители: пользователь, система.

Предусловие: система инициализирована.

Постусловие: отчет создан.

Основной поток событий: Пользователь выбирает куда сохранить отчет, иначе переходим к АПС.

Альтернативный поток событий: Пользователь получает сообщение об ошибке создания отчета.

Прецедент №8 «Выход»

Назначение: выход из программы.

Исполнители: пользователь, система.

Предусловие: нажатие пользователем кнопки, завершающей работу приложения.

Постусловие: работа с программой завершена.

Основной поток событий: Происходит завершение работы с приложением, иначе, в случае «зависания» приложения, выполняется АПС.

Альтернативный поток событий: Аварийное завершение работы приложения.

Уточненная диаграмма прецедентов:

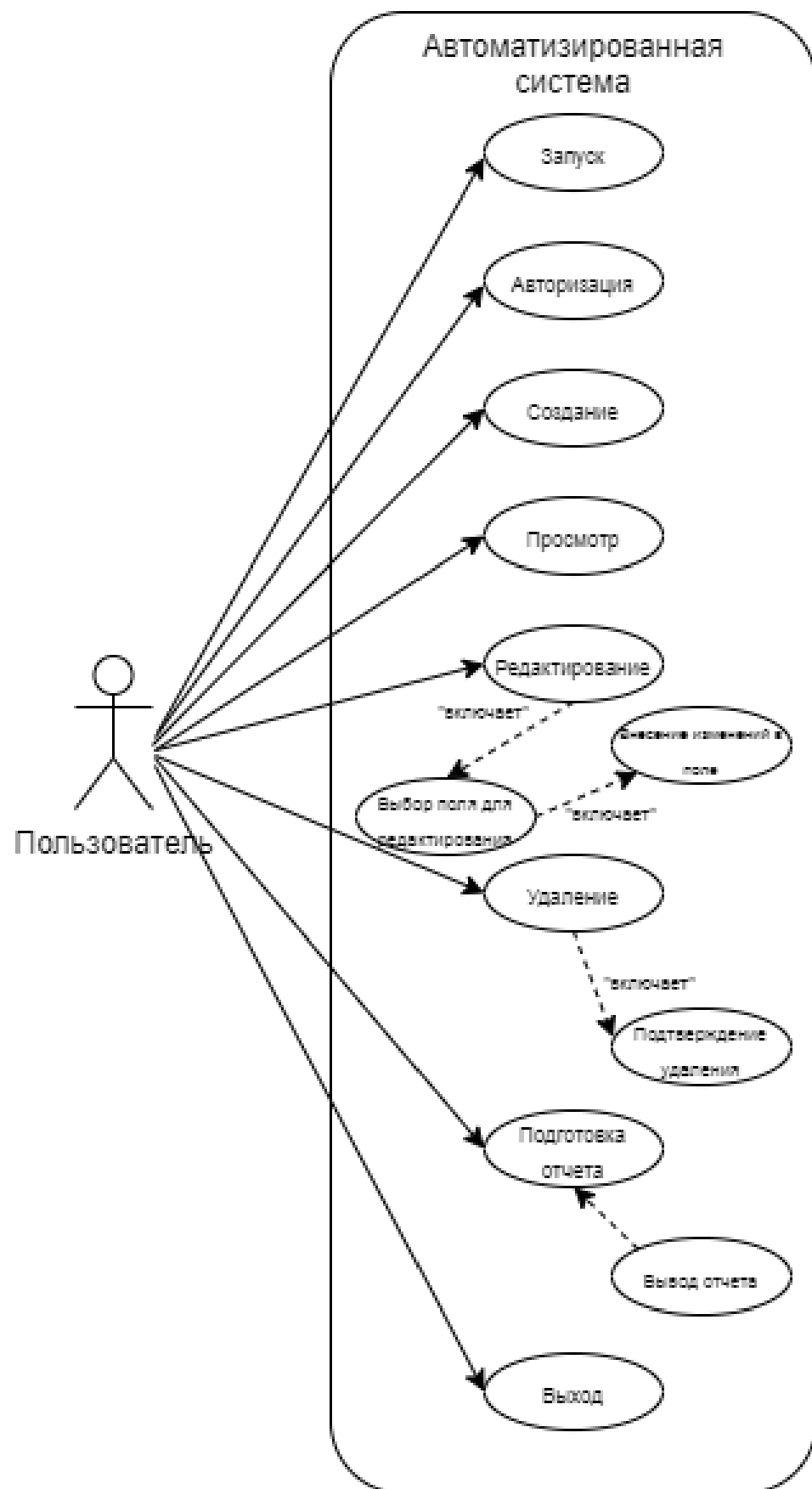


Рисунок 1.2 Уточненная диаграмма прецедентов

1.3 Первичное описание объектов и классов, прецедентов системы:

Описание классов:

Отдел — класс, хранящий сведения об отделе.

Свойства:

- **Идентификатор** — идентификационный номер отдела.
- **Название** — название отдела.
- **Сотрудники** — перечень сотрудников отдела.
- **Средняя заработная плата** — средняя заработная плата в отделе.

Сотрудник — класс, хранящий сведения о сотруднике.

Свойства:

- **Идентификатор** — идентификатор сотрудника.
- **Имя** — имя сотрудника.
- **Фамилия** — фамилия сотрудника.
- **Дата рождения** — дата рождения сотрудника.
- **Почта** — почта для связи с сотрудником.
- **Зарботная плата** — заработная плата сотрудника.
- **Пол** — пол сотрудника.
- **Отдел** — отдел, в котором работает сотрудник.

Диаграммы классов:

Отдел	Сотрудник
- идентификатор - название - сотрудники - средняя заработная плата	- идентификатор - имя - фамилия - дата рождения - почта - заработная плата - пол - отдел
+ изменить данные об отделе()	+ изменить данные о сотруднике()
Хранит сведения об отделах	Хранит сведения о сотрудниках

1.4 Первоначальное описание отношений между классами:

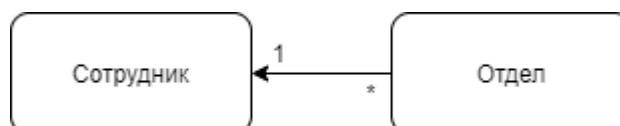


Рисунок 1.4 - Диаграмма отношений на уровне ассоциаций.

1.5 Диаграммы состояний для прецедентов:

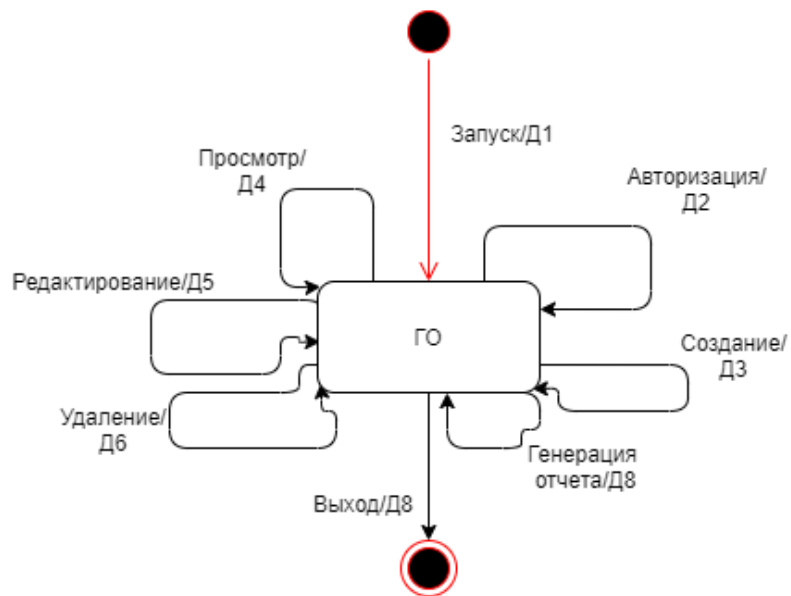


Рисунок 1.5 - Диаграмма состояний для ГО приложения

Д1 — инициализация и визуализация приложения.

Д2 — авторизация администратора.

Д3 — создание записи.

Д4 — просмотр записей.

Д5 — редактирование записей.

Д6 — удаление записей.

Д7 — генерация отчета.

Д8 — завершения работы приложения.

2. ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ ПРОЕКТИРОВАНИЕ

2.1 Диаграммы классов предметной области:

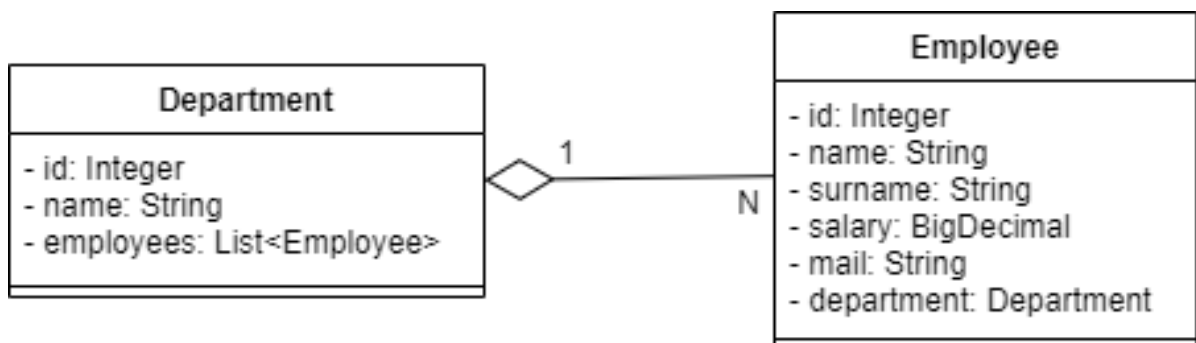


Рисунок 2.1 - Диаграмма классов предметной области

2.2 Графический интерфейс приложения:

1. Просмотр списка отделов со средней зарплатой в каждом отделе:

Departments Employees

DEPARTMENTS +Add

Title	Avg salary	
Development	500.0	Edit Delete
Production	700.0	Edit Delete

Рисунок 2.2 - Графический интерфейс просмотра списка отделов

2. Добавление отдела:

Departments Employees

← NEW DEPARTMENT

Department name

Development

Cancel Save

Рисунок 2.3 - Графический интерфейс добавления отдела

3. Обновление отдела:

Departments Employees

← EDIT DEPARTMENT

Department name

Development

Cancel Save

Рисунок 2.4 - Графический интерфейс обновления отдела

4. Удаление отдела:

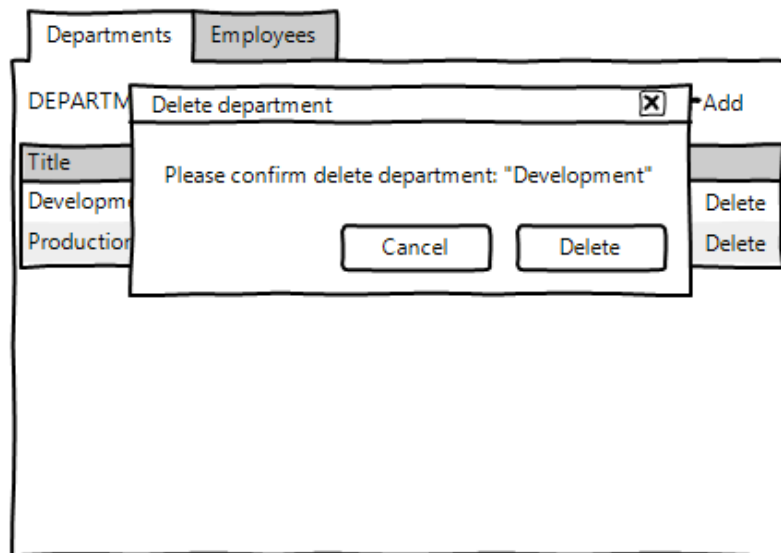


Рисунок 2.5 - Графический интерфейс удаления отдела

5. Добавления сотрудника в отдел:

The mockup shows a web interface with two tabs: 'Departments' and 'Employees'. The 'Employees' tab is active, displaying a form titled 'NEW EMPLOYEE'. The form has four input fields: 'Department' (a dropdown menu with 'Development' selected), 'Firstname' (containing 'Vasilii'), 'Lastname' (containing 'Vasiliev'), and 'Salary' (containing '500.0'). At the bottom right of the form are two buttons: 'Cancel' and 'Save'.

Рисунок 2.6 - Графический интерфейс добавления сотрудника в отдел

6. Обновление данных о сотруднике:

Рисунок 2.7 - Графический интерфейс обновления данных о сотруднике

7. Удаление сотрудника:

Рисунок 2.8 - Графический интерфейс удаление сотрудника

2.3 Общая диаграмма с учетом каркаса.

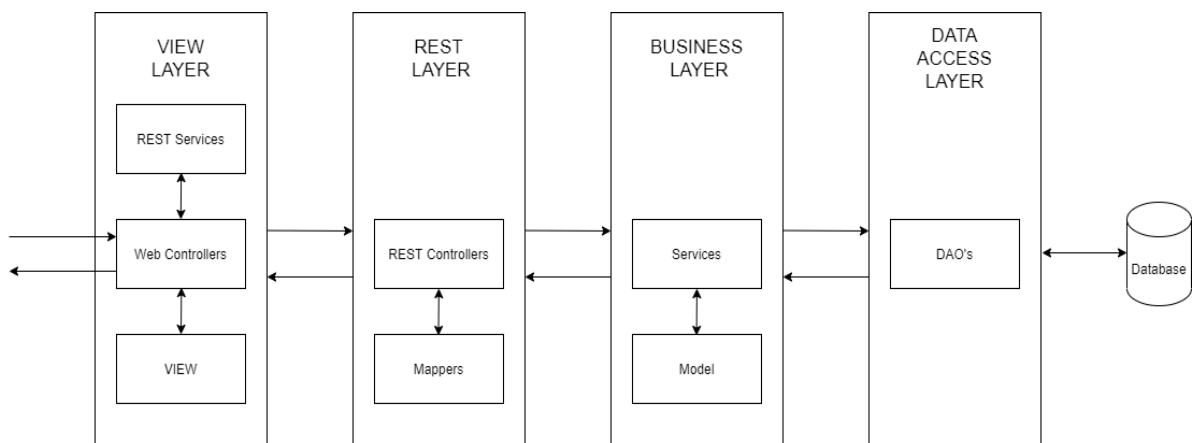


Рисунок 2.9 - Общая диаграмма с учетом каркаса

2.4 Диаграммы последовательностей.

Прецедент №2 «Авторизация»:

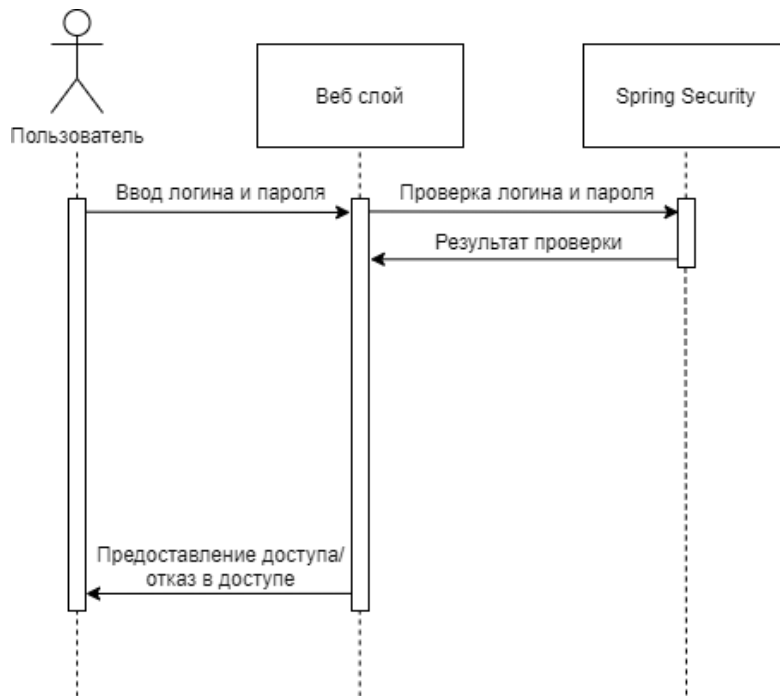


Рисунок 2.10 - Диаграмма последовательностей авторизации

Прецедент №4 «Просмотр»:

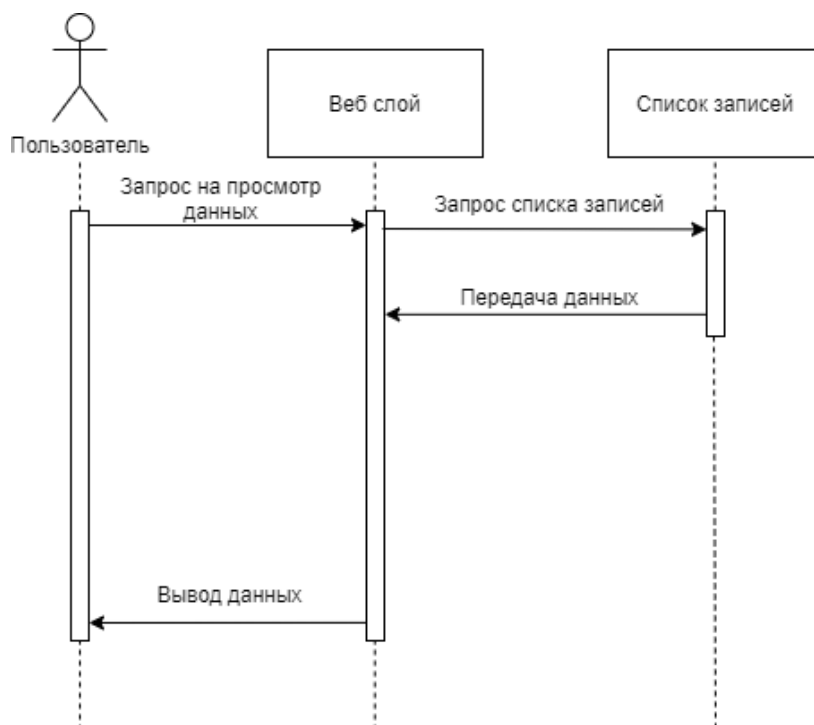


Рисунок 2.11 - Диаграмма последовательностей поиска

Прецедент №7 «Формирование отчета»: не выполняем

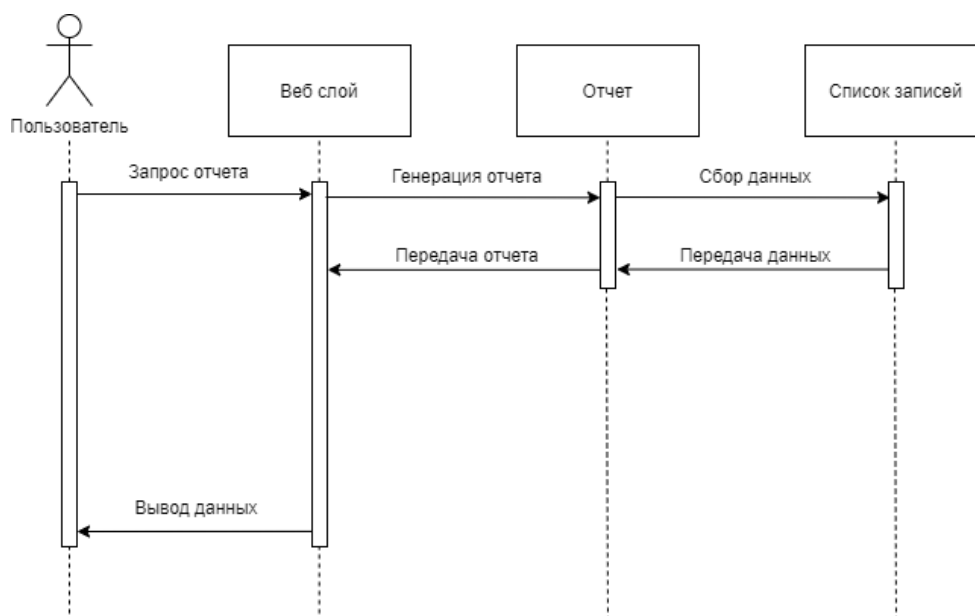


Рисунок 2.12 - Диаграмма последовательностей формирования отчета

2.5 Диаграммы видов деятельности

Прецедент №7 «Формирование отчета»:



Рисунок 2.13 - Диаграмма видов деятельности формирования отчета

Прецедент №2 «Авторизация»:

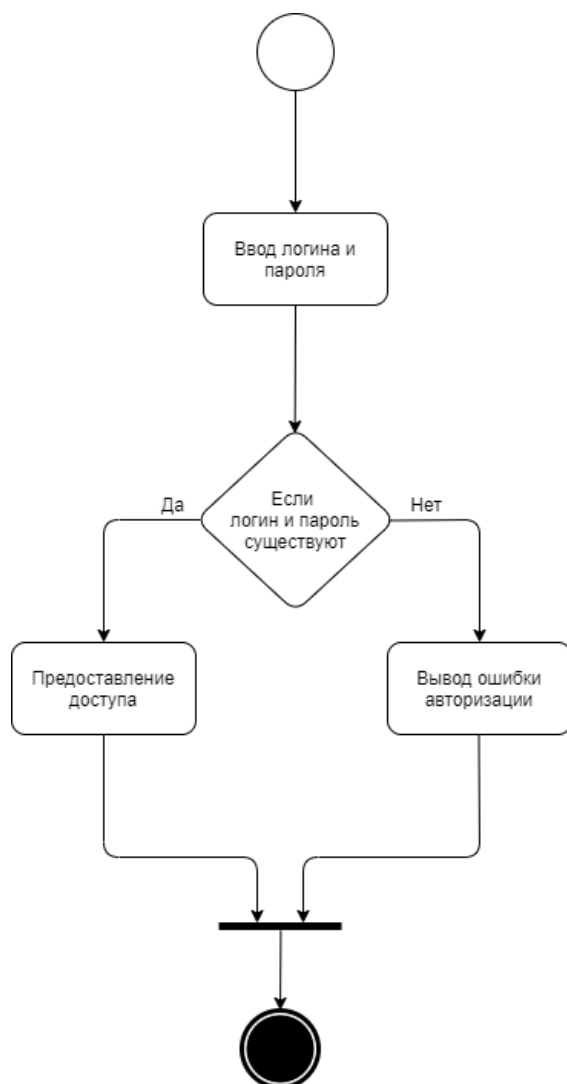


Рисунок 2.14 - Диаграмма видов деятельности авторизации

2.6 Описание результатов макетирования

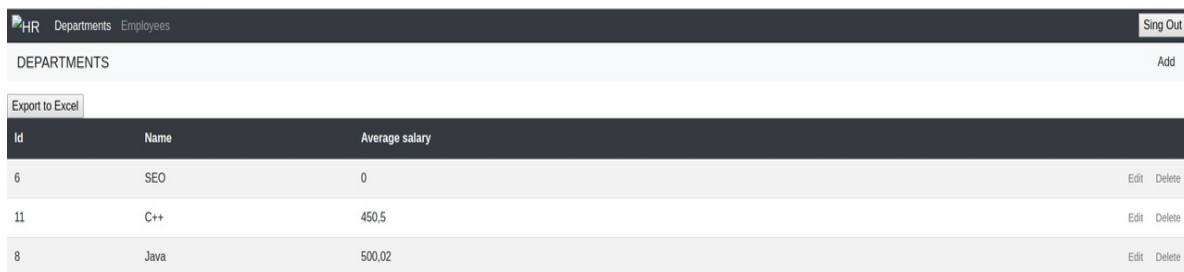
1. Авторизация:

User name:

Password:

Рисунок 2.15 - Макет авторизации

2. Просмотр всех отделов



HR Departments Employees Sing Out

DEPARTMENTS Add

Export to Excel

Id	Name	Average salary	
6	SEO	0	Edit Delete
11	C++	450,5	Edit Delete
8	Java	500,02	Edit Delete

Рисунок 2.16 - Макет просмотра всех отделов

3. Редактирование отдела



HR Departments Employees Sing Out

UPDATE DEPARTMENT

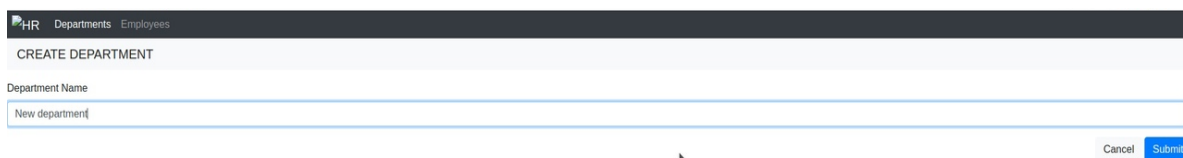
Department Name

Java

Cancel Submit

Рисунок 2.17 - Макет редактирования отдела

4. Добавление отдела



HR Departments Employees Sing Out

CREATE DEPARTMENT

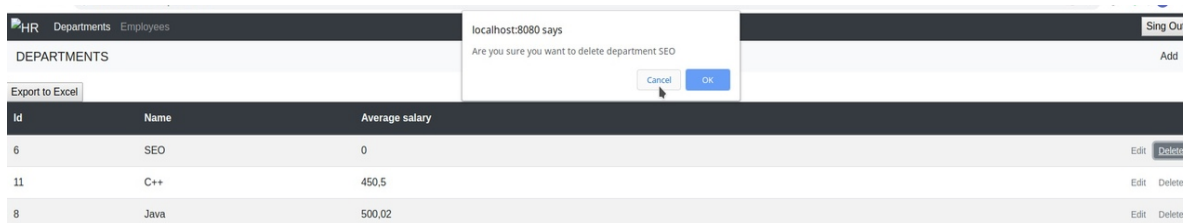
Department Name

New department

Cancel Submit

Рисунок 2.18 - Макет добавления отдела

5. Удаление отдела



HR Departments Employees Sing Out

DEPARTMENTS Add

Export to Excel

localhost:8080 says
Are you sure you want to delete department SEO

Cancel OK

Id	Name	Average salary	
6	SEO	0	Edit Delete
11	C++	450,5	Edit Delete
8	Java	500,02	Edit Delete

Рисунок 2.19 - Макет удаления отдела

6. Просмотр всех сотрудников



HR Departments Employees Sing Out

EMPLOYEES Add

Id	Name	Surname	Email	Salary	Department	
3	Alex	Alex	alex@mail.com	450,5	C++	Edit Delete
1	Vlad	Zhuravlev	vlad@mail.com	500,02	Java	Edit Delete

Рисунок 2.20 - Макет просмотра сотрудников

7. Редактирование сотрудника

HR Departments Employees

UPDATE EMPLOYEE

Department: C++

Employee Name: Alex

Employee Surname: Alex

Employee Email: alex@mail.com

Employee Salary: 450,5

Cancel Submit

Рисунок 2.21 - Макет редактирования сотрудника

8. Добавление сотрудника

HR Departments Employees

CREATE EMPLOYEE

Department: SEO

Employee Name:

Employee Surname:

Employee Email:

Employee Salary:

Cancel Submit

Рисунок 2.22 - Макет добавления сотрудника

9. Удаление сотрудника

HR Departments Employees

EMPLOYEES

localhost:8080 says: Are you sure you want to delete employee Alex? Cancel OK

Sing Out Add

Id	Name	Surname	Email		Department	
3	Alex	Alex	alex@mail.com	450,5	C++	Edit Delete
1	Vlad	Zhuravlev	viad@mail.com	500,02	Java	Edit Delete

Рисунок 2.23 - Макет удаления сотрудника

ПРИМЕРНОЕ СОДЕРЖАНИЕ ОТЧЕТА ПО ЛАБОРАТОРНОЙ РАБОТЕ №7

3. РЕАЛИЗАЦИЯ И ТЕСТИРОВАНИЕ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

3.1 Диаграмма компонентов:

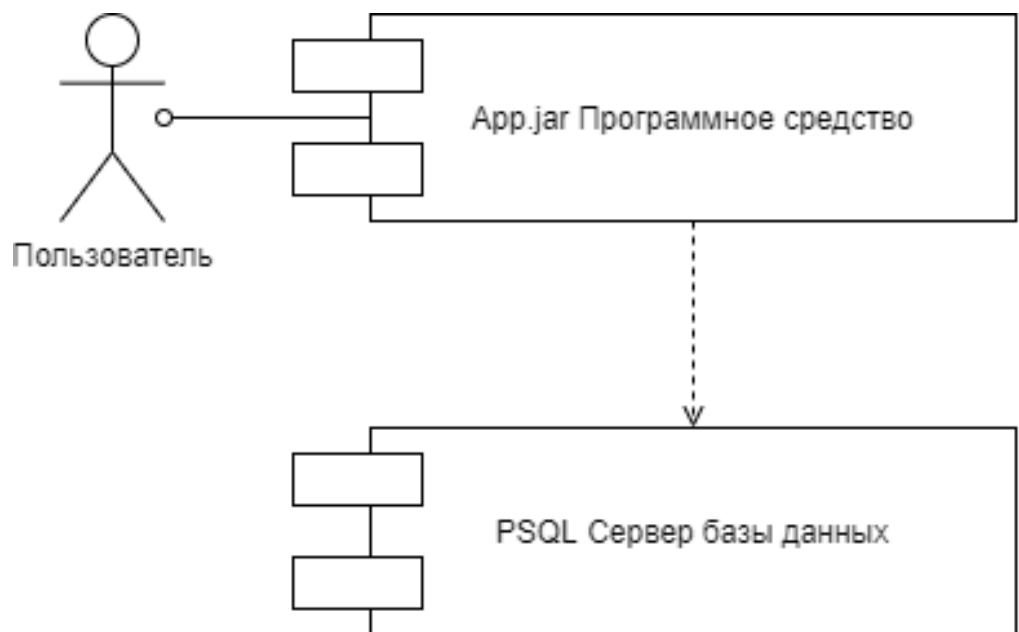


Рисунок 3.1 - Диаграмма компонентов приложения. Общий вид

3.2 Диаграмма развертывания приложения:

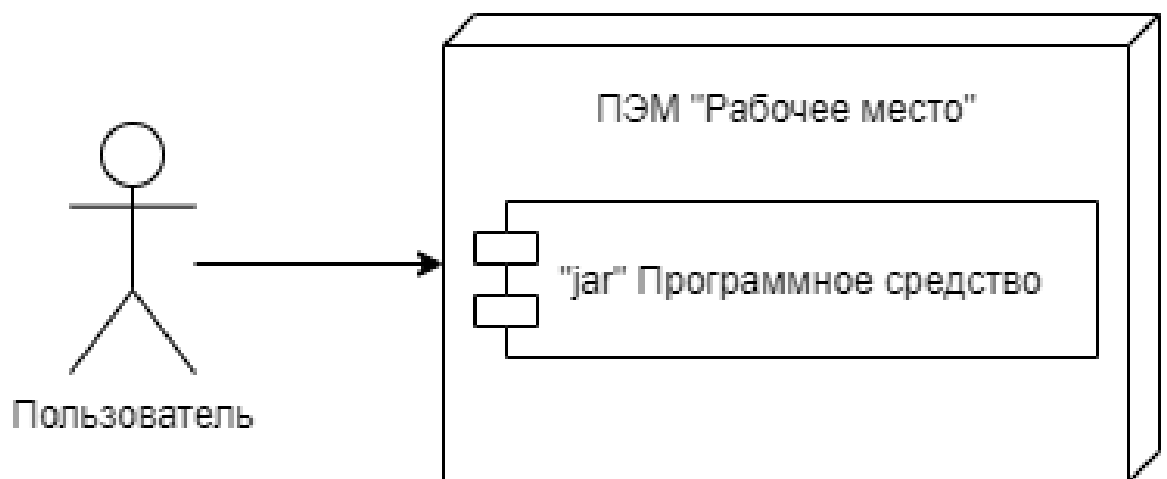


Рисунок 3.2 - Диаграмма развертывания приложения

3.3 Тестирование приложения

Тестируемая задача: Авторизация

Ожидаемый результат:

Полученный результат:

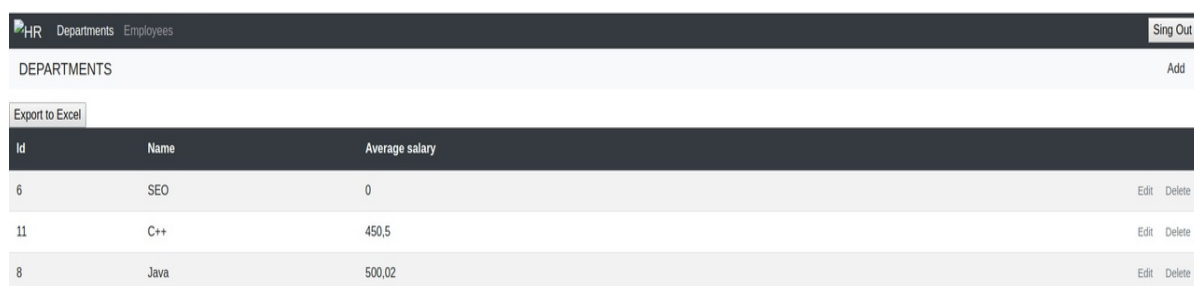
Выводы по тесту:

User name:

Password:

Рисунок 3.3 - Авторизация

Просмотр всех отделов:



Id	Name	Average salary	
6	SEO	0	Edit Delete
11	C++	450,5	Edit Delete
8	Java	500,02	Edit Delete

Рисунок 3.4 - Просмотр всех отделов

Редактирование отдела:



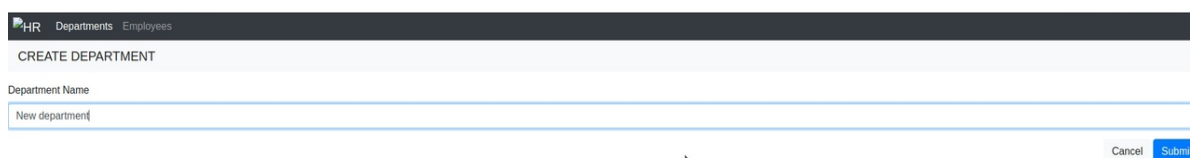
Department Name

Java

Cancel Submit

Рисунок 3.5 - Редактирования отдела

Добавление отдела:



Department Name

New department

Cancel Submit

Рисунок 3.6 - Добавление отдела

Удаление отдела

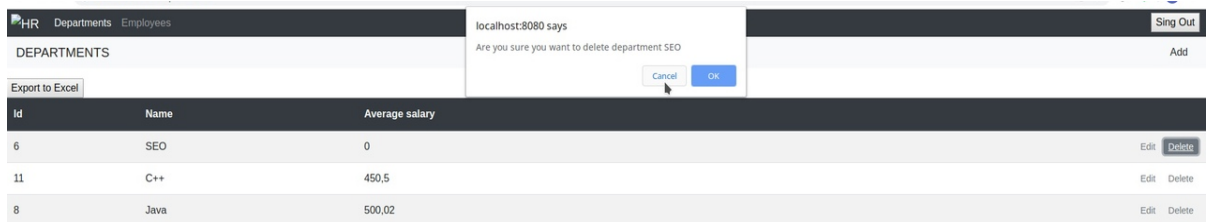


Рисунок 3.7 - Удаление отдела

Просмотр всех сотрудников

The screenshot shows the 'EMPLOYEES' section of the HR system. The table displays the following data:

Id	Name	Surname	Email	Salary	Department
3	Alex	Alex	alex@mail.com	450,5	C++
1	Vlad	Zhuravlev	vlad@mail.com	500,02	Java

Рисунок 3.8 - Просмотр сотрудников

Редактирование сотрудника

The screenshot shows the 'UPDATE EMPLOYEE' form in the HR system. The form contains the following fields:

- Department: C++
- Employee Name: Alex
- Employee Surname: Alex
- Employee Email: alex@mail.com
- Employee Salary: 450,5

Buttons: Cancel, Submit

Рисунок 3.9 - Редактирование сотрудника

Добавление сотрудника

The screenshot shows the 'CREATE EMPLOYEE' form in the HR system. The form contains the following fields:

- Department: SEO
- Employee Name:
- Employee Surname:
- Employee Email:
- Employee Salary:

Buttons: Cancel, Submit

Рисунок 3.10 - Добавление сотрудника

Удаление сотрудника

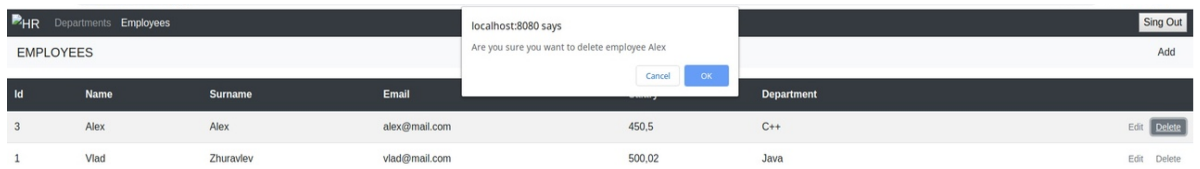


Рисунок 3.11 - Удаление сотрудника

Отчет **// не выполняем**

	A	B	C	D
1	Id	Name	Average salary	
2	6	SEO	0	
3	11	C++	450.50	
4	8	Java	500.02	

Рисунок 3.12 - Отчет

ЗАКЛЮЧЕНИЕ

В ходе данной работы была построена система управления компанией. В программе был использован объектно-ориентированный подход к проектированию и реализации приложения. Основу составили две сущности: отдел и сотрудник. В работе использовались все принципы объектно-ориентированного программирования – инкапсуляция, наследование и полиморфизм.

Результатом работы является приложение позволяющее удобно управлять компанией состоящей из нескольких отделов. Все поставленные задачи были решены.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. "Java 8. Полное руководство", Герберт Шилдт, 2015
2. "Философия Java", Брюс Эккель, 2017
3. Д.Э. Кнут. Искусство программирования, том 3. Пер. с англ. М.: Издательский дом "Вильямс", 2007. (и др. издания)
4. "Spring 5.0 Microservices - Second Edition: Scalable systems with Reactive Streams and Spring Boot", Rajesh R V, 2017
5. Oracle Java Documentation [Электронный ресурс]. – Режим доступа: <https://docs.oracle.com/javase/8/docs/technotes/tools/windows/javadoc.html>
6. ГОСТ 19.504-79. ЕСПД. Руководство программиста. Требования к содержанию и оформлению.
7. ГОСТ 2.105-95. Единая система конструкторской документации(ЕСКД). Общие требования к текстовым документам.

**УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ
«БРЕСТСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ
УНИВЕРСИТЕТ»
КАФЕДРА ИНТЕЛЛЕКТУАЛЬНЫХ ИНФОРМАЦИОННЫХ
ТЕХНОЛОГИЙ**

ПРОЕКТИРОВАНИЕ ИЕРАРХИИ КЛАССОВ

ТЕКСТ ПРОГРАММЫ

КР.ИИ-16. 180090-03 81 00

Листов: 9

Руководитель

Г.Л. Муравьев

Выполнил

В.А. Журавлев

Консультант

по ЕСПД

Г.Л. Муравьев

2021

СОДЕРЖАНИЕ

Department.java – файл с классом отдела.

Employee.java – файл с классом работника.

DepartmentDao.java – файл с классом для связывания отдела с базой данных.
EmployeeDao.java – файл с классом для связывания работника с базой данных.
DepartmentService.java – файл с классом для бизнес логики связанной с отделом.
EmployeeService.java – файл с классом для бизнес логики связанной с работником.
DepartmentController.java – файл с классом контроллером для отдела.
EmployeeController.java – файл с классом контроллером для отдела.
DepartmentWebController.java – файл с классом отвечающим за принятие запросов и отправку ответов связанных с графическим интерфейсом отдела.
EmployeeWebController.java – файл с классом отвечающим за принятие запросов и отправку ответов связанных с графическим интерфейсом работника.

Department. java

```
package com.ppvis.manager.entity;

@Entity
@Data
@Builder
@NoArgsConstructor
@AllArgsConstructor
public class Department {

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

    private String name;

    @OneToMany(mappedBy = "department")
    private List<Employee> employees;

}
```

Employee. java

```
package com.ppvis.manager.entity;

@Entity
@Data
@Builder
@NoArgsConstructor
@AllArgsConstructor
public class Employee {

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

    @Size(min = 1)
    private String name;

    @Size(min = 1)
    private String surname;

    @NotNull
    private BigDecimal salary;

    @Email
    private String mail;

    @ManyToOne
    private Department department;

}
```

DepartmentDao. java

```
package com.ppvis.manager.dao;

public interface DepartmentDao extends CrudRepository<Department, Integer> {}
```

EmployeeDao. java

```
package com.ppvis.manager.dao;

@Repository
public interface EmployeeDao extends CrudRepository<Employee, Integer> {}
```

DepartmentService. java

```
package com.ppvis.manager.service;

@Service
@RequiredArgsConstructor
```



```

public class DepartmentService {

    private final DepartmentDao departmentDao;

    public List<Department> findAll(){
        return StreamSupport.stream(departmentDao.findAll().spliterator(), false)
            .collect(Collectors.toList());
    }

    public Department findById(final Integer id){
        return departmentDao.findById(id).orElseThrow(() -> new RuntimeException());
    }

    public Department add(final Department department){
        return departmentDao.save(department);
    }

    public Department update(final Department newDepartment, final Integer id){
        Department department = findById(id);
        department.setName(newDepartment.getName());
        return departmentDao.save(department);
    }

    public void delete(final Integer id){
        departmentDao.deleteById(id);
    }
}

```

EmployeeService. java

```
package com.ppvis.manager.service;
```

```

@Service
@RequiredArgsConstructor
public class EmployeeService {

    private final EmployeeDao employeeDao;

    public List<Employee> findAll(){
        return StreamSupport.stream(employeeDao.findAll().spliterator(), false)
            .collect(Collectors.toList());
    }

    public Employee findById(final Integer id){
        return employeeDao.findById(id).orElseThrow(() -> new RuntimeException());
    }

    public Employee add(final Employee employee){
        return employeeDao.save(employee);
    }

    public Employee update(final Employee newEmployee, final Integer id){
        Employee employee = findById(id);
        employee.setName(newEmployee.getName());
        employee.setSurname(newEmployee.getSurname());
        employee.setSalary(newEmployee.getSalary());
        employee.setMail(newEmployee.getMail());
        employee.setDepartment(newEmployee.getDepartment());
        return employeeDao.save(employee);
    }

    public void delete(final Integer id){
        employeeDao.deleteById(id);
    }
}

```

DepartmentController. java

```
package com.ppvis.manager.controller;
```

```
@RestController
```

```

@RequestMapping("rest/departments")
@RequiredArgsConstructor
public class DepartmentController {

    private final DepartmentService departmentService;
    private final DepartmentMapper departmentMapper;
    private final DepartmentExcelService departmentExcelService;

    @GetMapping
    public List<DepartmentDto> findAll(){
        return departmentService.findAll().stream()
            .map(departmentMapper::fromEntity)
            .collect(Collectors.toList());
    }

    @GetMapping("/{id}")
    public DepartmentDto findById(@PathVariable final Integer id){
        return departmentMapper.fromEntity(departmentService.findById(id));
    }

    @PostMapping
    public DepartmentDto add(@RequestBody final DepartmentDto departmentDto){
        return departmentMapper.fromEntity(
            departmentService.add(
                departmentMapper.toEntity(departmentDto)));
    }

    @PutMapping("/{id}")
    public DepartmentDto update(@RequestBody final DepartmentDto newDepartmentDto,
        @PathVariable final Integer id){
        return departmentMapper.fromEntity(
            departmentService.update(
                departmentMapper.toEntity(newDepartmentDto), id));
    }

    @DeleteMapping("/{id}")
    public void delete(@PathVariable final Integer id){
        departmentService.delete(id);
    }

    @GetMapping(value = "/excel/departments.xlsx")
    public ResponseEntity<InputStreamResource> excelCarsReport() {

        ByteArrayInputStream in = departmentExcelService.departmentsToExcel();

        HttpHeaders headers = new HttpHeaders();
        headers.add("Content-Disposition", "attachment; filename=departments.xlsx");

        return ResponseEntity
            .ok()
            .headers(headers)
            .body(new InputStreamResource(in));
    }
}

```

EmployeeController. java

```

package com.ppvis.manager.controller;

@RestController
@RequestMapping("rest/employees")
@RequiredArgsConstructor
public class EmployeeController {

    private final EmployeeService employeeService;
    private final EmployeeMapper employeeMapper;

    @GetMapping
    public List<EmployeeDto> findAll(){
        return employeeService.findAll().stream()

```

```

        .map(employeeMapper::fromEntity)
        .collect(Collectors.toList());
    }

    @GetMapping("/{id}")
    public EmployeeDto findById(@PathVariable final Integer id){
        return employeeMapper.fromEntity(employeeService.findById(id));
    }

    @PostMapping
    public EmployeeDto add(@RequestBody final EmployeeDto employeeDto){
        return employeeMapper.fromEntity(
            employeeService.add(
                employeeMapper.toEntity(employeeDto)));
    }

    @PutMapping("/{id}")
    public EmployeeDto update(@RequestBody final EmployeeDto newEmployeeDto,
    @PathVariable final Integer id){
        return employeeMapper.fromEntity(
            employeeService.update(
                employeeMapper.toEntity(newEmployeeDto), id));
    }

    @DeleteMapping("/{id}")
    public void delete(@PathVariable final Integer id){
        employeeService.delete(id);
    }
}

```

DepartmentWebController. java

```
package com.ppvis.manager.web;
```

```

@Controller
@RequiredArgsConstructor
public class DepartmentWebController {

    private final DepartmentRestService departmentRestService;

    @GetMapping("departments")
    public String findAll(final Map<String, Object> model){
        List<DepartmentDto> departmentDtos = departmentRestService.findAll();
        model.put("departmentDtos", departmentDtos);
        return "departments";
    }

    @GetMapping("department")
    public final String gotoAddDepartmentPage(final Map<String, Object> model) {
        model.put("department", new Department());
        model.put("isNew", true);
        model.put("hide", true);
        model.put("error", "");
        return "department_add";
    }

    @GetMapping("departments/{id}")
    public String findById(@PathVariable final Integer id, final Map<String, Object>
model){
        DepartmentDto departmentDto = departmentRestService.findById(id);
        model.put("department", departmentDto);
        model.put("isNew", false);
        model.put("hide", true);
        model.put("error", "");
        return "department_update";
    }

    @PostMapping("departments")
    public String add(@RequestParam final String name,
        final Map<String, Object> model){

```

```

        DepartmentDto departmentDto = DepartmentDto.builder()
            .name(name)
            .build();
    try {
        departmentRestService.add(departmentDto);
    } catch (ConstraintViolationException ex) {
        Optional<String> optionalString = ex.getConstraintViolations().stream()
            .map(ConstraintViolation::getMessage)
            .reduce((x, y) -> x + '\n' + y);
        String errors = optionalString.get();
        model.put("error", errors);
        model.put("hide", false);
        model.put("department", new Department());
        model.put("isNew", true);
        return "department_add";
    }
    return "redirect:/departments";
}

@PostMapping("departments/{id}")
public String update(@RequestParam final String name,
                    @PathVariable final Integer id,
                    final Map<String, Object> model) {
    DepartmentDto departmentDto = DepartmentDto.builder()
        .name(name)
        .build();
    try {
        departmentRestService.update(departmentDto, id);
    } catch (ConstraintViolationException ex) {
        DepartmentDto realDepartment = departmentRestService.findById(id);
        Optional<String> optionalString = ex.getConstraintViolations().stream()
            .map(ConstraintViolation::getMessage)
            .reduce((x, y) -> x + '\n' + y);
        String errors = optionalString.get();
        model.put("error", errors);
        model.put("hide", false);
        model.put("department", realDepartment);
        model.put("isNew", false);
        return "department_update";
    }
    return "redirect:/departments";
}

@PostMapping("departments/{id}/delete")
public String delete(@PathVariable final Integer id,
                    final Map<String, Object> model) {
    departmentRestService.delete(id);
    return "redirect:/departments";
}
}

```

EmployeeWebController.java

```

package com.ppvis.manager.web;

@Controller
@RequiredArgsConstructor
public class EmployeeWebController {

    private final EmployeeRestService employeeRestService;
    private final DepartmentRestService departmentRestService;

    @GetMapping("employees")
    public String findAll(final Map<String, Object> model) {
        List<EmployeeDto> employeeDtos = employeeRestService.findAll();
        for (EmployeeDto employeeDto : employeeDtos) {
            DepartmentDto department =
                departmentRestService.findById(employeeDto.getDepartmentId());
            employeeDto.setDepartmentName(department.getName());
        }
    }
}

```

```

        model.put("employeeDtos", employeeDtos);
        return "employees";
    }

    @GetMapping("employee")
    public final String gotoAddEmployeePage(final Map<String, Object> model) {
        List<DepartmentDto> departmentDtos = departmentRestService.findAll();
        model.put("departmentDtos", departmentDtos);
        model.put("hide", true);
        model.put("error", "");
        return "employee_add";
    }

    @GetMapping("employees/{id}")
    public String findById(@PathVariable final Integer id, final Map<String, Object>
model){
        EmployeeDto employeeDto = employeeRestService.findById(id);
        DepartmentDto employeeDepartment =
departmentRestService.findById(employeeDto.getDepartmentId());
        List<DepartmentDto> foundDepartmentDtos = departmentRestService.findAll();
        List<DepartmentDto> departmentDtos = new ArrayList<>();
        departmentDtos.add(employeeDepartment);
        for (DepartmentDto departmentDto : foundDepartmentDtos) {
            if(!departmentDto.equals(employeeDepartment)){
                departmentDtos.add(departmentDto);
            }
        }
        model.put("departmentDtos", departmentDtos);
        model.put("employee", employeeDto);
        model.put("hide", true);
        model.put("error", "");
        return "employee_update";
    }

    @PostMapping("employees")
    public String add(@RequestParam final String name,
                     @RequestParam final String surname,
                     @RequestParam final String email,
                     @RequestParam String salary,
                     @RequestParam final Integer departmentId,
                     final Map<String, Object> model){
        salary = salary.replaceAll(",", ".");
        EmployeeDto employeeDto = EmployeeDto.builder()
            .name(name)
            .surname(surname)
            .mail(email)
            .salary(new BigDecimal(salary))
            .departmentId(departmentId)
            .build();

        try {
            employeeRestService.add(employeeDto);
        } catch (ConstraintViolationException ex){
            Optional<String> optionalString = ex.getConstraintViolations().stream()
                .map(ConstraintViolation::getMessage)
                .reduce((x, y) -> x + '\n' + y);
            String errors = optionalString.get();

            List<DepartmentDto> foundDepartmentDtos = departmentRestService.findAll();
            model.put("departmentDtos", foundDepartmentDtos);
            model.put("error", errors);
            model.put("hide", false);
            return "employee_add";
        }
        return "redirect:/employees";
    }

    @PostMapping("employees/{id}")
    public String update(@RequestParam final String name,
                        @RequestParam final String surname,
                        @RequestParam final String email,
                        @RequestParam String salary,

```

```

        @RequestParam final Integer departmentId,
        @PathVariable final Integer id,
        final Map<String, Object> model){
    salary = salary.replaceAll(",", ".");
    salary = salary.replaceAll(" ", "");
    double d = Double.parseDouble(salary);
    BigDecimal bigDecimalSalary = BigDecimal.valueOf(d);
    EmployeeDto employeeDto = EmployeeDto.builder()
        .name(name)
        .surname(surname)
        .mail(email)
        .salary(bigDecimalSalary)
        .departmentId(departmentId)
        .build();
    try {
        employeeRestService.update(employeeDto, id);
    } catch (ConstraintViolationException ex){
        EmployeeDto realEmployee = employeeRestService.findById(id);
        DepartmentDto employeeDepartment =
departmentRestService.findById(realEmployee.getDepartmentId());
        String errors = "";
        for(ConstraintViolation<?> er : ex.getConstraintViolations()){
            errors += "(" + er.getMessage() + "); ";
        }
        List<DepartmentDto> foundDepartmentDtos =
departmentRestService.findAll();
        List<DepartmentDto> departmentDtos = new ArrayList<>();
        departmentDtos.add(employeeDepartment);
        for (DepartmentDto departmentDto : foundDepartmentDtos) {
            if(!departmentDto.equals(employeeDepartment)){
                departmentDtos.add(departmentDto);
            }
        }
        model.put("departmentDtos", departmentDtos);
        model.put("error", errors);
        model.put("hide", false);
        model.put("employee", realEmployee);
        return "employee_update";
    }
    return "redirect:/employees";
}

@PostMapping("employees/{id}/delete")
public String delete(@PathVariable final Integer id,
                    final Map<String, Object> model){
    employeeRestService.delete(id);
    return "redirect:/employees";
}

}

```