**Содержание**

[Введение 3](#_Toc199762014)

[1 Описание предметной области 4](#_Toc199762015)

[1.1 Видение проекта 4](#_Toc199762016)

[1.2 Математические (вербальные) модели 5](#_Toc199762017)

[2 Проектирование программного модуля 7](#_Toc199762018)

[2.1 Перечень функциональности (User Stories) 7](#_Toc199762019)

[2.2 Сценарии взаимодействия (краткий и подробный) 9](#_Toc199762020)

[2.3 Прототипирование интерфейса взаимодействия с пользователем (графического интерфейса) 18](#_Toc199762021)

[2.4 Составление плана работы 26](#_Toc199762022)

[3 Разработка приложения 28](#_Toc199762023)

[3.1 Диаграмма классов 28](#_Toc199762024)

[3.2 Диаграмма деятельности 38](#_Toc199762025)

[4 Разработка документации приложения 41](#_Toc199762026)

[4.1 Выбор и обоснование технологий, используемых в программном модуле 41](#_Toc199762027)

[4.2 Описание используемых структур данных 42](#_Toc199762028)

[5 Тестирование приложения 46](#_Toc199762029)

[Заключение 65](#_Toc199762030)

[Список литературы 66](#_Toc199762031)

[Приложение 1 – Программный код приложения 67](#_Toc199762032)

# Введение

В современном образовательном процессе важную роль играет не только качество преподавания, но и состояние материально-технической базы учебных заведений. Эффективное управление имуществом и техническим обслуживанием в учреждениях образования является ключевым фактором, способствующим созданию комфортной и безопасной образовательной среды. В связи с этим, автоматизация работы технического персонала становится необходимостью, позволяющей оптимизировать процессы обслуживания, повысить их эффективность и снизить затраты времени и ресурсов.

Актуальная задача заключается в создании комплексного решения, которое не только автоматизирует процессы обслуживания имущества, но и улучшит взаимодействие между всеми участниками образовательного процесса, что в конечном итоге приведет к повышению качества образования и улучшению условий для обучения.

Целью разработки веб-приложения является создание эффективного инструмента, который автоматизирует работу технического персонала. Приложение должно обеспечить работникам возможность быстро и удобно планировать, отслеживать и выполнять работы по обслуживанию. В результате, приложение должно способствовать повышению качества обслуживания, оптимизации использования ресурсов и улучшению состояния материально-технической базы образовательных учреждений.

Ожидаемым результатом данной работы является создание функционального веб-приложения для помощи в обслуживании имущества в учреждениях образования, предоставляющего функционал для управления заявками на обслуживание, отслеживания выполнения работ и генерации отчетов.

# 1 Описание предметной области

* 1. Видение проекта

Основная цель системы — сократить время, затрачиваемое на обмен информацией между преподавателями и обслуживающим персоналом, а также улучшить оперативность в решении возникающих проблем.

Разрабатываемое WEB-приложение предназначено для повышения производительности работников, занимающихся обслуживанием помещений и техники в учебных организациях, таких как школы, лицеи, гимназии и колледжи.

Веб-приложение должно отвечать следующему набору требований:

1. Предоставлять информацию о всех активных заявках;
2. Предоставлять информацию об активных заявках по категориям проблем сотрудника;
3. Предоставлять информацию о работниках руководителю;
4. Предоставлять информацию о несчастных случаях;
5. Предоставлять информацию о выполненных заявках, с указанием даты начала выполнения и конца;
6. Предоставлять полную информацию о заявке;
7. Предоставлять полную информацию о сотруднике;
8. Возможность руководителю технического персонала удалять несчастные случаи;
9. Возможность руководителю добавлять несчастные случаи;
10. Возможность руководителю удалять заявки;
11. Возможность руководителю назначать заявки сотруднику;
12. Возможность работнику добавлять заявки в свой список выполняемых заявок;
13. Возможность работнику менять статус заявки, которую он выполняет;
14. Возможность работнику менять свой статус занятости;
15. Возможность руководителю экспортировать данные работников или заявок в Excel или PDF;
16. Предоставлять руководителю календарь с временными рамками выполнения заявок выбранного сотрудника.

Главной причиной внедрения данной системы является отсутствие у образовательных учреждений и преподавателей специализированной платформы, которая полностью охватывала бы процесс подачи, обработки и отслеживания заявок.

* 1. Математические (вербальные) модели

На рисунке 1.1 представлена навигация между страницами.

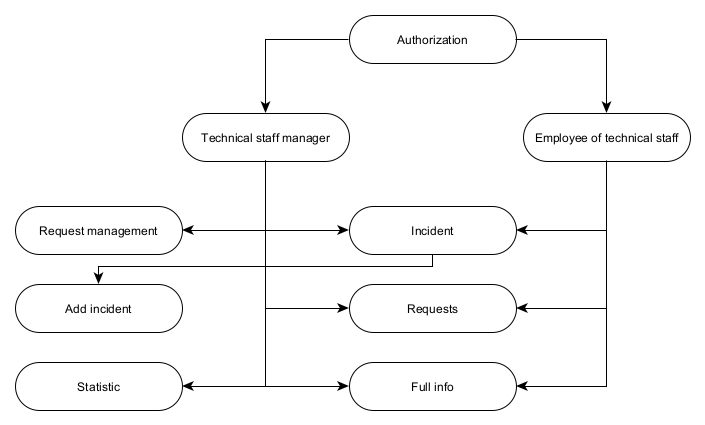


Рисунок 1.1 – Схема навигации

Система ролей приложения предусматривает разделение на сотрудников технического персонала и руководителя технического персонала.

Незарегистрированный пользователь имеет доступ только к странице «Authorization». После авторизации, в зависимости от роли, пользователь перенаправляется на одну из страниц: «Technical staff manager» — для руководителя и «Employee of technical staff» — для работника.

На странице «Employee of technical staff» работник технического персонала может просматривать историю своих выполненных заявок и заявки в работе. С этой страницы пользователь может перейти на страницы «Incident», «Requests» и «Full info»

На странице «Technical staff manager» руководитель имеет возможность просмотреть список сотрудников технического персонала и экспортировать его в файл. С этой страницы он может перейти на страницы «Request management», «Statistic», «Incident», «Requests» и «Full info».

На странице «Incident» представлена информация о несчастных случаях во время выполнения заявок. Сотрудник технического персонала может только просматривать информацию на этой странице, в то время как руководитель технического персонала может удалять и экспортировать данные. Так же для руководителя доступна кнопка «Добавить» для перехода на страницу «Add incident»

Руководитель на странице «Add incident» указывает информацию нового несчастного случая и добавляет его в систему.

На странице «Requests» представлена информация о доступных заявках. Сотрудник имеет возможность добавить заявку в свой список выполняемых заявок. Руководитель так же имеет доступ к этой странице с возможностью назначить выбранному сотруднику заявку для выполнения и экспортировать доступные заявки в файл.

На странице «Full info» указана подробная информация о сотруднике технического персонала и список выполняемых им заявок. Руководитель так же может просмотреть эту информацию с возможностью экспорта данных в файл.

Страница «Request management» предназначена только руководителю для управления заявками. Он может удалить заявку из общего списка доступных заявок и экспортировать данные в файл.

Страница «Statistic» предназначена только руководителю для просмотра статистики по выбранному сотруднику с возможность экспорта данных в файл.

# 2 Проектирование программного модуля

2.1 Перечень функциональности (User Stories)

Use case диаграмма «Сотрудник технического персонала» представлена на рисунке 2.1.



Рисунок 2.1 – Диаграмма вариантов использования для сотрудника технического персонала

Use case диаграмма «Руководитель технического персонала» представлена на рисунке 2.2.

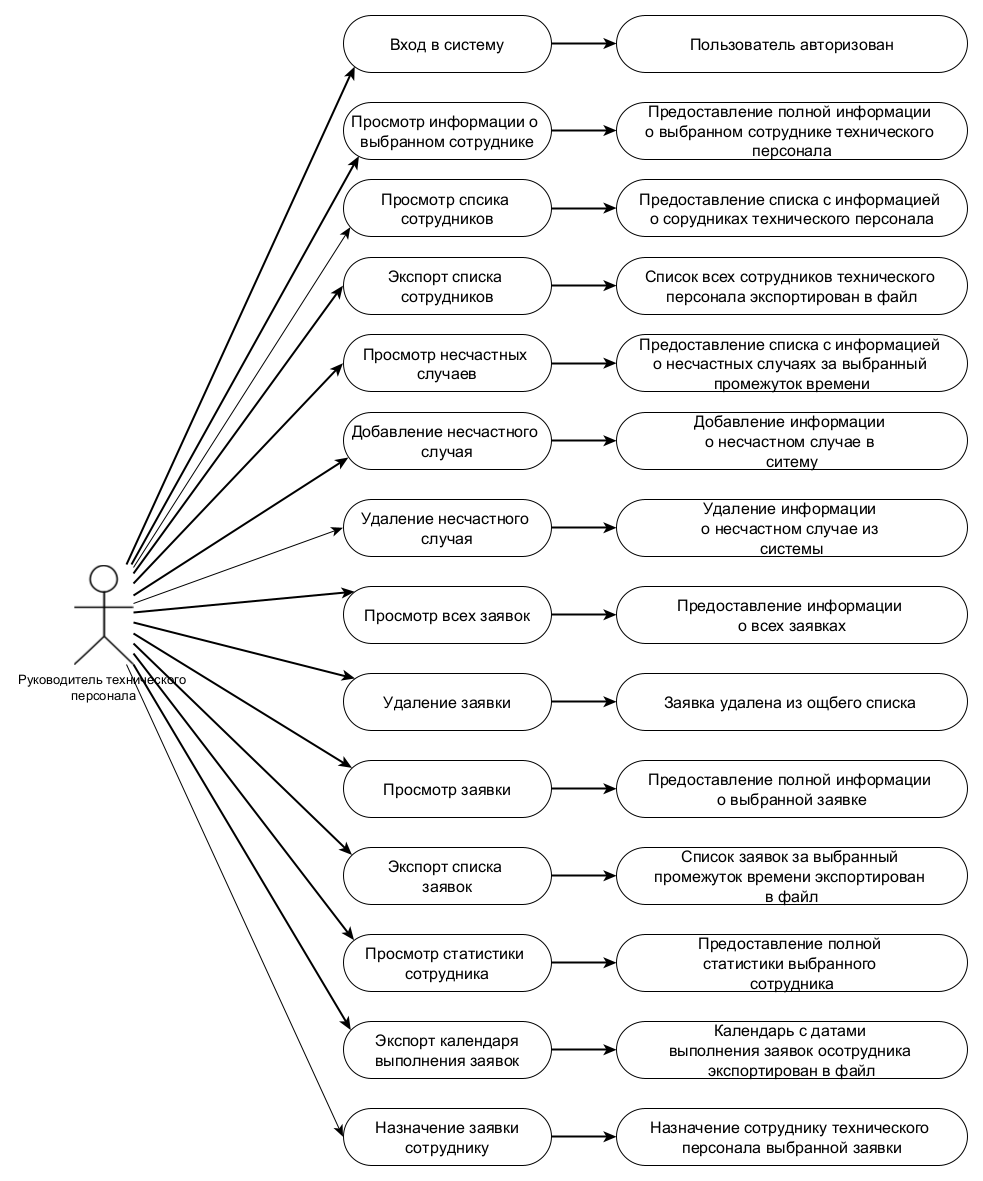


Рисунок 2.2 – Диаграмма вариантов использования для руководителя технического персонала

2.2 Сценарии взаимодействия (краткий и подробный)

Краткие сценарии взаимодействия представлены в таблице 2.1.

Таблица 2.1 – Краткие сценарии взаимодействия

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Действие | Тип объекта | Объект | Доп. функции | Сущность |
| Работник технического персонала | | | | |
| Вход в систему | Сущность | User | - | User |
| Просмотр всех заявок | Список | List<Request> | - | Request |
| Просмотр заявки | Сущность | Request | - | Request |
| Принятие заявки | Сущность | RequestAssignment | - | RequestAssignment |
| Изменение статуса заявки | Сущность | Request | - | Request |
| Изменение статуса занятости | Сущность | EmployeeOfTechnical Staff | - | EmployeeOfTechnical Staff |
| Просмотр несчастных случаев | Список | List<Incident> | - | Incident |
| Просмотр информации о сотруднике | Сущность | EmployeeOfTechnical Staff | - | EmployeeOfTechnical Staff |
| Просмотр выполняемых заявок | Список | List<Request> | - | Request |
| Просмотр выполненных заявок | Список | List<Request> | - | Request |
| Руководитель технического персонала | | | | |
| Вход в систему | Сущность | User | - | User |
| Просмотр информации о выбранном сотруднике | Сущность | EmployeeOfTechnical Staff | - | EmployeeOfTechnical Staff |
| Просмотр списка сотрудников | Список | List<EmployeeOf  TechnicalStaff> | - | EmployeeOfTechnical Staff |
| Экспорт списка сотрудников | Сущность | EmployeeOfTechnical Staff | - | EmployeeOfTechnical Staff |
| Просмотр несчастных случаев | Список | List<Incident> | - | Incident |

Продолжение таблицы 2.1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Действие | Тип объекта | Объект | Доп. функции | Сущность |
| Добавление несчастного случая | Сущность | Incident | - | Incident |
| Удаление несчастного случая | Список | Incident | - | Incident |
| Просмотр всех заявок | Список | List<Request> | - | Request |
| Удаление заявки | Сущность | Request | - | Request |
| Просмотр заявки | Сущность | Request | - | Request |
| Экспорт списка заявок | Сущность | Request | - | Request |
| Просмотр статистики сотрудника | Сущность | Request, Task, WorkTimeLog | - | Request, Task, WorkTimeLog |
| Экспорт календаря выполнения заявок | Сущность | Request, Task, WorkTimeLog | - | Request, Task, WorkTimeLog |
| Назначение заявки сотруднику | Сущность | RequestAssignment | - | RequestAssignment |

В таблицах 2.2 – 2.24 представлены подробные сценарии взаимодействия.

Таблица 2.2 – Вход в систему (для сотрудника)

|  |  |
| --- | --- |
| Вход в систему (для сотрудника) | |
| Входные данные | Номер телефона/адрес электронной почты, пароль пользователя |
| Выходные данные | - |
| Предусловия | Нажатие на кнопку «Войти |
| Постусловия | Авторизация пользователя и переход на страницу для сотрудников |
| Последовательность действий | Ввод данных в поля «Номер телефона/адрес электронной почты», и «пароль». Нажатие на кнопку «Войти». Получение данных о роли из таблицы User и переход на страницу employee |

Таблицы 2.3 – Просмотр всех заявок

|  |  |
| --- | --- |
| Просмотр всех заявок | |
| Входные данные | Данные таблиц Request, Task и Location |
| Выходные данные | Список доступных по категориям проблем заявок |
| Предусловия | Пользователь должен быть авторизован и находится на странице Employee of technical staff. Нажатие на ссылку в навигационном меню «Заявки» |
| Постусловия | Вывод заявок |
| Последовательность действий | Получение данных из таблиц Request, Task, Location базы данных. Вывод на экран заявок в виде таблицы |

Таблицы 2.4 – Просмотр заявки

|  |  |
| --- | --- |
| Просмотр заявки | |
| Входные данные | Данные таблиц Request, Task, Location |
| Выходные данные | Подробная информация о выбранной заявке |
| Предусловия | Пользователь должен быть авторизован. Нажатие на Id заявки в таблице заявок |
| Постусловия | Отображение информации о заявке |
| Последовательность действий | Получение данных из таблиц Request, Task, Location базы данных. Вывод на экран информации о заявке в модальном окне |

Таблица 2.5 – Принятие заявки

|  |  |
| --- | --- |
| Принятие заявки | |
| Входные данные | Данные таблиц Request, Task, Location |
| Выходные данные | - |
| Предусловия | Пользователь должен быть авторизован. Нажатие на кнопку «Добавить» в модальном окне с подробным описанием заявки |
| Постусловия | Добавление в список выполняемых заявок |
| Последовательность действий | Получение данных из таблиц Request, Task, Location базы данных для отображения доступных заявок. Нажатие на кнопку «Добавить» |

Таблица 2.6 – Изменение статуса заявки

|  |  |
| --- | --- |
| Изменение статуса заявки | |
| Входные данные | Статус заявки из таблицы Request |
| Выходные данные | - |
| Предусловия | Пользователь должен быть авторизован. Выбор статуса заявки из выпадающего меню |
| Постусловия | Статус занятости заявки изменен |
| Последовательность действий | Нажать на ячейку с информацией о статусе заявки в таблице выполняемых заявок и выбрать один из доступных статусов в выпадающем меню |

Таблица 2.7 – Изменение статуса занятости

|  |  |
| --- | --- |
| Изменение статуса занятости | |
| Входные данные | Статус занятости сотрудника из таблицы EmployeeOfTechnicalStaff |
| Выходные данные | - |
| Предусловия | Пользователь должен быть авторизован. Выбор статуса занятости из выпадающего меню |
| Постусловия | Статус занятости сотрудника изменен |
| Последовательность действий | Нажать на текущий статус занятости на странице работника. Нахождение записи в таблице EmployeeOfTEchnicalStaff по Id и изменение статуса занятости. |

Таблица 2.8 – Просмотр несчастных случаев (для сотрудника)

|  |  |
| --- | --- |
| Просмотр несчастных случаев (для сотрудника) | |
| Входные данные | Данные таблицы Incidents |
| Выходные данные | Список всех несчастных случаев |
| Предусловия | Пользователь должен быть авторизован и находится на странице Employee of technical staff. Нажатие на ссылку в навигационном меню «Несчастные случаи» |
| Постусловия | Вывод имеющихся несчастных случаев |
| Последовательность действий | Получение данных из таблицы Incidents базы данных. Вывод на экран информации об несчастных случаях в виде таблицы |

Таблица 2.9 – Просмотр информации о сотруднике (для сотрудника)

|  |  |
| --- | --- |
| Просмотр информации о сотруднике (для сотрудника) | |
| Входные данные | Данные таблиц EmployeeOfTechnicalStaff, User |
| Выходные данные | Полная информация о сотруднике технического персонала |
| Предусловия | Пользователь должен быть авторизован и находится на странице Employee of technical Staff. Нажатие на текст с ФИО |
| Постусловия | Вывод информации о сотруднике |
| Последовательность действий | Нажатие на текст ФИО. Переход на страницу Full info. Получение данных из таблиц EmployeeOfTechnicalStaff и таблицы User базы данных. Вывод на экран информации о сотруднике. |

Таблица 2.10 – Просмотр выполняемых заявок

|  |  |
| --- | --- |
| Просмотр выполняемых заявок | |
| Входные данные | Данные таблицы Request, Location, Task |
| Выходные данные | Список с выполняемыми заявками сотрудника |
| Предусловия | Пользователь должен быть авторизован и находится на странице Employee of technical staff |
| Постусловия | Отображение таблицы с выполняемыми заявками |
| Последовательность действий | Получение данных из таблиц Request, Task, Location базы данных. Отображение данных в виде таблицы. |

Таблица 2.11 – Просмотр выполненных заявок

|  |  |
| --- | --- |
| Просмотр выполненных заявок | |
| Входные данные | Данные таблиц Request, Location, Task и WorkTimeLog |
| Выходные данные | Список с выполненными заявками сотрудника |
| Предусловия | Пользователь должен быть авторизован и находится на странице Employee of technical staff |
| Постусловия | Отображение таблицы с выполненными заявками |
| Последовательность действий | Получение данных из таблиц Request, Task, Location и WorkTimeLog базы данных. Отображение данных в виде таблицы. |

Таблица 2.12 – Вход в систему (для руководителя)

|  |  |
| --- | --- |
| Вход в систему (для руководителя) | |
| Входные данные | Номер телефона/адрес электронной почты, пароль пользователя |
| Выходные данные | - |
| Предусловия | Нажатие на кнопку «Войти» |
| Постусловия | Авторизация пользователя и переход в личный кабинет руководителя |
| Последовательность действий | Ввод данных в поля «Номер телефона/адрес электронной почты», и «пароль». Нажатие на кнопку «Войти». Получение информации о роли из таблицы User и переход на страницу Technical staff manager |

Таблица 2.13 – Просмотр информации о сотруднике (для руководителя)

|  |  |
| --- | --- |
| Просмотр информации о сотруднике (для руководителя) | |
| Входные данные | Данные таблиц EmployeeOfTechnicalStaff и User |
| Выходные данные | Полная информация о сотруднике технического персонала |
| Предусловия | Руководитель технического персонала должен быть авторизован. Нажатие на Id в таблице с работниками |
| Постусловия | Вывод информации о сотруднике |
| Последовательность действий | Нажатие на Id сотрудника в таблице. Переход на страницу Full info. Получение данных из таблиц EmployeeOfTechnicalStaff и User базы данных. Вывод на экран информации о сотруднике. |

Таблица 2.14 – Просмотр списка сотрудников

|  |  |
| --- | --- |
| Просмотр списка сотрудников | |
| Входные данные | Данные таблиц EmployeeOfTechnicalStaff и User |
| Выходные данные | Таблица работников |
| Предусловия | Руководитель должен быть авторизован и находится на странице Technical staff manager |
| Постусловия | Вывод таблицы с сотрудниками |
| Последовательность действий | Получение данных из таблиц EmployeeOfTechnicalStaff и User. Вывод данных в виде таблицы |

Таблица 2.15 – Экспорт списка сотрудников

|  |  |
| --- | --- |
| Экспорт списка сотрудников | |
| Входные данные | Данные таблицы EmployeeOfTechnicalStaff и User |
| Выходные данные | Файл со списком сотрудников (Excel или PDF) |
| Предусловия | Руководитель должен быть авторизован и находится на странице Technical staff manager. Нажатие на кнопку «PDF» или «Excel» |
| Постусловия | Получение файла со списком сотрудников |
| Последовательность действий | Получение данных из таблиц EmployeeOfTechnicalStaff и User базы данных. Нажатие на кнопку. Получение файла со списком сотрудников. |

Таблица 2.16 – Просмотр несчастных случаев (для руководителя)

|  |  |
| --- | --- |
| Просмотр несчастных случаев (для руководителя) | |
| Входные данные | Данные таблицы Incidents |
| Выходные данные | Список всех несчастных случаев |
| Предусловия | Руководитель должен быть авторизован и находится на странице Technical staff manager. Нажатие на текст в навигационном меню «Несчастные случаи» |
| Постусловия | Вывод имеющихся несчастных случаев |
| Последовательность действий | Получение данных из таблицы Incidents базы данных. Вывод на экран информации об несчастных случаях в виде таблицы |

Таблица 2.17 – Добавление несчастного случая

|  |  |
| --- | --- |
| Добавление несчастного случая | |
| Входные данные | Дата и время, описание, идентификатор назначения сотруднику заявки |
| Выходные данные | - |
| Предусловия | Руководитель должен быть авторизован. Нажатие на кнопку «Добавить» на странице Incidentss |
| Постусловия | Добавление несчастного случая в систему |
| Последовательность действий | Нажатие на кнопку «Добавить», ввод информации, подтверждение, добавление записи в таблицу Incidents |

Таблица 2.18 – Удаление несчастного случая

|  |  |
| --- | --- |
| Удаление несчастного случая | |
| Входные данные | Id несчастного случая |
| Выходные данные | - |
| Предусловия | Руководитель должен быть авторизован. Нажатие на иконку с корзиной в таблице несчастных случаев на странице Incidents |
| Постусловия | Несчастный случай удален |
| Последовательность действий | Нахождение и удаление записи в таблице Incidents по Id. Удаления несчастного случая из общего списка |

Таблица 2.19 – Просмотр всех заявок

|  |  |
| --- | --- |
| Просмотр всех заявок | |
| Входные данные | Данные таблиц Request, Task, Location |
| Выходные данные | Список имеющихся заявок |
| Предусловия | Руководитель должен быть авторизован и находится на странице Technical staff manager. Нажатие на кнопку «Заявки» в навигационном меню |
| Постусловия | Вывод имеющихся заявок в виде таблицы |
| Последовательность действий | Получение данных из таблиц Request, Location и Task базы данных. Вывод на экран всех заявок в виде таблицы |

Таблица 2.20 – Удаление заявки

|  |  |
| --- | --- |
| Удаление заявки | |
| Входные данные | Id заявки |
| Выходные данные | - |
| Предусловия | Руководитель должен быть авторизован. Нажатие на иконку с корзиной в таблице заявок на странице Request management |
| Постусловия | Заявка удалена |
| Последовательность действий | Нахождение и удаление записи в таблице request по Id. Удаления заявки из общего списка |

Таблица 2.21 – Экспорт списка заявок

|  |  |
| --- | --- |
| Экспорт списка заявок | |
| Входные данные | Данные таблиц Request, Location, Task и WorkTimeLog |
| Выходные данные | Файл со списком заявок (Excel или PDF) |
| Предусловия | Руководитель должен быть авторизован. Нажатие на кнопку «Excel» или «PDF» под таблицей с заявками |
| Постусловия | Получение файла со списком заявок |
| Последовательность действий | Получение данных из таблиц Request, Location, Task и WorkTimeLog базы данных. Нажатие на кнопку. Получение файла со списком заявок. |

Таблица 2.22 – Просмотр статистики сотрудника

|  |  |
| --- | --- |
| Просмотр статистики сотрудника | |
| Входные данные | Данные таблиц WorkTimeLog, Request, Task, EmployeeOfTechnicalStaff, User |
| Выходные данные | Таблица выполненных заявок сотрудника и календарь с отображением процесса создания заявки и выполнения её сотрудником за указанный промежуток времени |
| Предусловия | Руководитель должен быть авторизован. Нажатие на текст «Статистика» в таблице с работниками на странице Technical staff manager |
| Постусловия | Отображение данных |
| Последовательность действий | Нажатие на текст «Статистика». Получение данных из таблиц WorkTimeLog, Task, Request, EmployeeOfTechnicalStaff, User базы данных. Вывод на экран выбранной информации. |

Таблица 2.23 – Экспорт календаря выполнения заявок

|  |  |
| --- | --- |
| Экспорт календаря выполнения заявок | |
| Входные данные | Данные таблицы WorkTimeLog, Request, Task |
| Выходные данные | Изображение с календарем |
| Предусловия | Руководитель должен быть авторизован и находится на странице Statistic. Нажатие на кнопку «Сделать снимок» под календарём |
| Постусловия | Получение изображения с календарем |
| Последовательность действий | Получение данных таблиц WorkTimeLog, Request, Task. Нажатие на кнопку. Получение изображения с календарем |

Таблица 2.24 – Назначение заявки сотруднику

|  |  |
| --- | --- |
| Назначение заявки сотруднику | |
| Входные данные | Данные таблиц Request, Task, Location, EmployeeOfTechnicalStaff, User |
| Выходные данные | - |
| Предусловия | Руководитель должен быть авторизован и находится на странице Technical staff manager. Нажатие на текст «Назначит» в таблице работников технического персонала. Нажатие на Id заявки и на кнопку «Назначить» в модальном окне с подробным описанием заявки |
| Постусловия | Сотруднику назначается выбранная заявка |
| Последовательность действий | Получение данных из таблиц Request, Task, Location, EmployeeOfTechnicalStaff, User базы данных для отображения заявок. Нажатие на кнопку «Назначить» |

2.3 Прототипирование интерфейса взаимодействия с пользователем (графического интерфейса)

Макет страницы Employee of technical staff представлен на рисунке 2.3. В верхней левой части располагаются ссылки для перехода на другие страницы, а справа – статус занятости сотрудника, который можно изменить и кнопка «Выйти» для перехода на страницу Authorization. Раздел «Текущие заявки» содержит таблицу с заявками, которые назначены сотруднику для выполнения (со статусом «В процессе»). Ниже расположен раздел «Выполненные заявки», где отображается таблица с информацией о завершенных заявках.

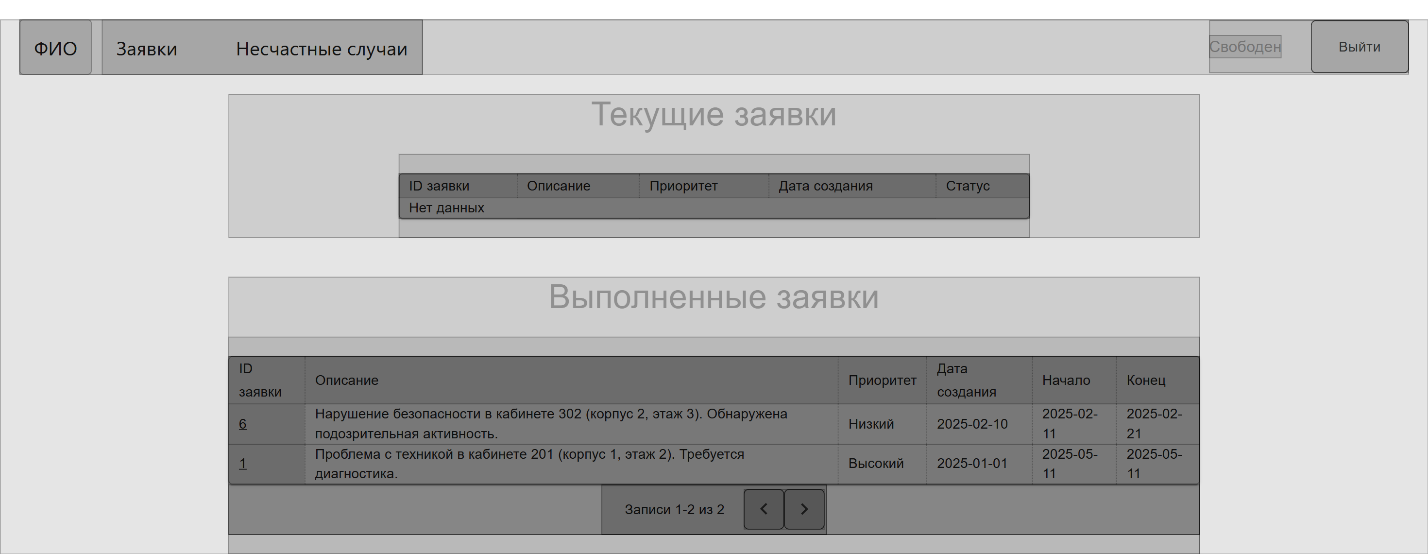


Рисунок 2.3 – Макет страницы Employee of technical staff

Макет страницы Request management представлен на рисунке 2.4. Страница управления заявками содержит инструменты для поиска и фильтрации заявок. Список заявок представлен в виде таблицы с колонками. В последнем столбце каждой строки есть кнопка для удаления заявки. Внизу страницы расположены элементы навигации для переключения между страницами, а также кнопки для экспорта данных в форматы Excel и PDF и кнопка «Назад» для возврата к предыдущей странице.

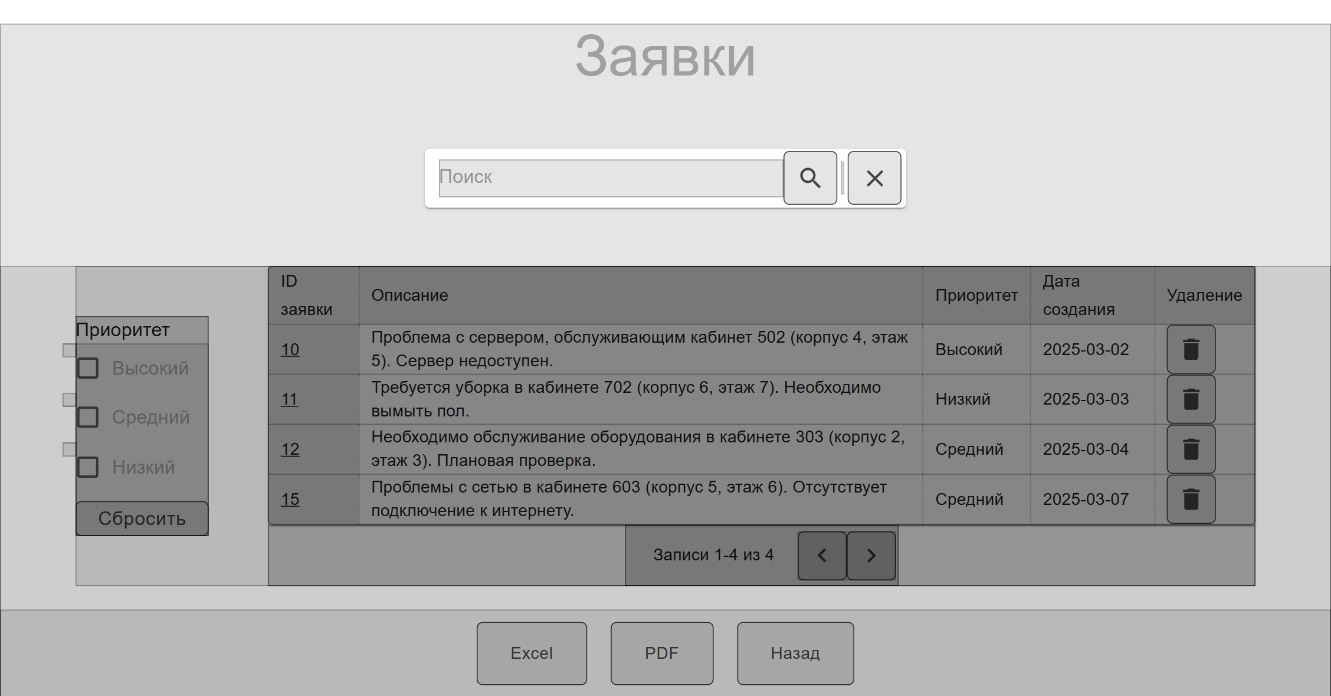


Рисунок 2.4 – Макет страницы Request management

На рисунке 2.5 представлен макет страницы Statistic. Страница Statistic позволяет руководителю просматривать информацию о завершенных заявках выбранного работника, а также период выполнения каждой завершенной заявки. В центре страницы расположена таблица с информацией о выполненных заявках и календарь. Календарь отображает месяц и выделяет даты, когда выполнялись заявки, предоставляя визуальное представление о загруженности. Внизу страницы находятся кнопки для экспорта данных, а также кнопка «Сделать снимок», которая позволяет сохранить текущий вид календаря в виде изображения.

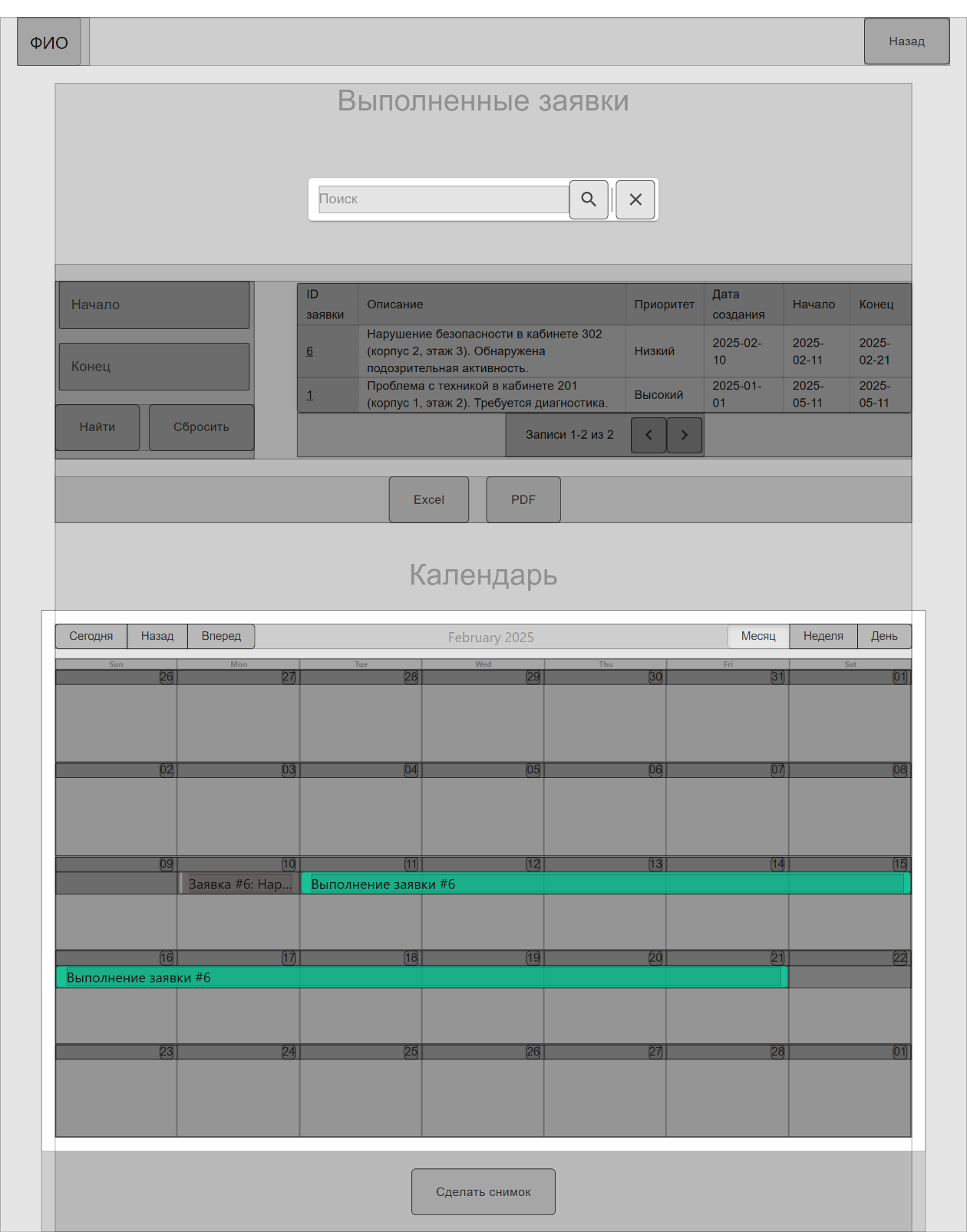


Рисунок 2.5 – Макет страницы Statistic

Макет страницы Full info для руководителя представлен на рисунке 2.6. В верхней части страницы находится блок с личными данными сотрудника, включающий фотографию. Ниже расположен раздел «Текущие заявки», представленный в виде таблицы. В нижней части страницы расположены кнопки «Excel», «PDF» и «Назад» для экспорта данных и навигации.

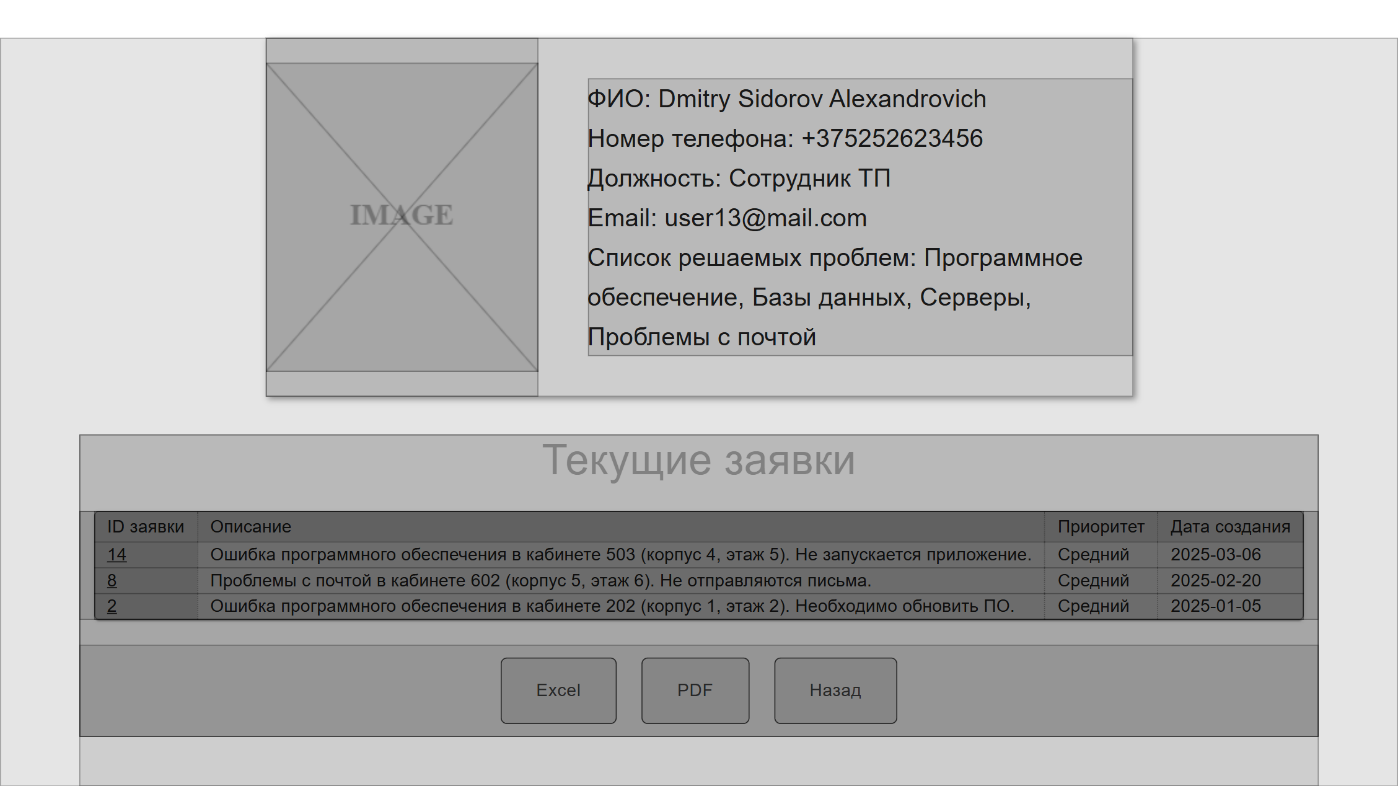


Рисунок 2.6 – Макет страницы Full info для руководителя

На рисунке 2.7 представлен макет страницы Full info для работника. Эта страница отличается тем, что не содержит кнопки для экспорта данных

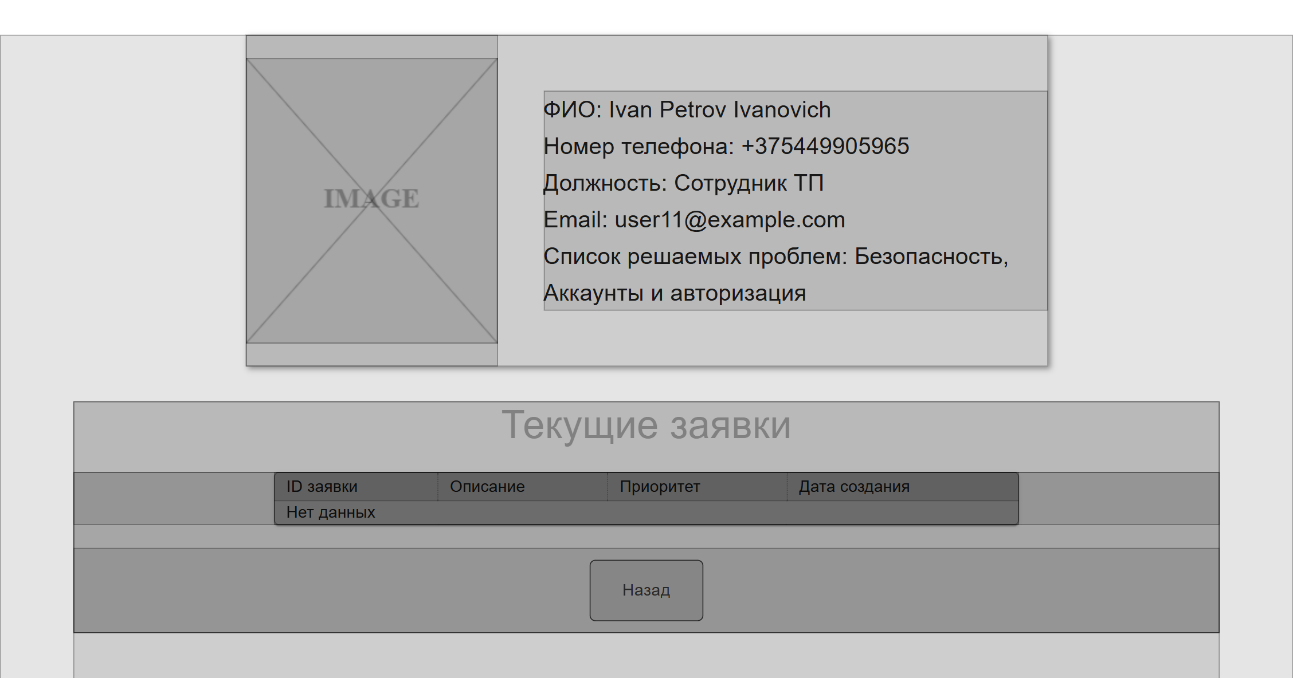


Рисунок 2.7 – Макет страницы Full info для работника

На рисунке 2.8 представлен макет страницы Incidents для руководителя. В верхней части страницы расположен заголовок, а также поле для поиска несчастный случаев. Основная часть страницы представлена таблицей, с возможностью удаления записи несчастного случая. В нижней части отображается информация о количестве записей и элементы навигации для переключения между страницами с инцидентами. Также внизу расположены кнопки «Добавить», «Excel», «PDF» и «Назад» для добавления нового инцидента, экспорта данных и возврата к предыдущей странице соответственно.

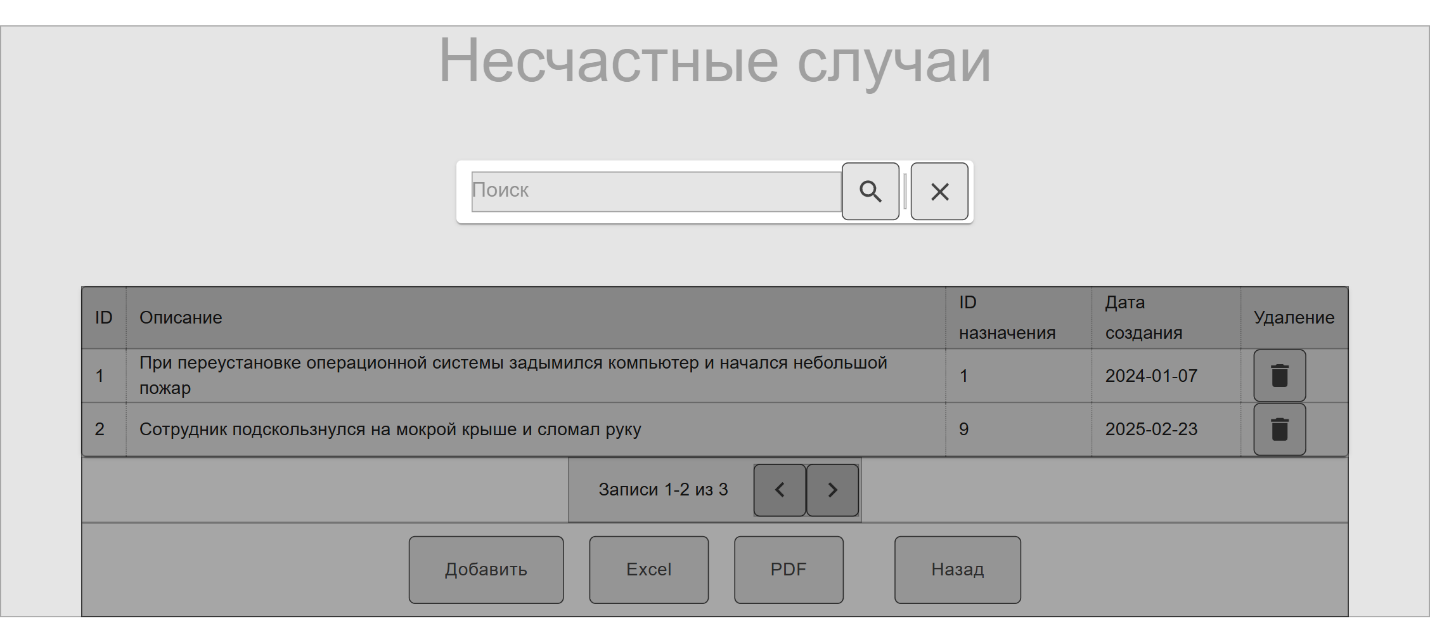


Рисунок 2.8 – Макет страницы Incidents для руководителя

На рисунке 2.9 представлен макет страницы Incidents для работника. Эта страница отличается тем, что не содержит кнопок «Добавить», «Excel» и «PDF»

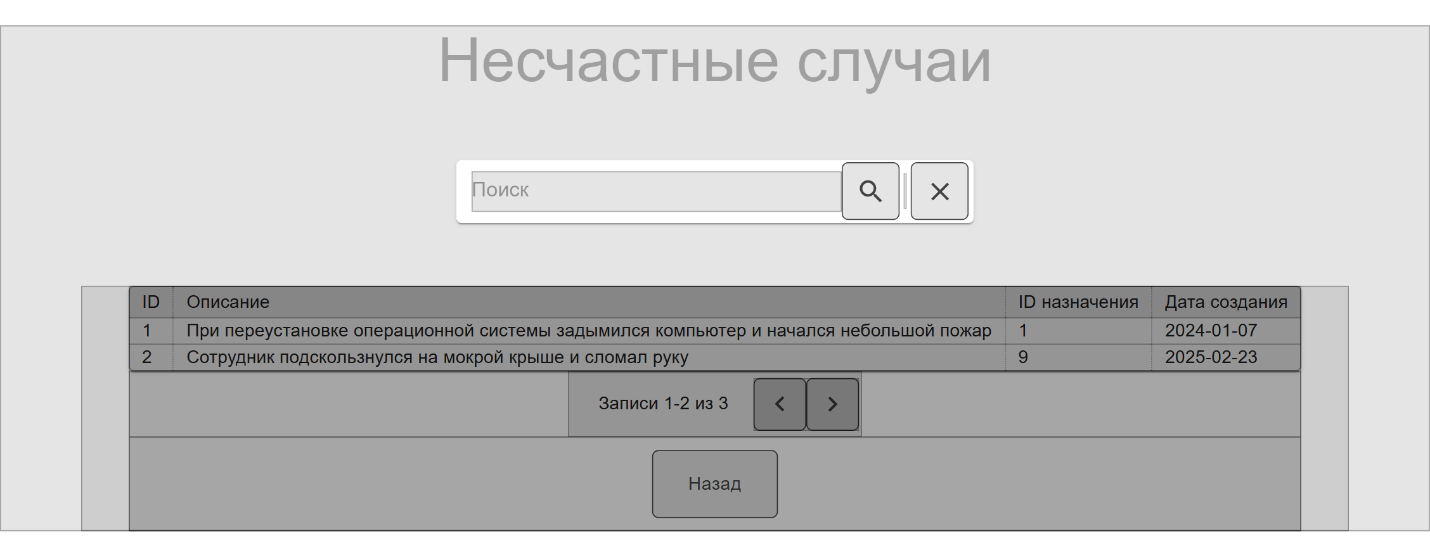


Рисунок 2.9 – Макет страницы Incidents для работника

На рисунке 2.10 представлен макет страницы Technical staff manager. В верхней части страницы расположены ссылки для навигации, и кнопка «Выйти» для выхода из аккаунта. Основная часть страницы представлена таблицей, содержащей информацию о сотрудниках. В нижней части страницы отображается информация о количестве записей и элементы навигации для переключения между страницами со списком сотрудников. Также внизу расположены кнопки для экспорта данных в соответствующие форматы.

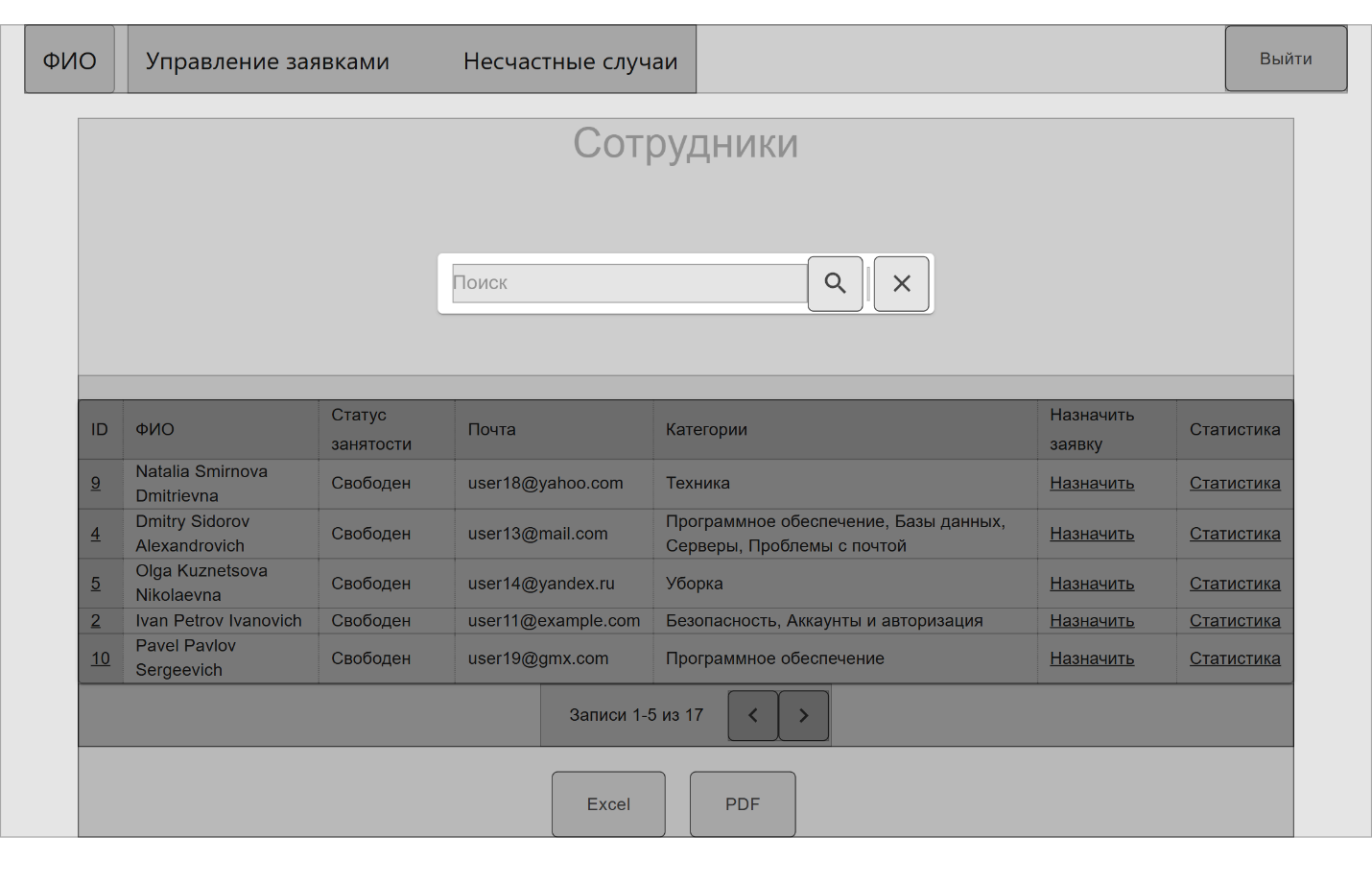


Рисунок 2.10 – Макет страницы Technical staff manager

На рисунке 2.11 представлен макет страницы Requests для руководителя. В верхней части страницы расположен заголовок «Заявки», а также поле для поиска заявок. Слева находится панель фильтров по приоритету. Также есть кнопка «Сбросить» для сброса фильтров. Основная часть страницы представлена таблицей, содержащей информацию о заявках. Под таблицей находится кнопка «Отобразить все заявки». В нижней части страницы отображается информация о количестве записей и элементы навигации. Также внизу расположены кнопки для экспорта данных в форматы Excel и PDF, и возврата к предыдущей странице.

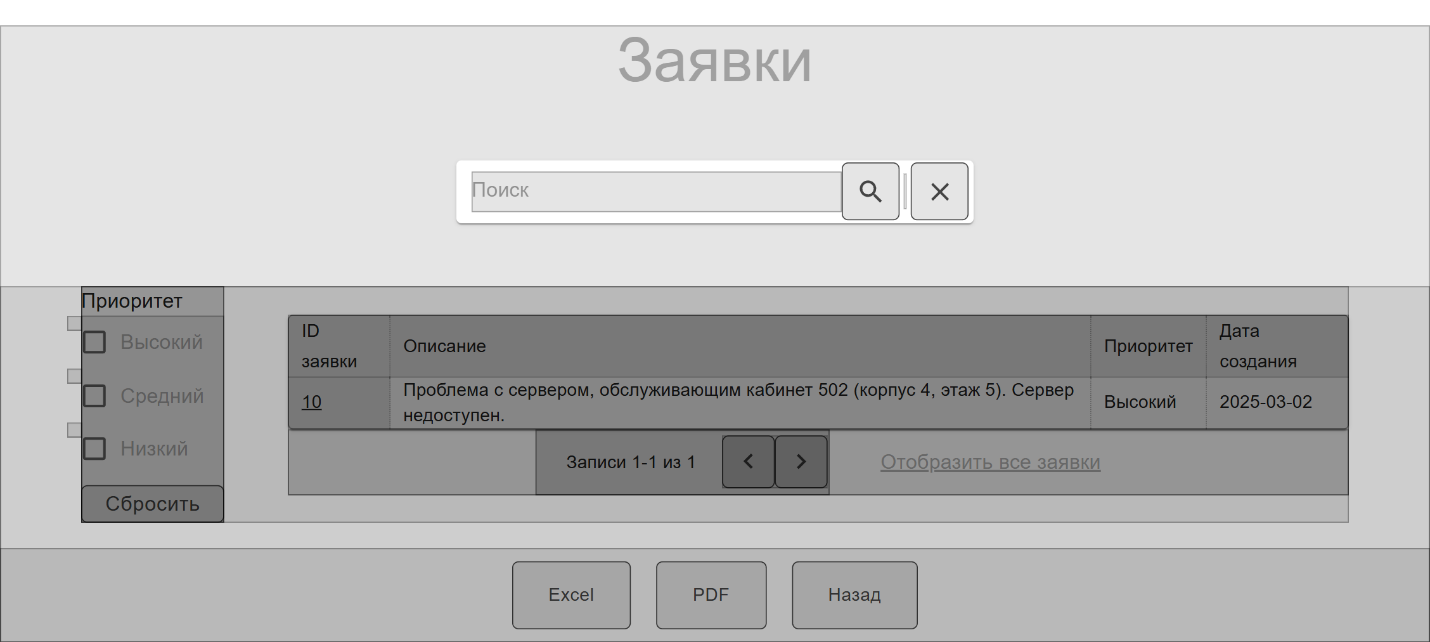


Рисунок 2.11 – Макет страницы Requests для руководителя

На рисунке 2.12 представлен макет модального окна для просмотра подробной информации о заявке для руководителя.

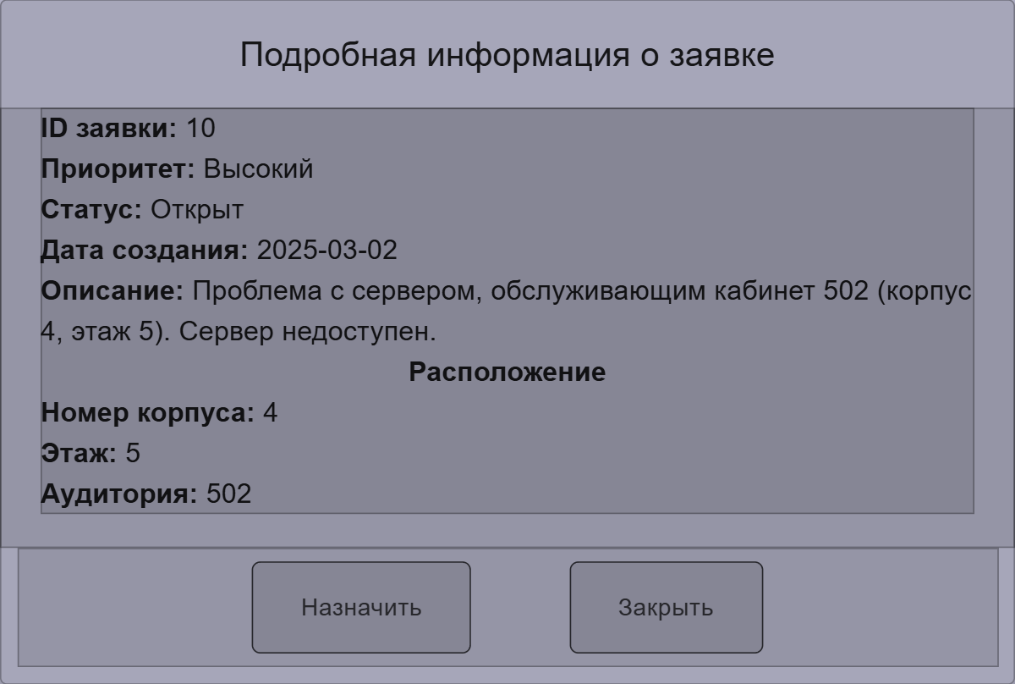


Рисунок 2.12 – Макет модального окна для просмотра подробной информации о заявки для руководителя.

Макет модального окна для просмотра подробной информации о заявке для сотрудника отличается тем, что вместо кнопки «Назначить» располагается кнопка «Добавить».

На рисунке 2.13 представлен макет страницы Requests для сотрудника. Эта страница отличается от страницы Requests для руководителя тем, что не содержит кнопок для экспорта данных.

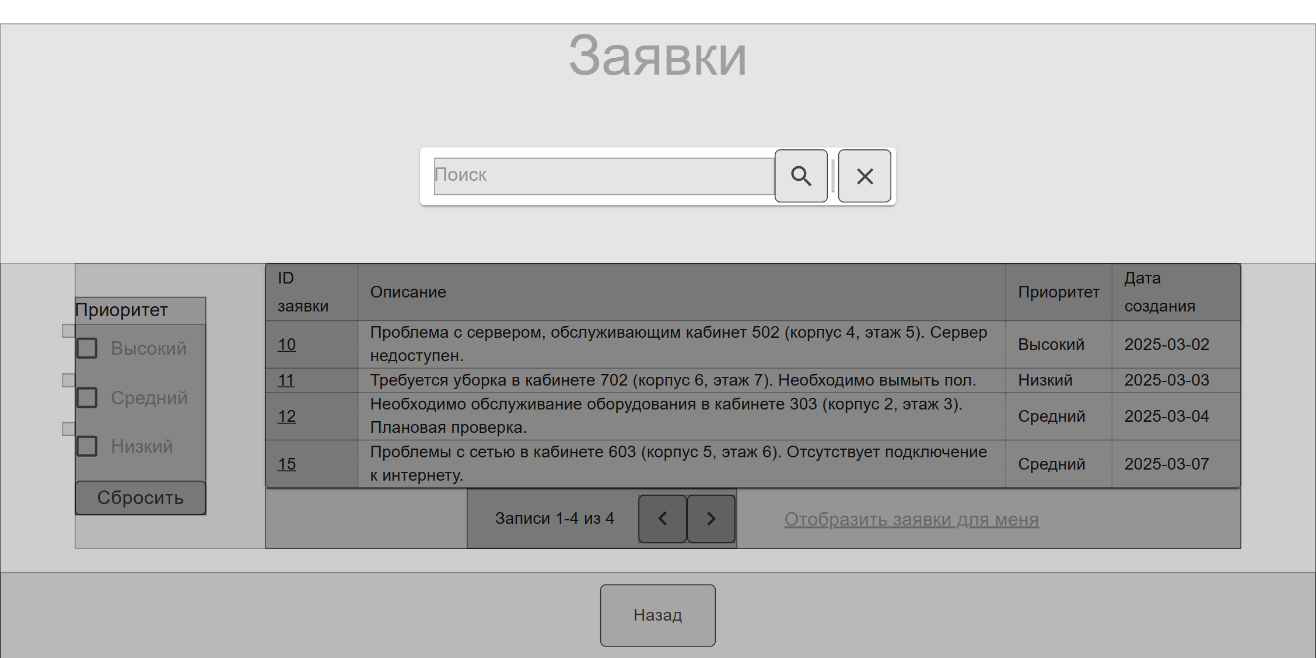


Рисунок 2.13 – Макет страницы Requests для работника

На рисунке 2.14 представлен макет страницы Add incident. Основная часть страницы представлена формой, которая содержит поля для ввода данных о новом несчастном случае. Под формой находятся кнопки «Добавить» и «Назад».

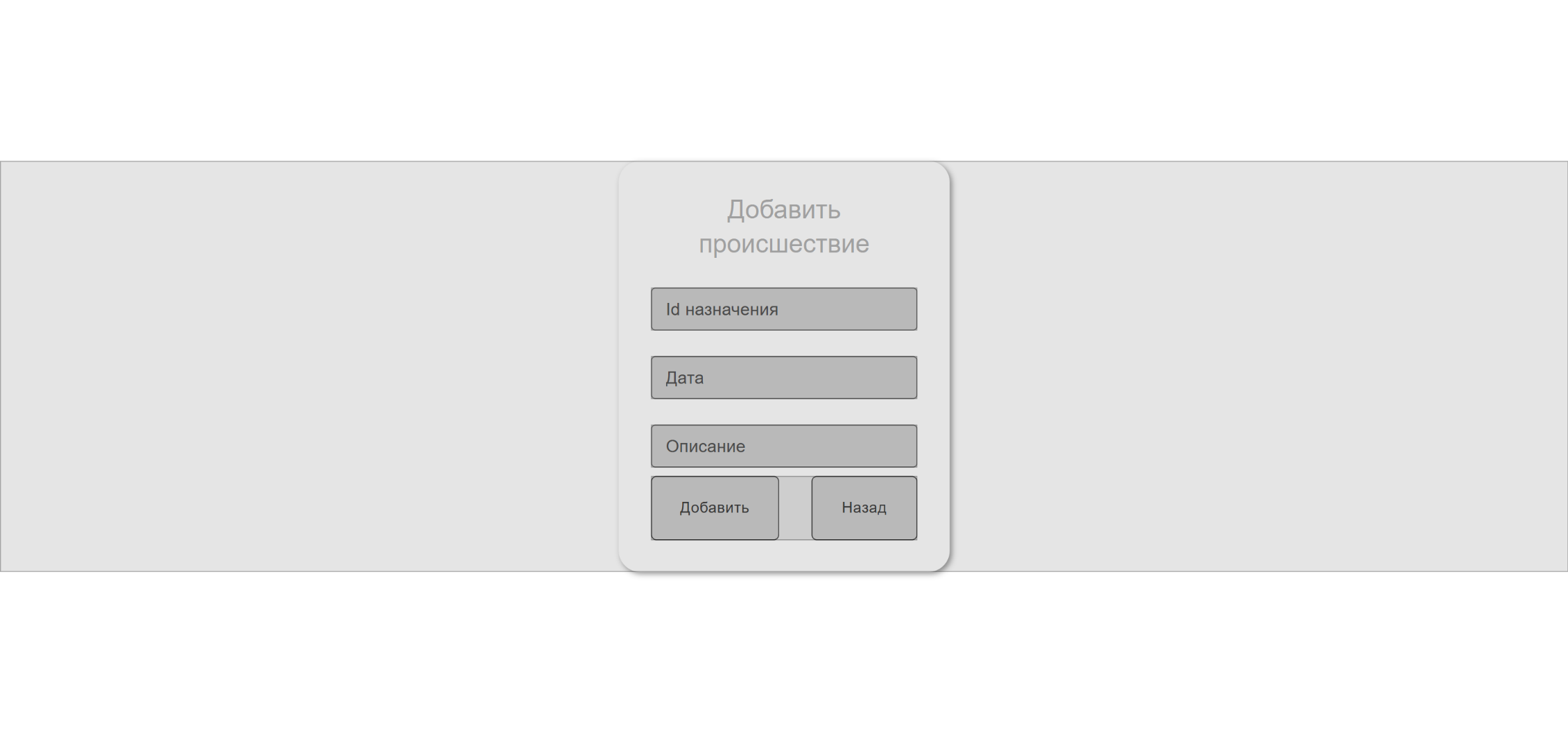


Рисунок 2.14 – Макет страницы Add incident

2.4 Составление плана работы

Разработка приложения осуществлялась поэтапно. Наибольшую сложность представляли два основных направления: создание пользовательского интерфейса (frontend) и разработка серверной части (backend). Формирование функционального интерфейса потребовало значительных временных затрат на проектирование UX/UI, реализацию интерактивных элементов и обеспечение адаптивности для различных устройств. Не менее трудоемкой оказалась работа над серверной частью, которая включала проектирование архитектуры API, разработку сложной бизнес-логики и обеспечение стабильного взаимодействия с базой данных. Все этапы разработки, их сроки и последовательность выполнения подробно представлены в таблице 2.25, где указаны даты начала и завершения каждого из ключевых этапов проекта.

Таблица 2.25 – Общий план работы

|  |  |  |
| --- | --- | --- |
| Название действий | Дата начала | Дата окончания |
| Изучение технологии React | 14.02.2025 | 14.03.2025 |
| Изучение основ Node.js | 15.03.2025 | 20.03.2025 |
| Изучение фреймворка Express | 21.03.2025 | 28.03.2025 |
| Разработка backend | 29.03.2025 | 15.04.2025 |
| Разработка frontend | 16.04.2025 | 30.04.2025 |
| Тестирование проекта и исправление багов | 01.05.2025 | 05.05.2025 |
| Подготовка документации | 05.05.2025 | 15.05.2025 |

Индивидуальный план работы представлен в таблице 2.26.

Таблица 2.26 – Индивидуальный план работы

|  |  |  |
| --- | --- | --- |
| Название действий | Дата начала | Дата окончания |
| Изучение технологии React | 14.02.2025 | 20.02.2025 |
| Изучение основ Node.js | 14.02.2025 | 25.02.2025 |
| Изучение фреймворка Express | 26.02.2025 | 28.02.2025 |
| Изучение функций экспорта в PDF и Excel | 01.03.2025 | 03.03.2025 |
| Реализация функций отображения заявок по решаемым проблемам и отображения всех заявок | 03.03.2025 | 07.03.2025 |
| Реализации функции принятия и назначения заявок | 07.03.2025 | 09.03.2025 |

Продолжение таблицы 2.26

|  |  |  |
| --- | --- | --- |
| Название действий | Дата начала | Дата окончания |
| Реализации функций обновления статуса заявок и статуса занятости сотрудника | 10.03.2025 | 12.03.2025 |
| Реализация функции отображения сотрудников для руководителя | 13.03.2025 | 16.03.2025 |
| Реализация функций отображения полной информации о сотруднике и заявке | 16.03.2025 | 18.03.2025 |
| Реализация функции получения данных о времени выполнения заявки для определенного сотрудника | 19.03.2025 | 22.03.2025 |
| Реализация функции отображения несчастных случаев | 23.03.2025 | 25.03.2025 |
| Реализация функций фильтрации и поиска | 26.03.2025 | 02.04.2025 |
| Реализация функции добавления несчастного случая | 03.04.2025 | 04.04.2025 |
| Реализация функций удаления несчастного случая и удаления заявки | 05.04.2025 | 07.04.2025 |
| Реализация экспорта в Excel и PDF | 08.04.2025 | 12.04.2025 |
| Тестирование проекта, и исправление багов | 13.04.2025 | 20.04.2025 |
| Подготовка документации | 21.04.2025 | 15.05.2025 |

# 3 Разработка приложения

3.1 Диаграмма классов

UML диаграмма классов слоя доступа к данным представлена на рисунке 3.1

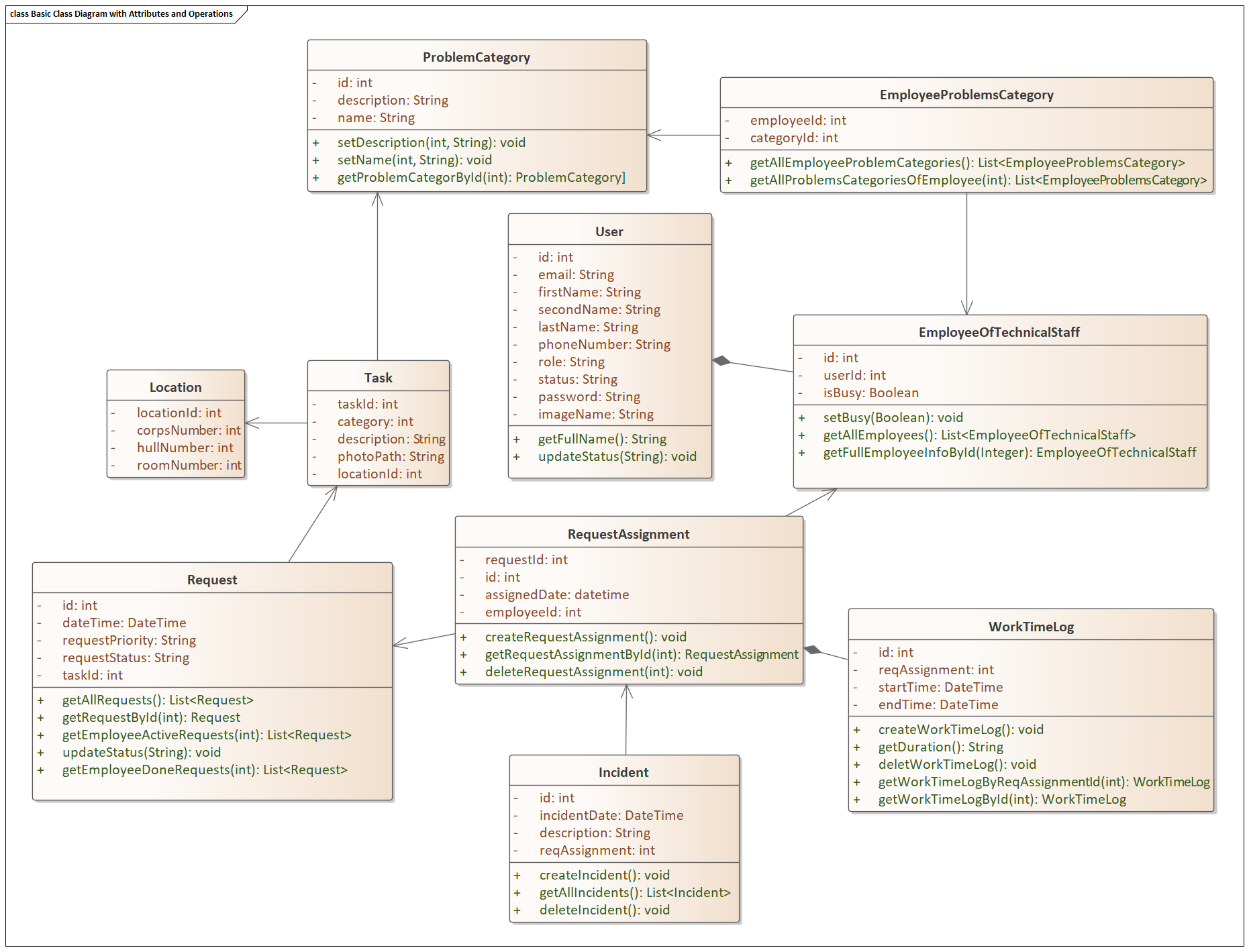


Рисунок 3.1 – UML диаграмма классов слоя доступа к данным

Класс ProblemCategory содержит информацию о категориях проблем. Класс включает следующие атрибуты: id – уникальный идентификатор, description – описание проблемы, name – название категории проблемы. Метод setDescription используется для изменения описания по id. Метод setName используется для изменения названия категории проблемы по ее id. Метод getProblemCategoryById используется для получения категории проблемы по id.

Класс EmployeeProblemsCategory содержит информацию о том, к какому работнику относиться определенная категория проблем. Класс включает следующие атрибуты: employeeId – часть составного первичного ключа, которая указывает на уникальный идентификатор работника, categoryId – часть составного первичного ключа, которая указывает на уникальный идентификатор категории проблем. Метод getAll EmployeeProblemsCategory возвращает все назначения категорий проблем работникам. Метод getAllEmployeeProblemsCategoryOfEmployee возвращает все категории проблем для конкретного работника. Класс имеет ассоциации с классами ProblemCategory и EmployeeOfTechnicalStaff.

Класс EmployeeOfTechnicalStaff содержит информацию о работниках технического персонала. Он включает следующие атрибуты: id – уникальный идентификатор работника, userId – внешний ключ, указывающий на пользователя, isBusy – статус занятости сотрудника. Метод setBusy устанавливает новый статус работника. Метод getAllEmployee возвращает всех работников. Метод getFullEmployeeInfoById возвращает работника по его id. Класс связан композицией с классом User, так как не может существовать без User.

Класс User содержит информацию о пользователях в системе. Он включает следующие атрибуты: email — адрес электронной почты пользователя, firstName — имя, secondName – фамилия, lastName — отчество, password — пароль пользователя в системе, phoneNumber — номер телефона, role — роль пользователя (например, администратор, работника и т.д.), status — текущий статус (например, активен или неактивен), imageName – название фотографии с пользователем. Метод getFullName возвращает полное имя. Метод updateStatus обновляет статус пользователя.

Класс Request содержит информацию о заявках. Он включает следующие атрибуты: id – уникальный идентификатор заявки, dateTime- дата создания заявки, requestPriority – приоритет заявки, requestStatus – статус заявки, taskId – внешний ключ, указывающий на задачу для заявки. Метод getAllRequests возвращает все заявки. Метод getRequestById возвращает заявку по ее Id. Метод getEmployeeActiveRequests возвращает заявки, которые выполняет сотрудник с указанным id. Метод getEmployeeDoneRequests возвращает заявки, выполненные сотрудником с указанным id. Метод deleteRequest удаляет заявку с указанным id. Класс имеет ассоциацию с классом Task.

Класс Task содержит информацию о задачах. Он включает следующие атрибуты: taskId – уникальный идентификатор задачи, category – внешний ключ, указывающий на категорию проблем, photoPath – название фотографии с задачей, description – описание задачи, locationId – внешний ключ, указывающий на локацию, где произошла проблема. Класс имеет ассоциации с классами ProblemCategory и Location.

Класс Location содержит информацию о местоположениях. Он включает следующие атрибуты: locationId – уникальный идентификатор местоположения, corpsNumber – номер корпуса, hullNumber – номер этажа, roomNumber – номер аудитории

Класс RequestAssignment содержит информацию о назначении заявок сотрудникам. Он включает следующие атрибуты: id – уникальный идентификатор назначения, assignedDate – дата назначения, employeeId – внешний ключ, указывающий на работника, который будет выполнять заявку, requestId – внешний ключ, который указывает на заявку, которую будет выполнять работник. Метод createAssignment создает новое назначение. Метод getRequestAssignmentById возвращает назначение по его Id. Метод deleteRequestAssignment удаляет назначение. Класс имеет ассоциациями с классами Request и EmployeeOfTechnicalStaff.

Класс WorkTimeLog содержит информацию о логах времени выполнения заявок. Он включает следующие атрибуты: id – уникальный идентификатор лога, startTime – дата начала выполнения заявки, endTime – дата окончания выполнения заявки, reqAssignment – внешний ключ, указывающий на назначение. Метод createWorkTimeLog создает лог. Метод deleteWorkTimeLog удаляет лог. Метод getDuration получает время выполнения заявки в минутах. Метод getWorkTimeLogById – возвращает лог по его Id. Метод getWorkTimeLogByReqAssignmentId возвращает лог по id назначения. Класс связан композицией с классом RequestAssignment, так как WorkTimeLog не может существовать без RequestAssignment.

Класс Incident содержит информацию о несчастных случаях, которые могли произойти во время выполнения заявки. Он включает следующие атрибуты: id – уникальный идентификатор несчастного случая, incidentDate – дата несчастного случая, description – описание несчастного случая, reqAssignment – внешний ключ, указывающий на назначение. Метод createIncident создает несчастный случай. Метод getAllIncidents – возвращает все несчастные случаи. Метод deleteIncident удаляет несчастный случай. Класс имеет ассоциацию с классом RequestAssignment

На рисунке 3.2 представлена диаграмма классов бизнес логики

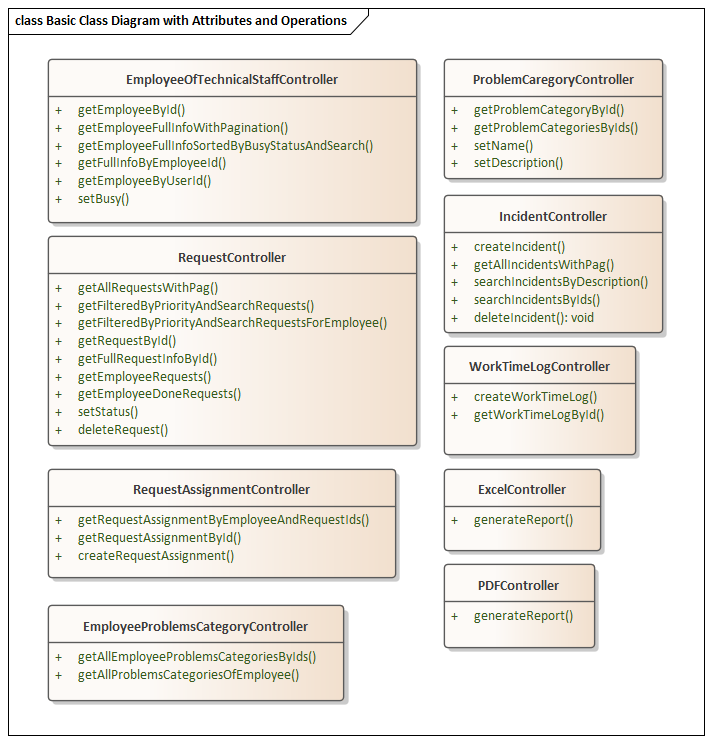


Рисунок 3.2 – Диаграмма классов бизнес-логики

Класс RequestController содержит методы:

* getAllRequestWithPag: получение заявок с пагинацией;
* getFilteredByPriorityAndSearchRequests: получение отфильтрованных по приоритету и найденных по ключевому слову заявок с пагинацией;
* getFilteredByPriorityAndSearchRequestsForEmployee: получение отфильтрованных по приоритету, категориям проблем (которые принадлежат сотруднику с Id, которое передается в контроллер ) и найденных по ключевому слову заявок с пагинацией;
* getRequestById: получение неполной информации о заявки по Id;
* getFullRequestInfoById: получение полной информации о заявке по Id;
* getEmployeeRequests: получение списка выполняемых заявок сотрудником по Id сотрудника;
* getEmployeeDoneRequests: получение списка выполненных заявок сотрудником по Id сотрудника;
* setStatus: изменение статуса заявки по Id заявки;
* deleteRequest: удаление заявки по ее Id.

Класс EmployeeOfTechnicalStaffController содержит методы:

* getEmployeeById: получение работника оп его Id;
* getEmployeesFullInfoWithPagination: получение списка работник с подробной информацией о каждом работника с пагинацией;
* getEmployeesFullInfoSorteredByBusyStatusAndSearch: получение полной информацию о сотрудниках, сортировка их по статусу занятости и фильтрация по поисковому запросу;
* getFullInfoByEmployee: получение полной информации о сотруднике по его Id;
* getEmployeeByUserId: получение работника по Id пользователя;
* setBusy: изменение статуса занятости сотрудника по его Id.

Класс RequestAssignmentController содержит методы:

* createRequestAssignment: создание нового назначения;
* getRequestAssignmentByEmployeeAndRequestIds: получение назначения по Id сотрудника и Id заявки;
* getRequestAssignmentById: получение назначения по его Id.

Класс EmployeeProblemsCategoryController содержит методы:

* getAllProblemsCategoriesOfEmployee: получение всех проблем, которые решает сотрудник по его Id;
* getEmployeeProblemsCategoryByIds: получение проблемы, которая относиться к сотруднику по Id проблемы и Id сотрудника;

Класс ProblemCategoryController содержит методы:

* getProblemCategoryById: получение категории проблемы по ее Id;
* getProblemCategoriesByIds: получение списка категорий по нескольким Id категорий проблем;
* setName: изменить название категории проблемы по ее Id;
* setDescription: изменить описание категории проблемы по ее Id;

Класс IncidentController содержит методы:

* createIncident: создание несчастного случая;
* getAllIncidentsWithPag: получение несчастных случаев с пагинацией;
* searchIncidentsByDescription: получение несчастных случаев с пагинацией, поиск по описание по ключевому слову;
* searchIncidentsByIds: получение несчастных случаев с пагинацией, поиск по Id несчастного случая или по Id назначения;
* deleteIncident: удаление несчастного случая по его Id.

Класс WorkTimeLogController содержит методы:

* createWorkTimeLog: создание лога времени выполнения заявки;
* getWorkTimeLogById: получение лога времени выполнения заявки по его Id.

Класс ExcelController содержит метод generateReport для генерации отчета в Excel.

Класс PDFController содержит метод generateReport для генерации отчета в PDF.

Диаграмма классов пользовательского интерфейса представлена на рисунке 3.3

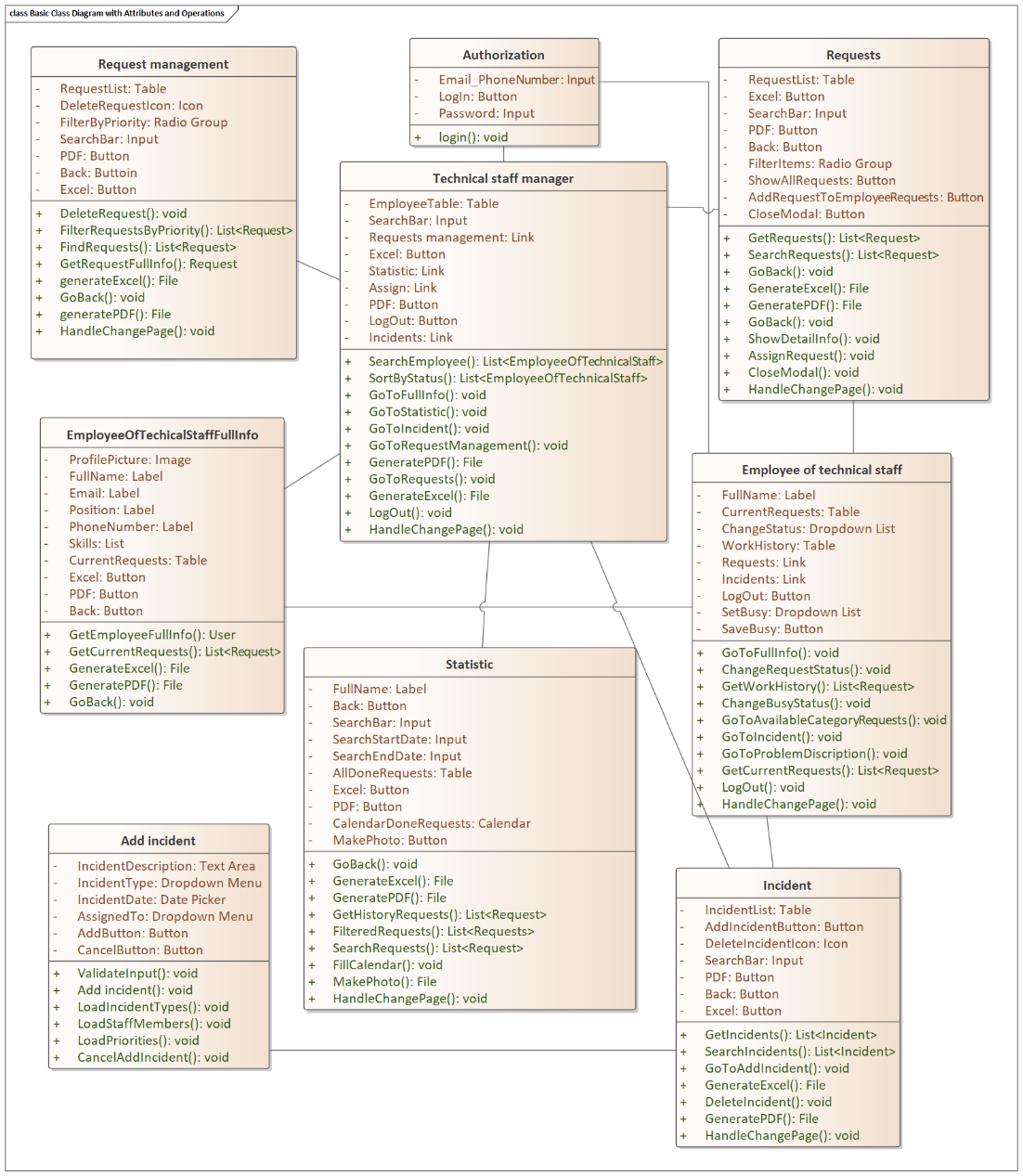


Рисунок 3.3 – Диаграмма классов пользовательского интерфейса

Класс Authorization. Атрибуты:

* Email\_PhoneNumber (Input): поле ввода для Email или номера телефона;
* Password (Input): поле ввода для пароля.

Метод класса:

* Login(): выполнение входа в систему.

Класс Technical Staff Manager. Атрибуты:

* EmployeeTable (Table): таблица со списком сотрудников;
* SearchBar (Input): поле поиска;
* Requests management (Link): ссылка на управление заявками;
* Excel (Button): кнопка для экспорта в Excel;
* Statistic (Link): ссылка на статистику;
* Assign (Link): ссылка на назначение заявок;
* PDF (Button): кнопка для экспорта в PDF;
* LogOut (Button): кнопка для выхода из системы;
* Incidents (Link): ссылка на инциденты;

Методы класса:

* SearchEmployee(): поиск сотрудников;
* SortByStatus(): сортировка сотрудников по статусу;
* GoToFullInfo(): переход к полной информации о сотруднике;
* GoToStatistic(): переход к статистике;
* GoToIncident(): переход к инцидентам;
* GoToRequestManagement(): переход к управлению заявками;
* GeneratePDF(): генерация PDF-файла;
* GoToRequests(): переход к заявкам;
* GenerateExcel(): генерация Excel-файла;
* LogOut(): выход из системы;
* HandleChangePage(): обработка смены страницы с сотрудниками.

Класс EmployeeOfTechicalStaffInfo. Атрибуты:

* ProfilePicture (Image): изображение профиля;
* FullName (Label): метка с полным именем;
* Email (Label): метка с email;
* Position (Label): метка с должностью;
* PhoneNumber (Label): метка с номером телефона;
* Skills (List): список категорий решаемых проблем;
* CurrentRequests (Table): таблица с текущими заявками;
* Excel (Button): кнопка для экспорта в Excel;
* PDF (Button): кнопка для экспорта в PDF;
* Back (Button): кнопка «Назад».

Методы класса:

* GetEmployeeFullInfo(): получение полной информации о сотруднике;
* GetCurrentRequests(): получение текущих заявок;
* GenerateExcel(): генерация Excel-файла;
* GeneratePDF(): генерация PDF-файла;
* GoBack(): возврат назад.

Класс Request Management. Атрибуты:

* RequestList (Table): таблица со списком заявок;
* DeleteRequestIcon (Icon): иконка для удаления заявки;
* FilterByPriority (Radio Group): группа радиокнопок для фильтрации по приоритету;
* SearchBar (Input): поле поиска;
* PDF (Button): кнопка для экспорта в PDF;
* Back (Button): кнопка «Назад»;
* Excel (Button): кнопка для экспорта в Excel.

Методы класса:

* DeleteRequest(): удаление заявки;
* FilterRequestsByPriority(): фильтрация заявок по приоритету;
* FindRequests(): поиск заявок;
* GetRequestFullInfo(): получение полной информации о заявке.
* GenerateExcel(): генерация Excel-файла;
* GoBack(): возврат назад;
* GeneratePDF(): генерация PDF-файла;
* HandleChangePage(): обработка смены страницы с заявками.

Класс Requests. Атрибуты:

* RequestList (Table): таблица со списком заявок;
* Excel (Button): кнопка для экспорта в Excel;
* SearchBar (Input): поле поиска;
* PDF (Button): кнопка для экспорта в PDF;
* Back (Button): кнопка «Назад»;
* FilterItems (Radio Group): группа радиокнопок для фильтрации элементов;
* ShowAllRequests (Button): кнопка для отображения всех заявок;
* AddRequestToEmployeeRequests (Button): кнопка для добавления заявки сотруднику;
* CloseModal (Button): кнопка для закрытия модального окна.

Методы класса:

* GetRequests(): получение списка заявок;
* SearchRequests(): поиск заявок;
* GoBack(): возврат назад;
* GenerateExcel(): генерация Excel-файла;
* ShowDetailInfo(): отображение детальной информации;
* AssignRequest(): назначение заявки;
* CloseModal(): закрытие модального окна;
* HandleChangePage(): обработка смены страницы с заявками.

Класс Employee of technical staff. Атрибуты:

* FullName (Label): метка с полным именем;
* CurrentRequests (Table): таблица с текущими заявками;
* ChangeStatus (Dropdown List): выпадающий список для изменения статуса заявки;
* WorkHistory (Table): таблица с историей работы;
* Requests (Link): ссылка на заявки;
* Incidents (Link): ссылка на инциденты;
* LogOut (Button): кнопка для выхода из системы;
* SetBusy (Dropdown List): выпадающий список для установки занятости.

Методы класса:

* GoToFullInfo(): переход к полной информации;
* ChangeRequestStatus(): изменение статуса заявки;
* GetWorkHistory(): получение истории работы;
* ChangeBusyStatus(): изменение статуса занятости;
* GoToAvailableCategoryRequests(): переход к доступным по категориям проблем заявкам;
* GoToIncidentDiscription(): переход к описанию несчастного случая;
* GetCurrentRequests(): получение текущих заявок;
* LogOut(): выход из системы;
* HandleChangePage(): обработка смены страницы с заявками;

Класс Statistic. Атрибуты:

* FullName (Label): метка с полным именем;
* Back (Button): кнопка «Назад»;
* SearchBar (Input): поле поиска;
* SearchStartDate (Input): поле ввода для даты начала поиска;
* SearchEndDate (Input): поле ввода для даты окончания поиска;
* AllDoneRequests (Table): таблица со всеми выполненными заявками;
* Excel (Button): кнопка для экспорта в Excel;
* PDF (Button): кнопка для экспорта в PDF;
* CalendarDoneRequests (Calendar): календарь выполненных заявок;
* MakePhoto (Button): кнопка для создания фотографии календаря.

Методы класса:

* GoBack(): возврат назад;
* GenerateExcel(): генерация Excel-файла;
* GeneratePDF(): генерация PDF-файла;
* GetHistoryRequests(): получение выполненных заявок;
* FilteredRequests(): фильтрация заявок;
* SearchRequests(): поиск заявок;
* FillCalendar(): заполнение календаря;
* MakePhoto(): создание фотографии;
* HandleChangePage(): обработка смены страницы с заявками.

Класс Add Incident. Атрибуты:

* IncidentDescription (Text Area): текстовое поле для описания инцидента;
* IncidentDate (Date Picker): Выбор даты инцидента;
* AssignedTO (Text Area): текстовое поле для указания номера назначения;
* AddButton (Button): кнопка «Добавить»;
* CancelButton (Button): кнопка «Отмена».

Методы класса:

* ValidateInput(): проверка введенных данных;
* AddIncident(): добавление инцидента;
* CancelAddIncident(): отмена добавления инцидента.

Класс Incident. Атрибуты:

* IncidentList (Table): таблица со списком инцидентов;
* AddIncidentButton (Button): кнопка «Добавить инцидент»;
* DeleteIncidentIcon (Icon): иконка для удаления инцидента;
* SearchBar (Input): поле поиска;
* PDF (Button): кнопка для экспорта в PDF;
* Back (Button): кнопка «Назад»;
* Excel (Button): кнопка для экспорта в Excel.

Методы класса:

* GetIncidents(): получение списка инцидентов;
* SearchIncidents(): поиск инцидентов;
* GoToAddIncident(): переход к добавлению инцидента;
* GenerateExcel(): генерация Excel-файла;
* DeleteIncident(): удаление инцидента;
* GeneratePDF(): генерация PDF-файла;
* HandleChangePage(): обработка смены страницы с несчастными случаями.
  1. Диаграмма деятельности

Диаграмма деятельности просмотра доступных по категориям проблем заявок для сотрудника представлена на рисунке 3.4.

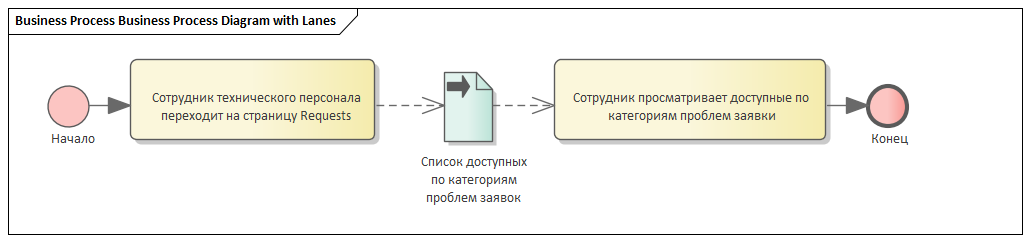


Рисунок 3.4 – Диаграмма деятельности просмотра доступных по категориям проблем заявок для сотрудника

Сотрудник технического персонала переходит на страницу Requests, система выводит список доступных по категориям проблем заявок, сотрудник просматривает заявки.

Диаграмма деятельности добавления заявки в список выполняемых заявок сотрудника представлена на рисунке 3.5.

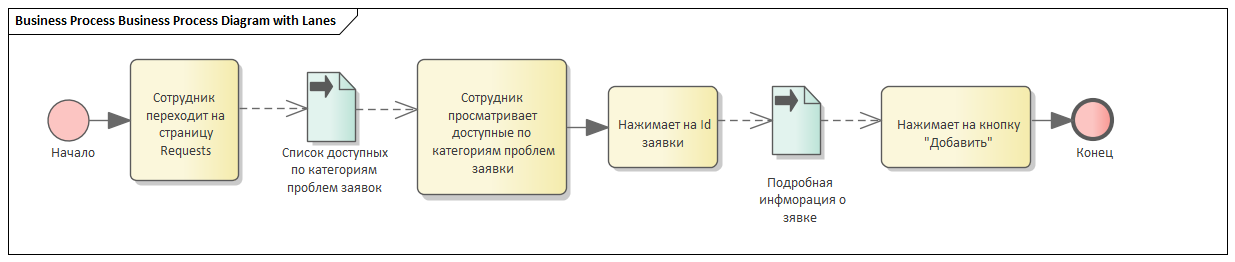


Рисунок 3.5 – Диаграмма деятельности добавления заявки в список выполняемых заявок сотрудника

Диаграмма описывает процесс добавления заявки сотрудником в список выполняемых заявок. Сотрудник переходит на страницу Requests, после чего он просматривает заявки, нажимает на Id конкретной заявки, просматривает всю информацию, касаемо этой заявки и нажимает на кнопку «Добавить». Система убирает выбранную заявку из общего списка и добавляет ее в список заявок сотрудника.

Диаграмма деятельности экспорта списка сотрудников в файл представлена на рисунке 3.6

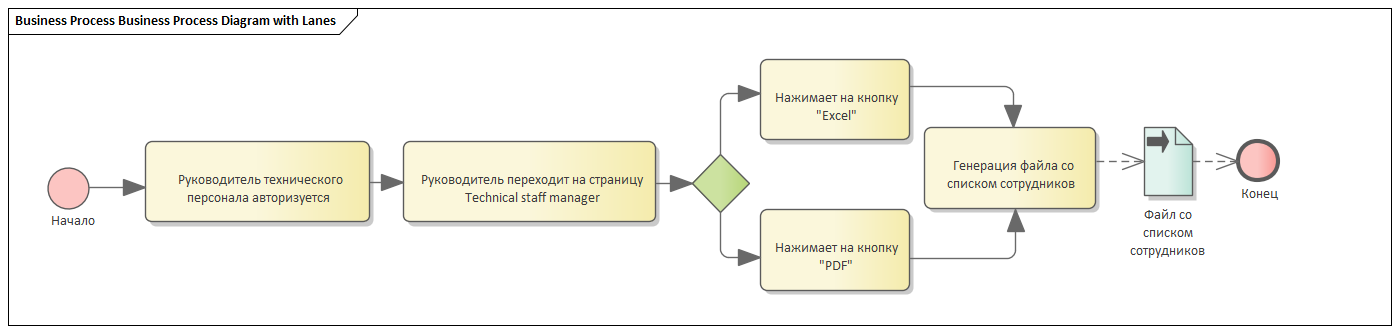


Рисунок 3.6 – Диаграмма деятельности экспорта списка сотрудников в файл

Процесс начинается с авторизации руководителя технического персонала, который затем переходит на страницу Technical staff manager, выбирает одну из двух кнопок генерации файла: «Excel» или «PDF», после чего система формирует файл выбранного формата.

Диаграмма деятельности добавления несчастного случая в систему представлена на рисунке 3.7

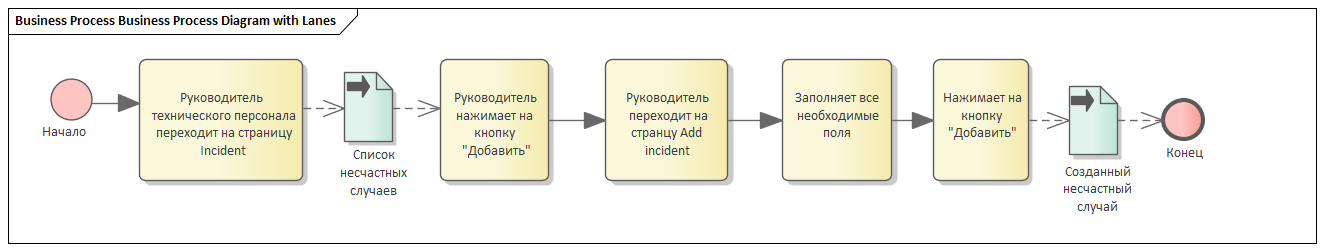


Рисунок 3.7 – Диаграмма последовательности добавления несчастного случая в систему

Диаграмма описывает процесс создания нового несчастного случая: руководитель технического персонала переходит на страницу Incident, где нажимает кнопку «Добавить». Затем система перенаправляет его на страницу Add incident. Руководитель заполняет необходимые поля в форме добавления и подтверждает действие кнопкой «Добавить», после чего система сохраняет новый несчастный случай в базе данных и завершает процесс.

# 4 Разработка документации приложения

4.1 Выбор и обоснование технологий, используемых в программном модуле

Для разработки веб-приложения были использованы современные технологии и инструменты, которые гарантируют простоту и удобство разработки, высокую надежность и возможность масштабирования. Ниже приведено обоснование выбора каждой из ключевых технологий.

В качестве основной среды разработки был выбран бесплатный и мощный редактор кода – Visual Studio Code. Он сочетает в себе простоту использования с расширенными возможностями для профессиональной разработки.

Клиентская часть разработана с использованием следующих технологий:

* React – используется для создания графического интерфейса веб приложения (таблиц, форм, кнопок и т.д) и управления его состояниями;
* React Router — библиотека, которая используется для маршрутизации внутри приложения. Компонент Router отвечает за назначение переходов между страницами;
* Yup – библиотека, обеспечивающая простую и гибкую валидацию данных при регистрации и добавлении несчастного случая;
* Axios – API для взаимодействия с сервером. Используется для отправки запросов с клиента на сервер используя эндпоинты;
* Formik — библиотека для React, которая используется для создания и управление формами: авторизации и добавления несчастного случая;
* html2canvas– библиотека, которая используется для фотографирования календаря с информацией о выполненных заявках;
* Material-UI — библиотека, которая предоставляет готовые компоненты, которые использовались при разработке WEB-приложения: формы, таблицы, кнопки, текста, ссылки, поля ввода и т.д.
* CSS – используются для создания и изменения стиля элементов веб-страниц и пользовательских интерфейсов, написанных на языке HTML.

Для серверной части веб-приложения были выбраны следующие технологии:

* Express.js (Express) — основной фреймворк для создания сервера;
* bcryptjs – библиотека для хеширования паролей;
* multer – это middleware для Node.js, который использовался для обработки загрузки файлов на сервер;
* cors – библиотека, которая позволила настроить и управлять политикой Cross-Origin Resource Sharing (CORS);
* exceljs, pdfkit-table – библиотеки для создания документов (Excel, PDF) на сервере;
* dotenv — библиотека Node.js, которая используется для загрузки переменных окружения из файла env (пароль БД, название БД и др.).

Для взаимодействия с базой данных и управления данными использовались следующие инструменты:

* PostgreSQL — СУБД, которая используется для управленя и хранения данных веб-приложения;
* Sequelize – ORM для Node.js, которая используется для описания моделей веб-приложения, таких как requests, incidents, problemCategories и т.д.;
* pg – библиотека, которая используется для создания и применения миграций синхронизации структуры моделей сервера веб-приложения и таблиц БД.
  1. Описание используемых структур данных

Диаграмма структуры базы данных представлена на рисунке 4.1

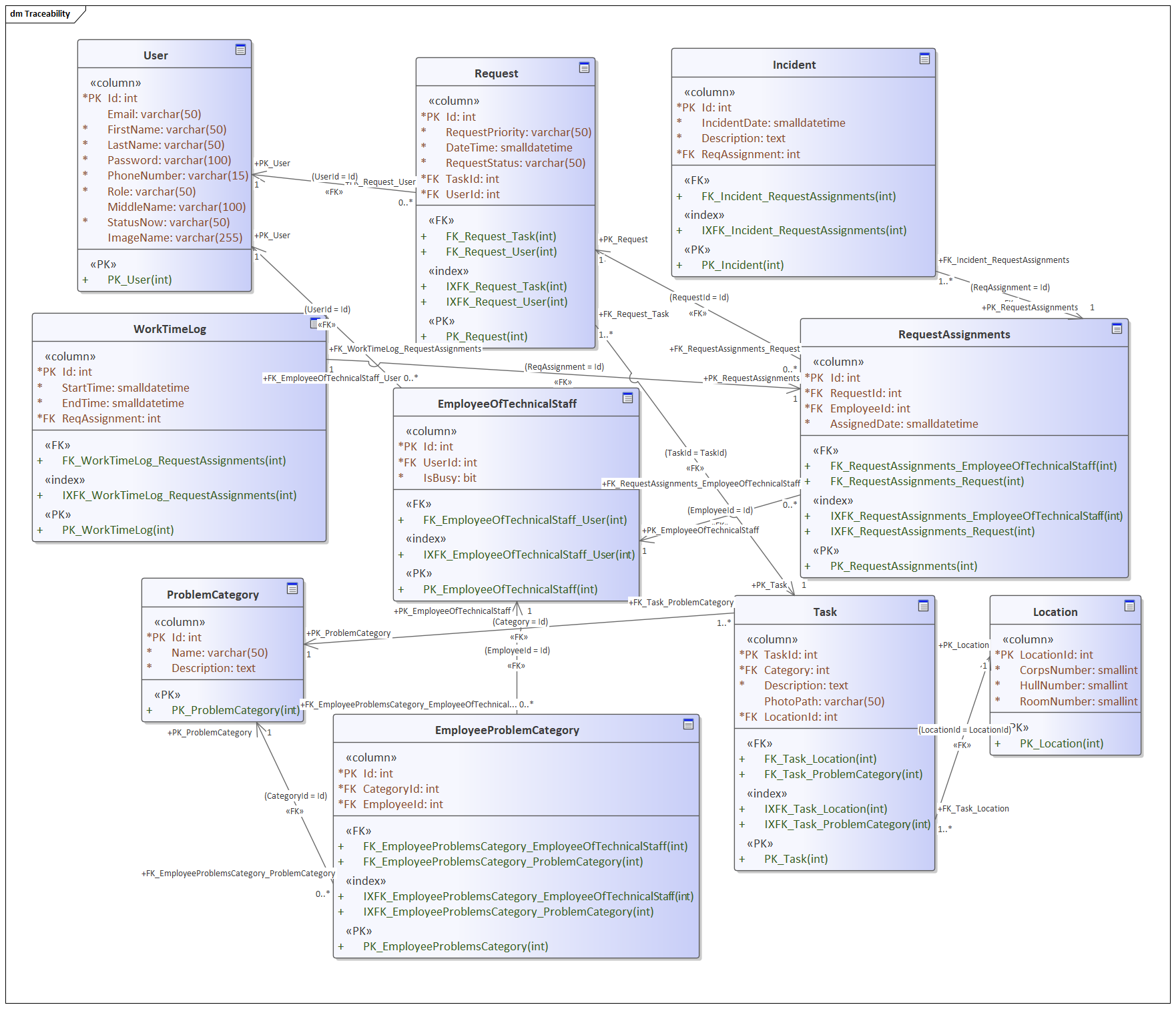


Рисунок 4.1 – Диаграмма структуры базы данных

Сущность User хранит информацию о пользователях системы. Она содержит следующие поля:

* + Id (int): уникальный идентификатор пользователя;
  + Email (varchar(50)): электронная почта пользователя;
  + FirstName(varchar(50)): имя пользователя;
  + SecondNmae(varchar(50)): фамилия пользователя;
  + Password(varchar(100)): пароль пользователя;
  + PhoneNumber(varchar(15)): номер телефона пользователя;
  + Role(varchar(50)): роль пользователя;
  + MiddleName(varchar(100)): отчество пользователя;
  + StatusNow(varchar(50)): текущий статус пользователя;
  + ImageName(varchar(255)): название фотографии пользователя.

Сущность Request содержит информацию о заявках пользователей. Она содержит следующие поля:

* + Id (int): уникальный идентификатор заявки;
  + RequestPriority (varchar(50)): приоритет заявки;
  + DateTime (datetime): дата и время создания заявки;
  + RequestStatus (varchar(50)): статус заявки;
  + TaskId (int): идентификатор задачи, связанной с заявкой;
  + UserId (int): идентификатор пользователя, создавшего заявку.

Сущность Incident хранит информацию о несчастных случаях, которые произошли во время выполнения заявки. Она содержит следующие поля:

* + Id (int): уникальный идентификатор инцидента;
  + IncidentDate (datetime): дата несчастного случая;
  + Description (text): описание случившегося несчастного случая;
  + ReqAssignment (int): идентификатор назначения, связанного с несчастным случаем.

Сущность WorkTimeLog хранит время начала и конца выполнения заявки сотрудником. Она содержит следующие поля:

* + Id (int): уникальный идентификатор лога;
  + StartTime (datetime): дата и время начала выполнения заявки;
  + EndTime (datetime): дата и время окончания выполнения заявки;
  + ReqAssignment (int): идентификатор назначения, связанного с логом.

Сущность ProblemCategory хранит информацию о категориях проблем, с которыми могут столкнуться пользователи. Она содержит следующие поля:

* + Id (int): уникальный идентификатор категории;
  + Name (varchar(50)): название категории проблемы;
  + Description (text): описание категории проблемы.

Сущность EmployeeOfTechnicalStaff хранит информацию о сотруднике технического персонала. Она содержит следующие поля:

* + Id (int): уникальный идентификатор сотрудника;
  + User (int): идентификатор пользователя в системе, который является сотрудником технического персонала;
  + IsBusy (Boolean): логическое значение, которое отвечает за занятость сотрудника.

Сущность RequestAssignments хранит информацию о назначении заявок сотрудникам. Она содержит следующие поля:

* + Id (int): уникальный идентификатор назначения;
  + RequestId (int): идентификатор заявки, назначаемой сотруднику;
  + EmployeeId (int): идентификатор работника, которому назначается заявка;
  + AssignedDate (smalldatetime): дата назначения заявки сотруднику.

Сущность EmployeeProblemCategory хранит информацию о категориях проблем, которые устраняет сотрудник. Она содержит следующие поля:

* + Id (int): уникальный идентификатор записи о категории проблемы, который решает сотрудник;
  + CategoryId (int): уникальный идентификатор категории проблемы;
  + EmployeeId (int): уникальный идентификатор сотрудника технического персонала.

Сущность Location хранит информацию о локациях. Она содержит следующие поля:

* + LocationId (int): уникальный идентификатор локации;
  + CorpsNumber (smallint): номер корпуса;
  + HullNumber (smallint): номер этажа;
  + RoomNumber (smallint): номер аудитории

Сущность Task хранит информацию о задаче, которую необходимо решить сотруднику технического персонала во время выполнения заявки. Она содержит следующие поля:

* + TaskId (int): уникальный идентификатор задачи;
  + Category (int): уникальный идентификатор категории проблемы, которая относится к данной задаче;
  + Description (text): описание задачи;
  + PhotoPath (varchar(50)): название фотографии, на которой изображена проблема;
  + LocationId (int): уникальный идентификатор локации, где случилась проблема.

# 5 Тестирование приложения

Объектом тестирования является серверная и клиентская компонента приложения для помощи в обслуживании имущества в учреждениях образования (для технического персонала). Целью тестирования является проверка работоспособности серверной и клиентской компоненты и обнаружение ошибок, при их наличии.

В ходе тестирования серверной компоненты будут проверены эндпоинты:

* Получение несчастных случаев с пагинацией;
* Получение заявок с пагинацией;
* Получение работников с пагинацией;
* Получение полной информации о сотруднике по Id;
* Получения пользователя по Id, которого не существует;
* Создание несчастного случая;
* Создание несчастного случая с неполными данными;
* Удаление заявки по ее Id;
* Обновление статуса заявки по Id;
* Получение выполненных заявок сотрудника с указанным Id;
* Получение всех категорий проблем сотрудника по его Id;
* Получение текущих заявок работника по его Id;
* Поиск заявок по ключевому слову;
* Поиск несчастного случая по Id назначения;
* Сортировка сотрудников по статусу занятости;

Тестирование этих эндпоинтов связано с проверкой основной функциональности сервера веб-приложения. Проверяется корректность работы CRUD-операций, работа с конкретными сущностями, поиск и обработка ошибок. Для тестирования серверной компоненты веб-приложения был использован инструмент Postman.

В ходе тестирования клиентской компоненты будет проверена корректность отображения данных на страницах и корректность работы основных функций. Клиентская часть тестировалась в браузере «Google Chrome» версии 136.0.7103.114 (Официальная сборка) (64 бит).

Тестирование получения несчастных случаев с пагинацией представлено на рисунке 5.1

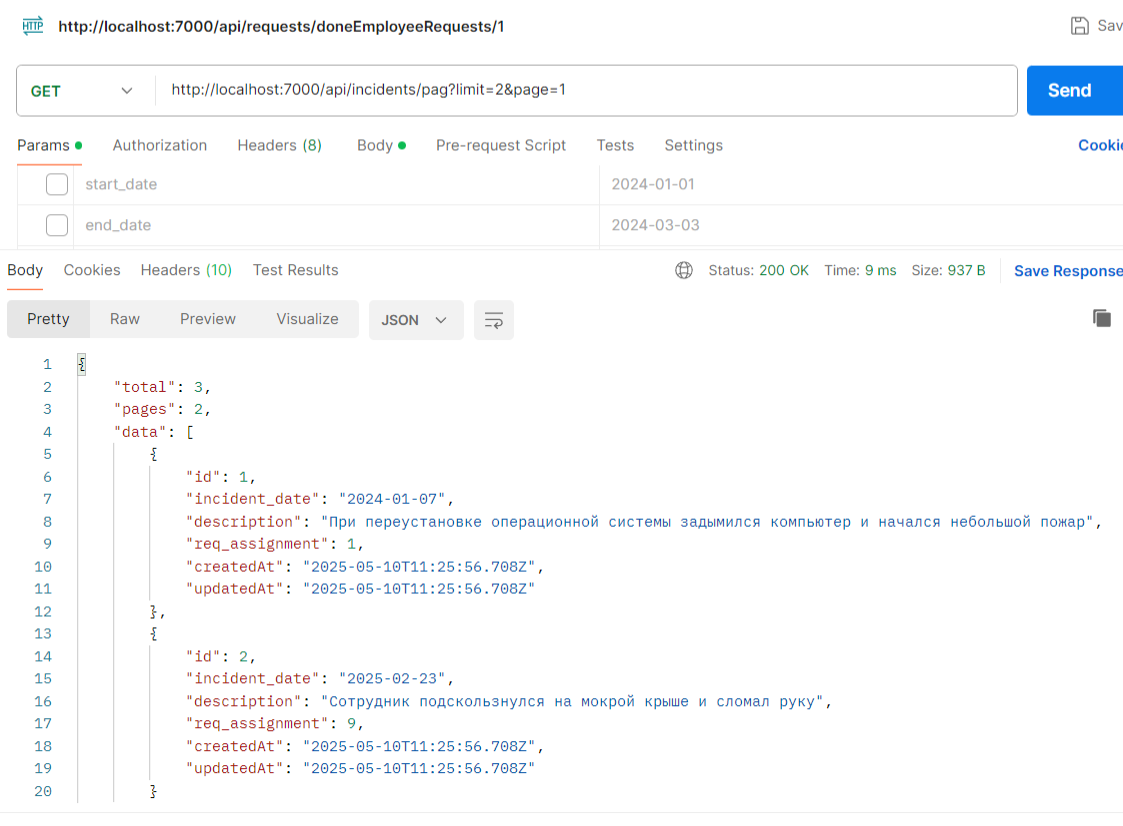


Рисунок 5.1 – Тестирование получения несчастных случаев с пагинацией

Тестирование получение заявок с пагинацией представлено на рисунке 5.2

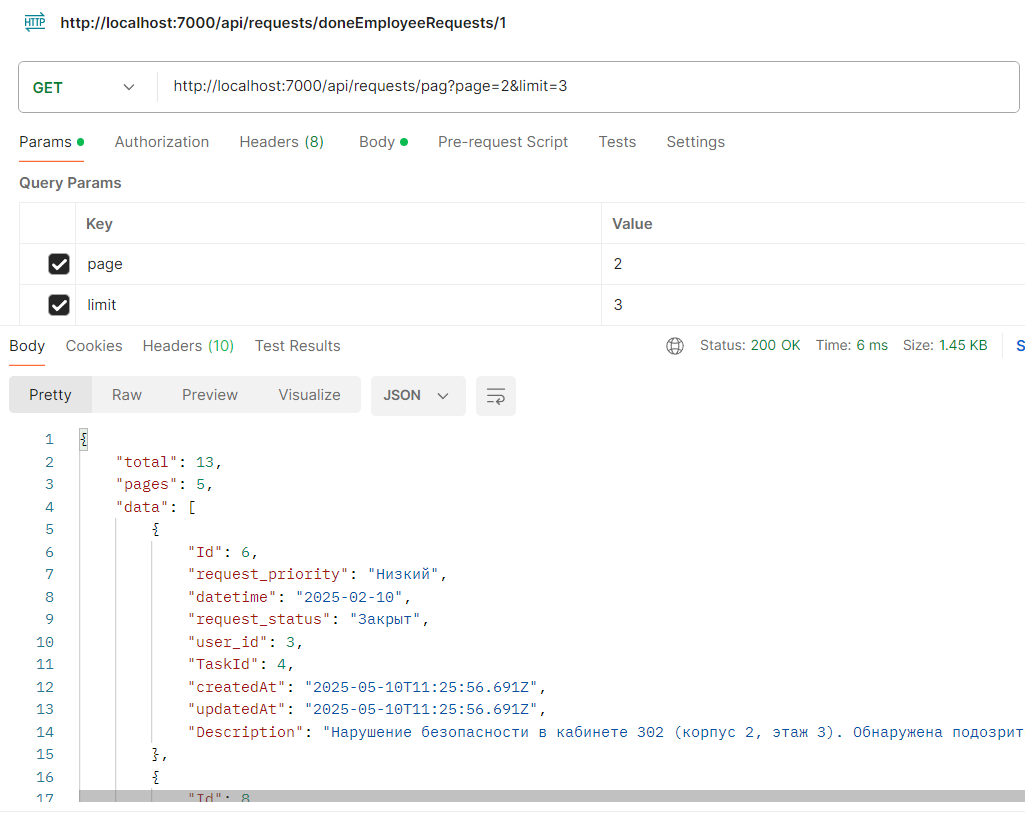


Рисунок 5.2 – Тестирование получения заявок с пагинацией

Тестирование получение работников с пагинацией представлено на рисунке 5.3

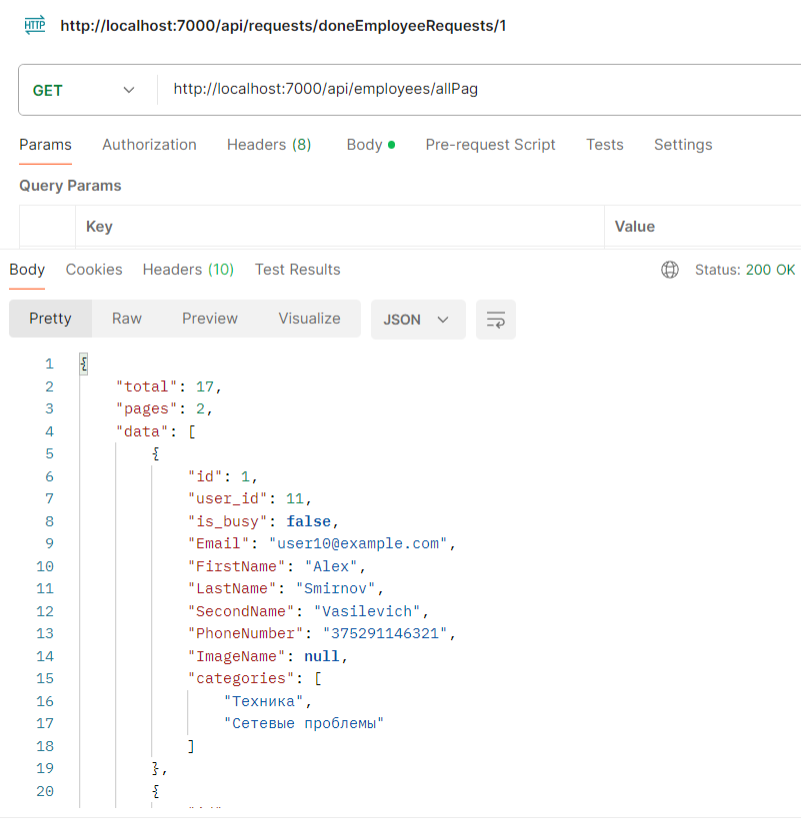


Рисунок 5.3 – Тестирование получения работников с пагинацией

Тестирование получение полной информации о сотруднике по Id представлено на рисунке 5.4

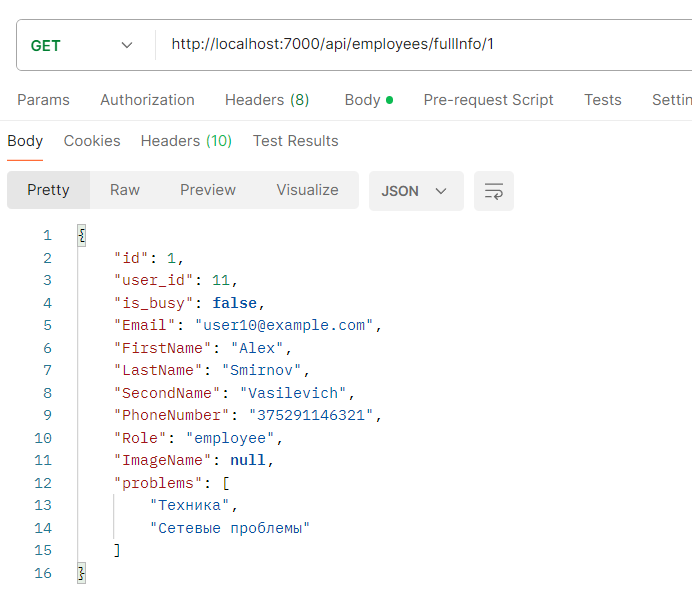


Рисунок 5.4 – Тестирование получения полной информации о сотруднике по Id

Тестирование получения пользователя по Id, когда пользователя с таким Id не существует, представлено на рисунке 5.5

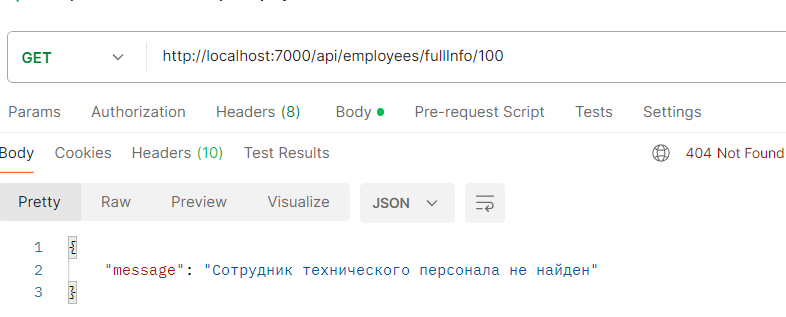


Рисунок 5.5 – Тестирования получения пользователя по Id, когда пользователя с таким Id не существует

Тестирование создания несчастного случая представлено на рисунке 5.6

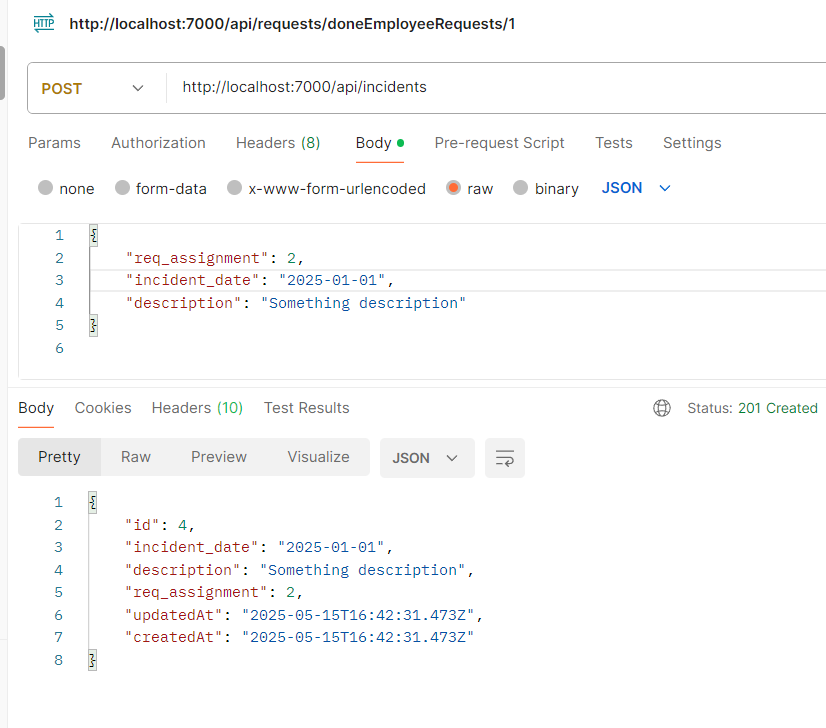


Рисунок 5.6 – Тестирование создания несчастного случая

Тестирование создания несчастного случая с неполными данными представлено на рисунке 5.7

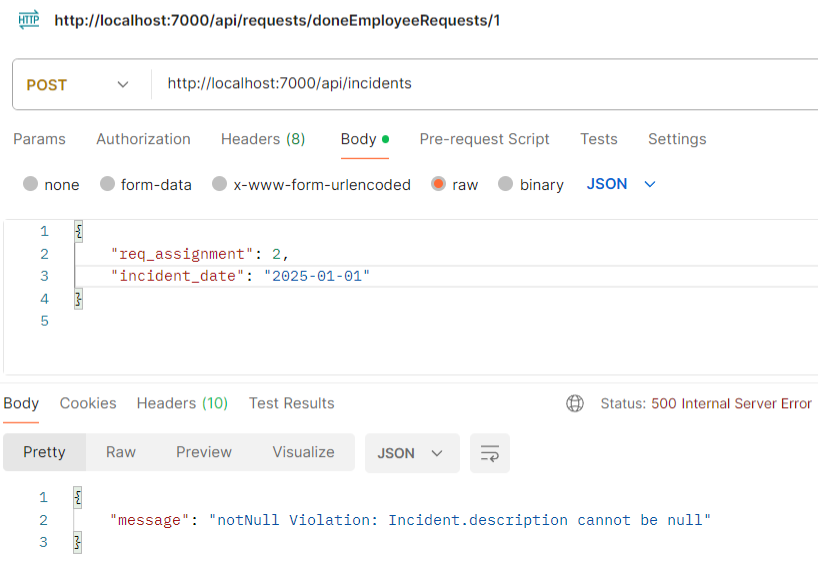


Рисунок 5.7 – Тестирование создания несчастного случая с неполными данными

Тестирование удаление заявки по ее Id представлено на рисунке 5.8

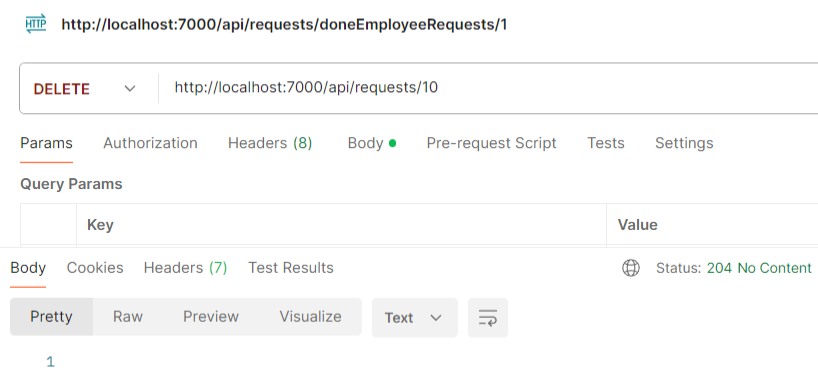


Рисунок 5.8 – Тестирование удаление заявки по ее Id

Тестирование обновление статуса заявки по ее Id представлено на рисунке 5.9

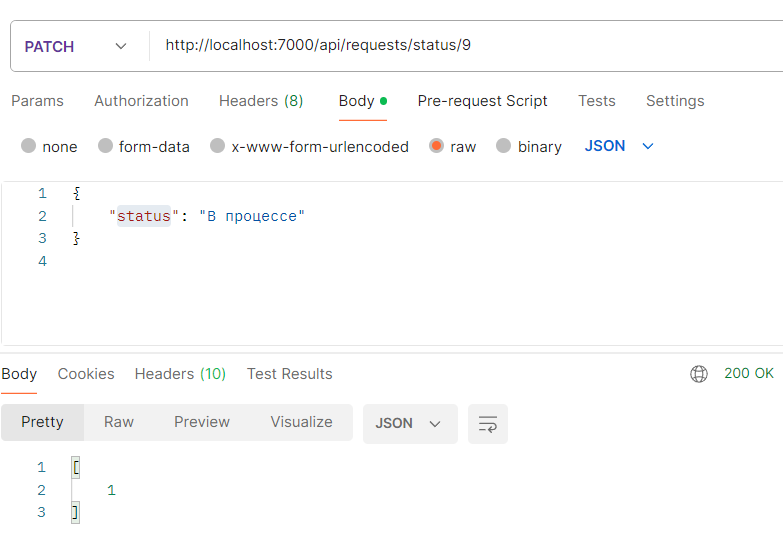


Рисунок 5.9 – Обновление статуса заявки по ее Id

Тестирование получения выполненных заявок сотрудником с указанным Id представлено на рисунке 5.10

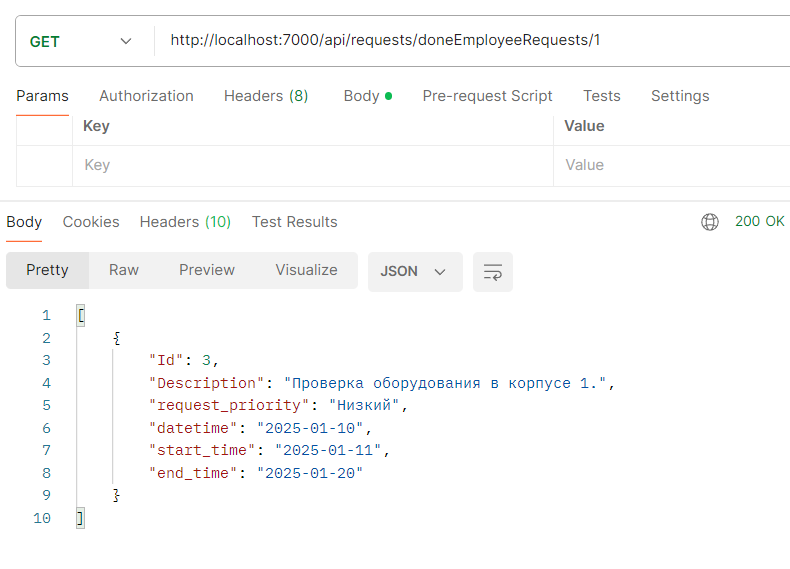


Рисунок 5.10 – Тестирование получения выполненных заявок сотрудником с указанными Id

Тестирование получения всех категорий проблем, которые решает сотрудник с указанным Id представлено на рисунке 5.11

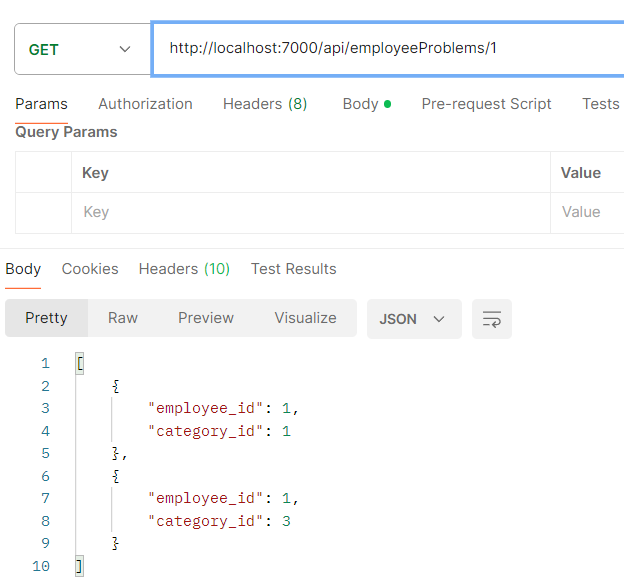


Рисунок 5.11 – Тестирование получения всех категорий проблем, которые решает сотрудник с указанным Id

Тестирование получения текущих заявок работника по его Id представлено на рисунке 5.12

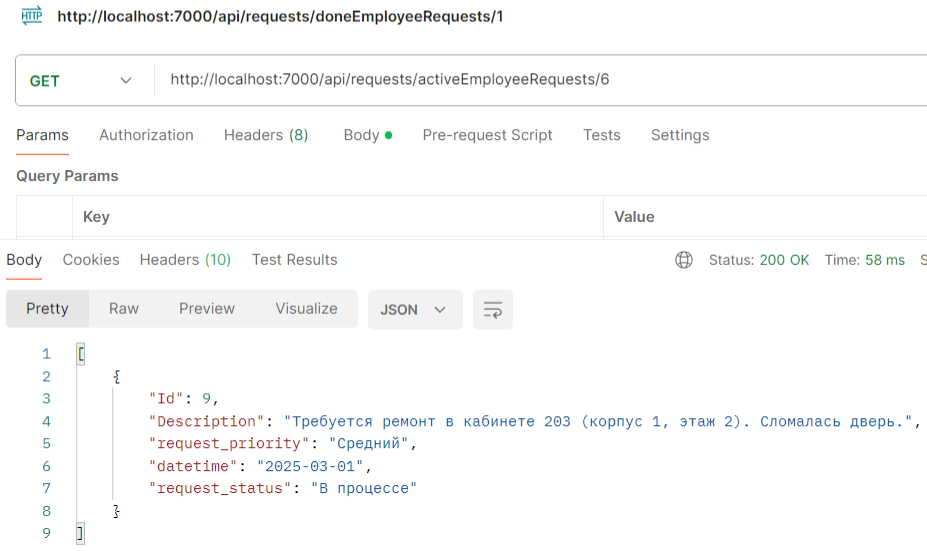


Рисунок 5.12 – Тестирование получения текущих заявок работника по его Id

Тестирование поиска заявок по ключевому слову представлено на рисунке 5.13

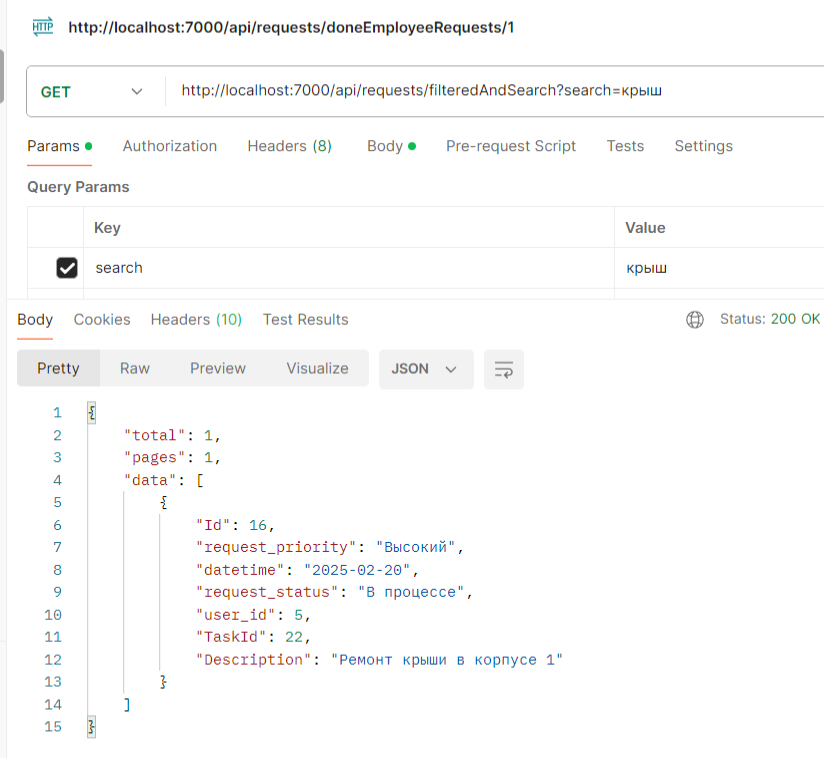


Рисунок 5.13 – Тестирование поиска заявок по ключевому слову

Тестирование поиска несчастного случая по Id назначения представлено на рисунке 5.14

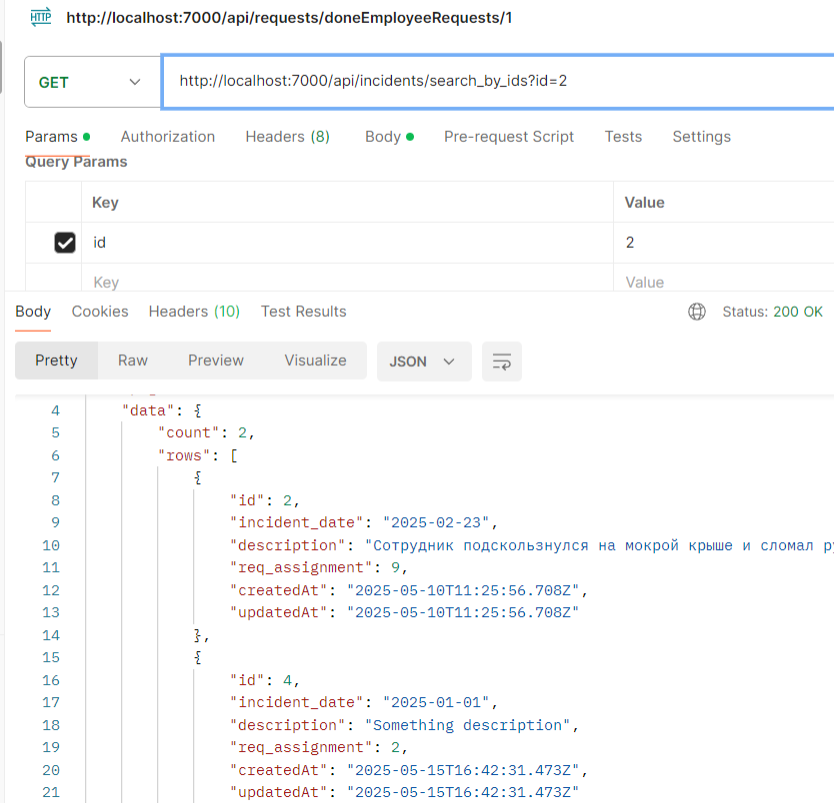


Рисунок 5.14 – Тестирование поиска несчастного случая по Id назначения

Тестирование сортировки сотрудников по статусу занятости представлено на рисунке 5.15

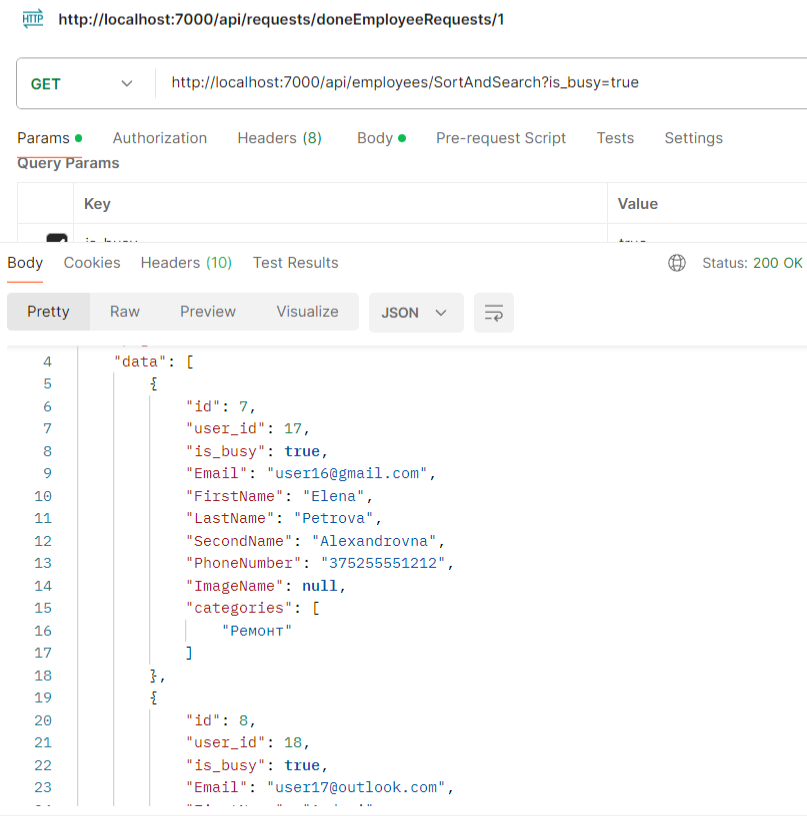


Рисунок 5.15 – Тестирование сортировки сотрудников по статусу занятости

После успешной авторизации как руководитель технического персонала, пользователь видит на странице таблицу с данными сотрудников. Результат представлен на рисунке 5.16

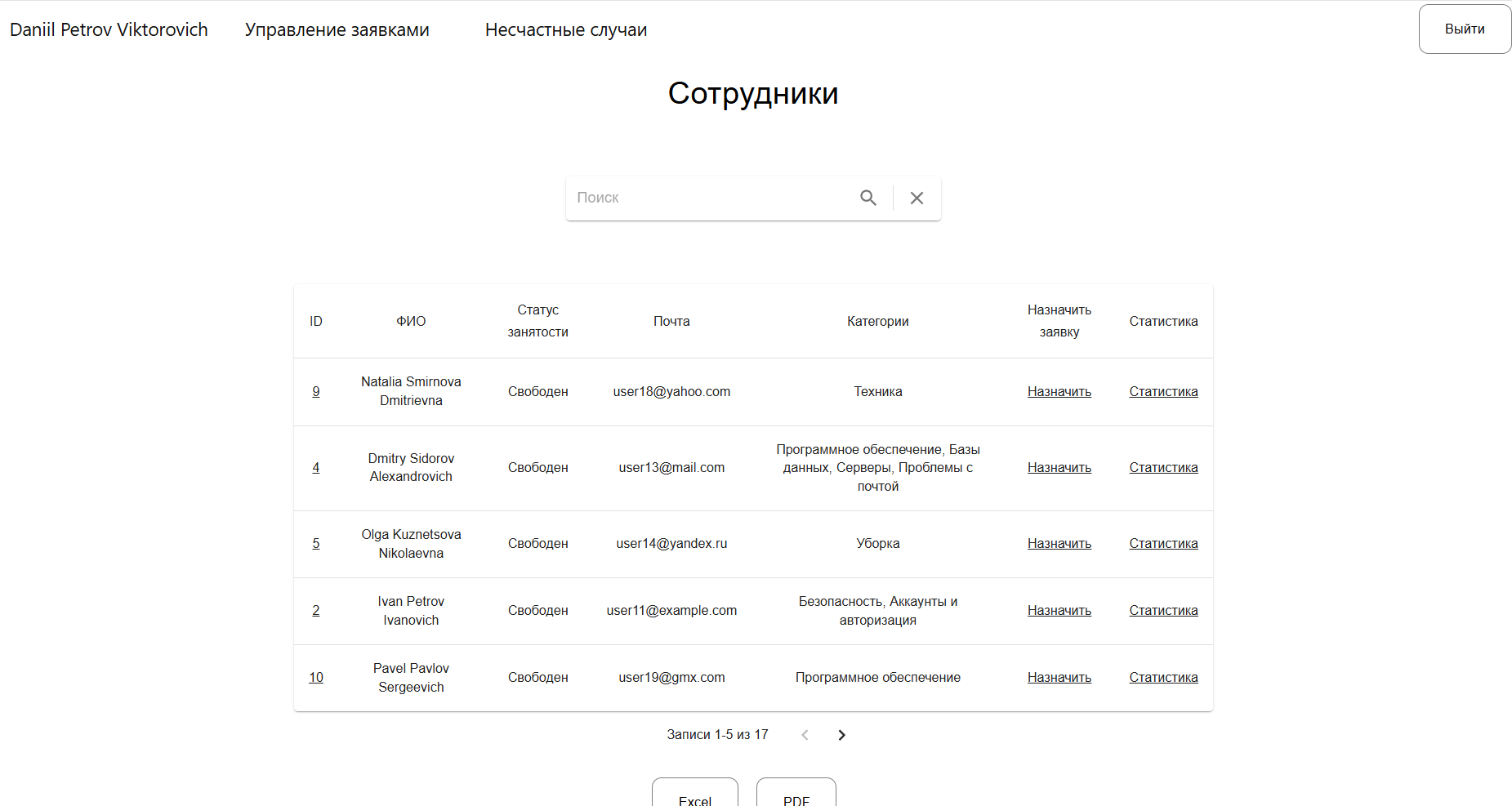


Рисунок 5.16 – Отображение сотрудников на странице руководителя

При нажатии на ссылку «Управление заявками» руководителя перенаправляет на страницу «Request management». Результат представлен на рисунке 5.17

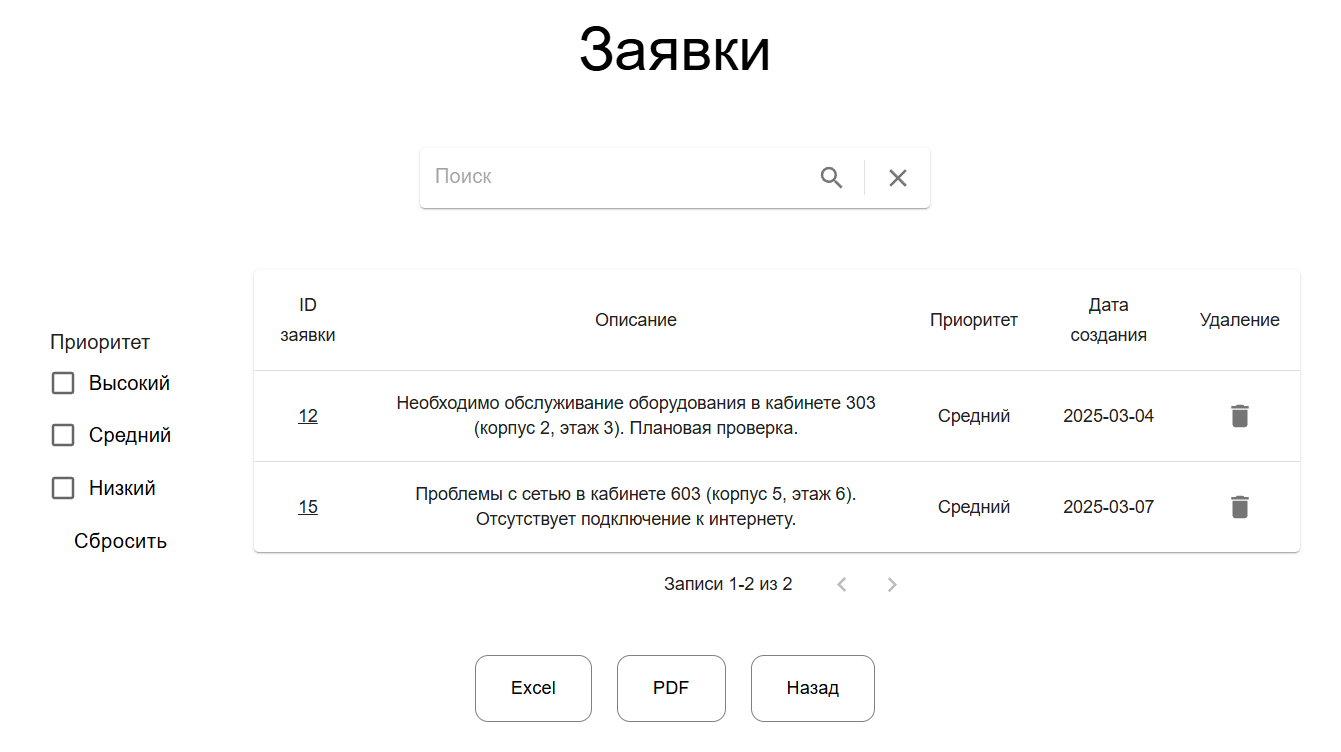


Рисунок 5.17 – Страница «Request management»

При нажатии на кнопку «PDF» происходит экспорт списка заявок, которые отображаются на экране, в файл. Результат представлен на рисунке 5.18

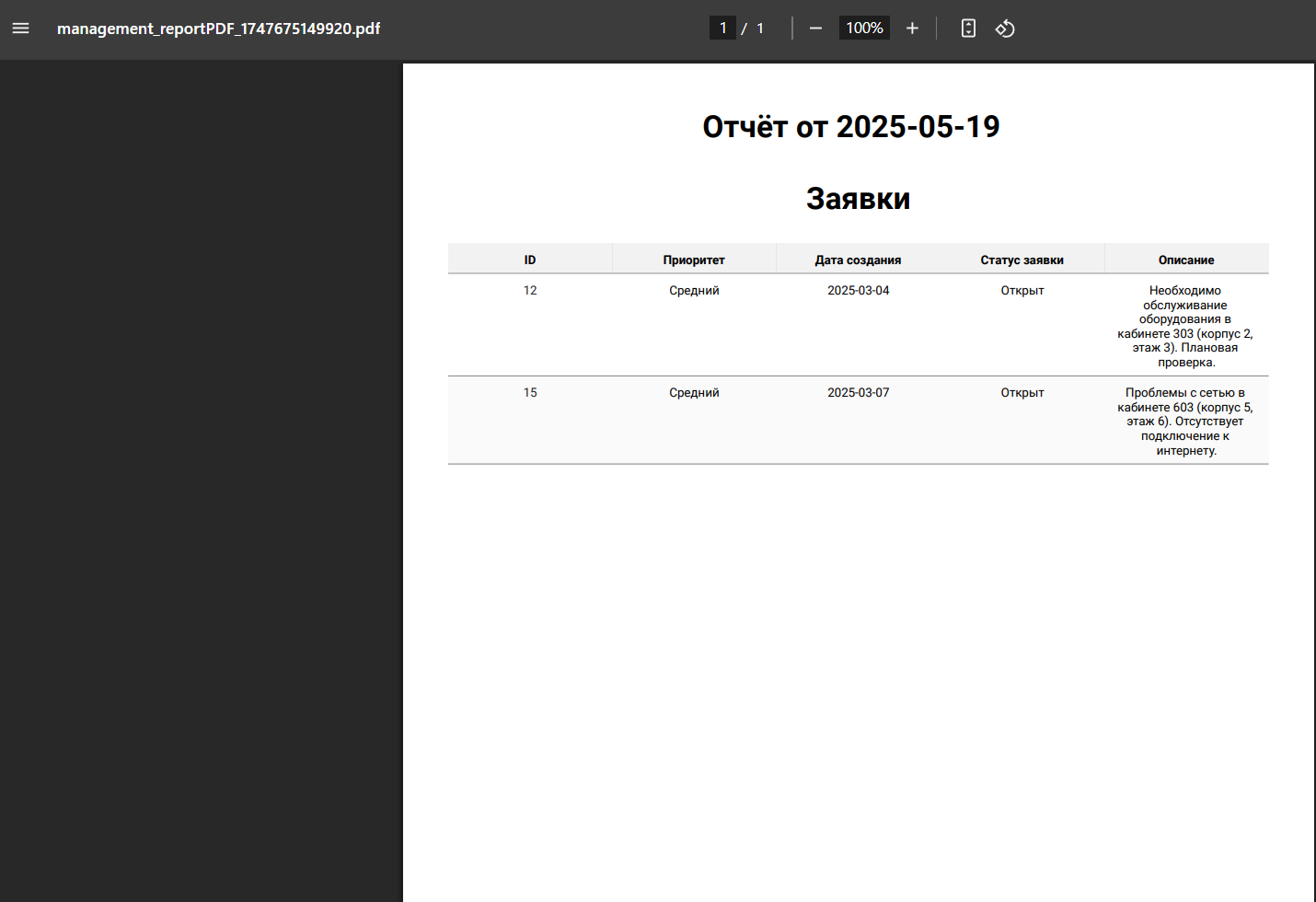


Рисунок 5.18 – PDF документ экспортированных заявок

При нажатии на иконку мусорного ведра на странице появляется модальное окно с подтверждением удаления заявки. Результат представлен на рисунке 5.19

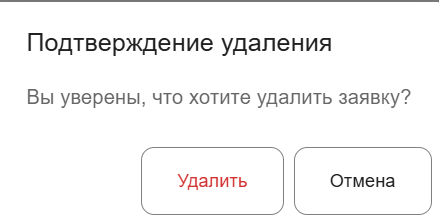


Рисунок 5.19 – Запрос на подтверждение удаления заявки

При нажатии на кнопку «Удалить» происходит удаление заявки. Результат представлен на рисунке 5.20

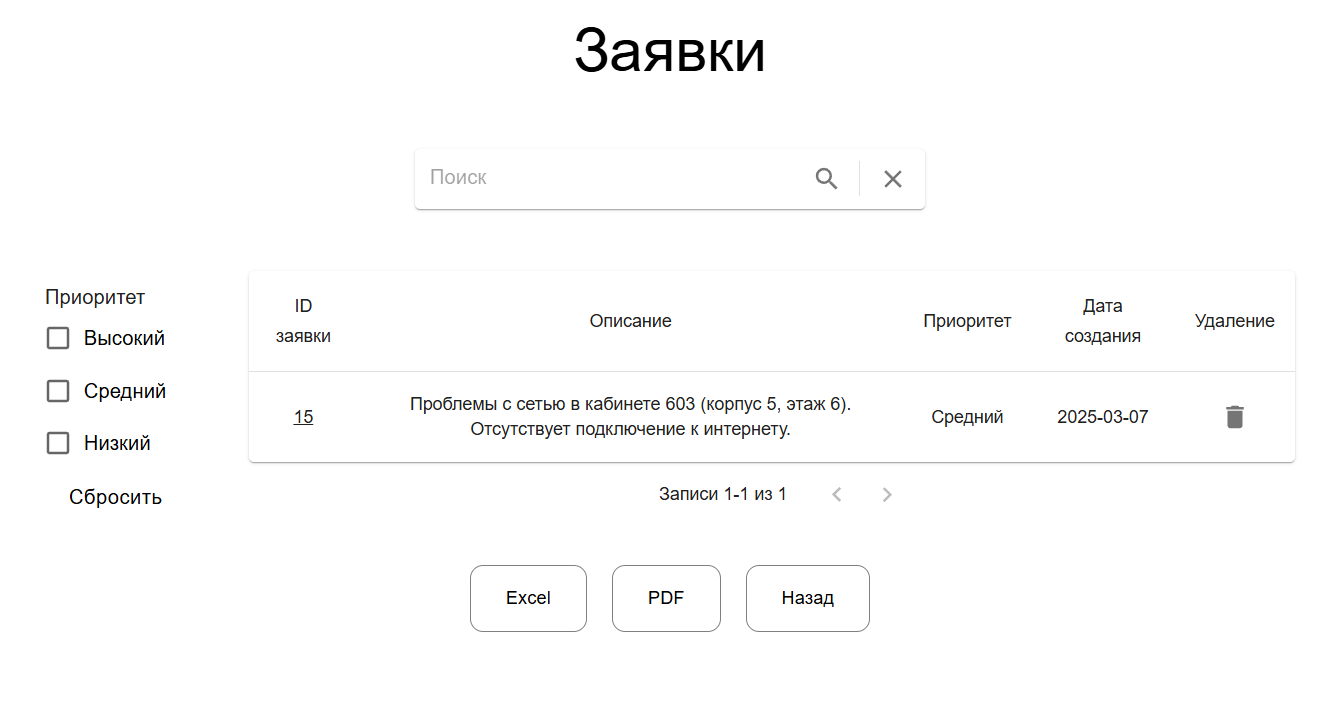


Рисунок 5.20 – Список заявок после удаления

При нажатии на кнопку «Назад» происходит перенаправление на страницу Technical staff manager. При нажатии на ссылку «Статистика» руководитель технического персонала перенаправляется на страницу «Statistic». Результат представлен на рисунке 5.21

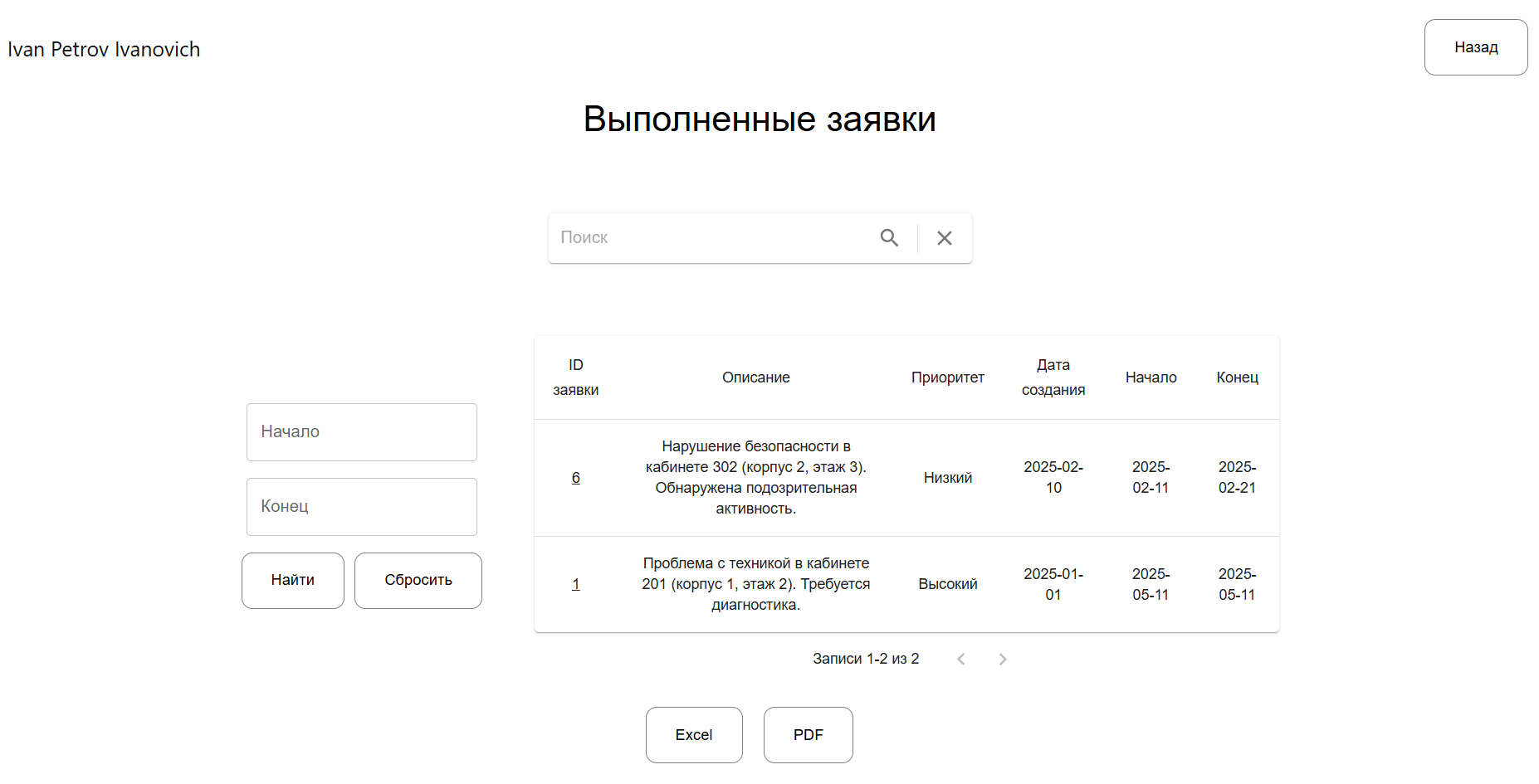


Рисунок 5.21 – Страница «Statistic»

На странице отображаются выполненные заявки сотрудника и календарь. В календаре можно посмотреть, когда заявка была создана, принята в исполнение и сколько она выполнялась по времени. Календарь для февраля выбранного сотрудника представлен на рисунке 5.22

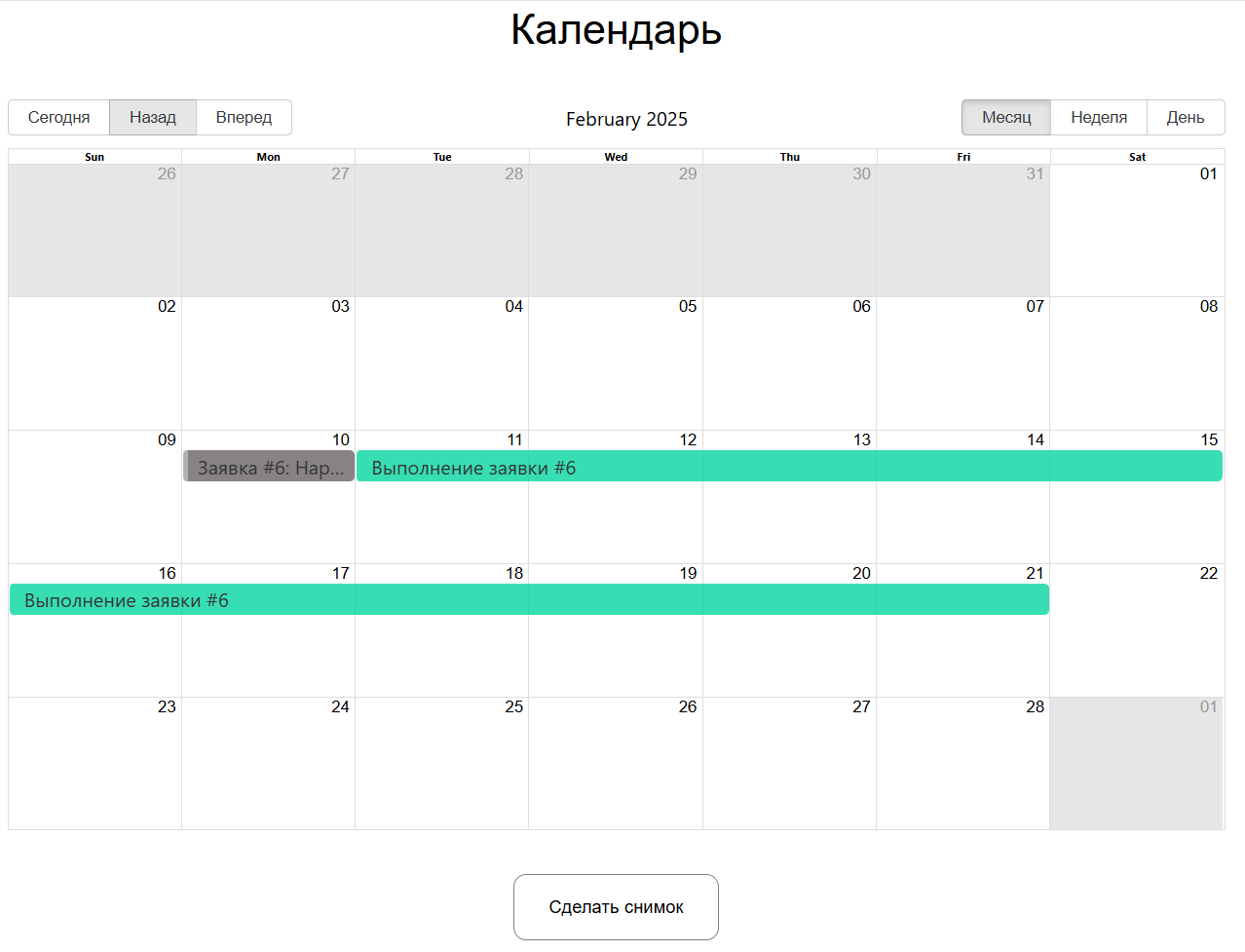


Рисунок 5.22 – Календарь февраля для выбранного сотрудника

При нажатии на кнопку «Сделать снимок» делается снимок календаря. Снимок календаря представлен на рисунке 5.23

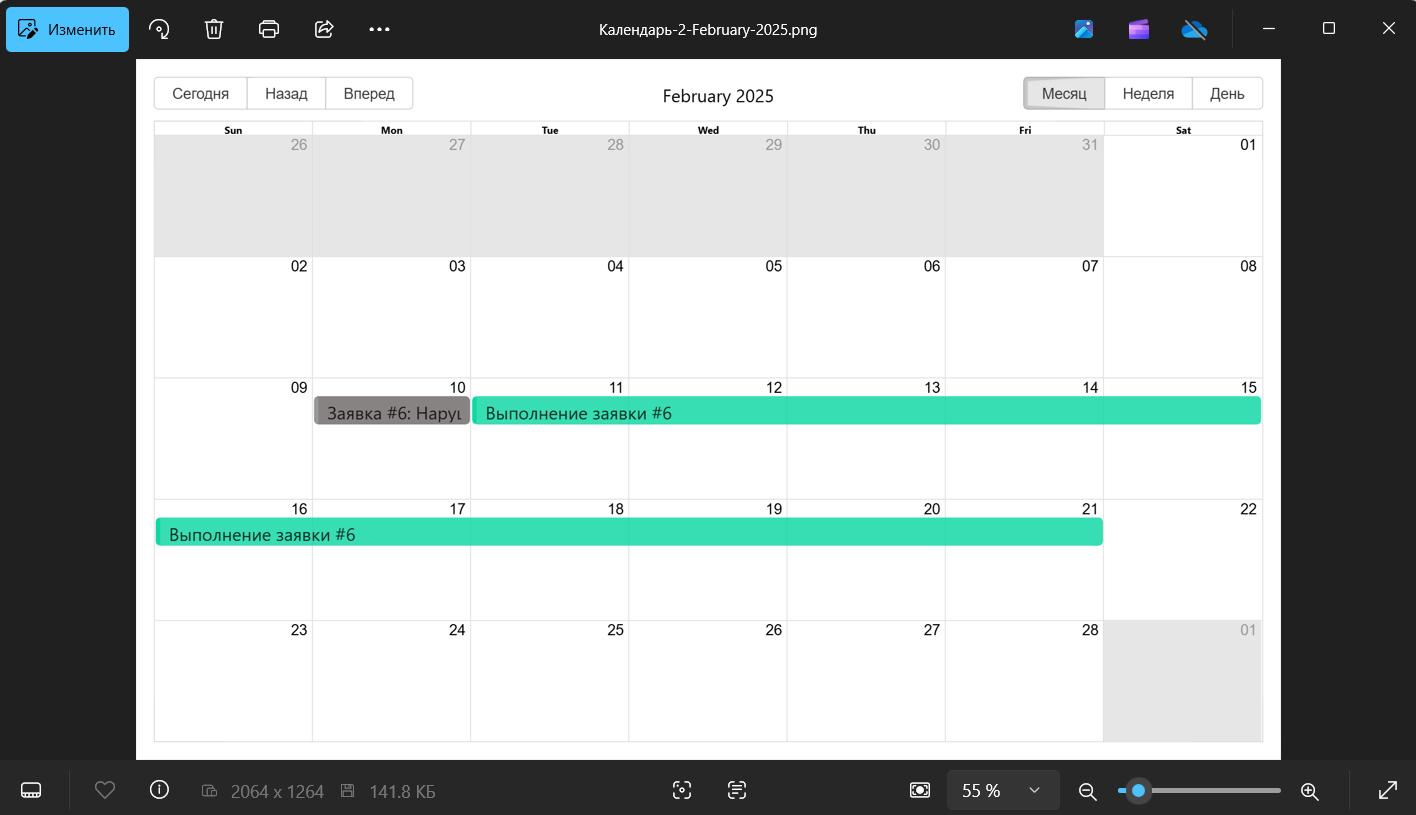


Рисунок 5.23 – Снимок календаря

При нажатии на ссылку «Назначить» на странице «Technical staff manager» руководитель перенаправляется на страницу «Requests». Результат представлен на рисунке 5.24

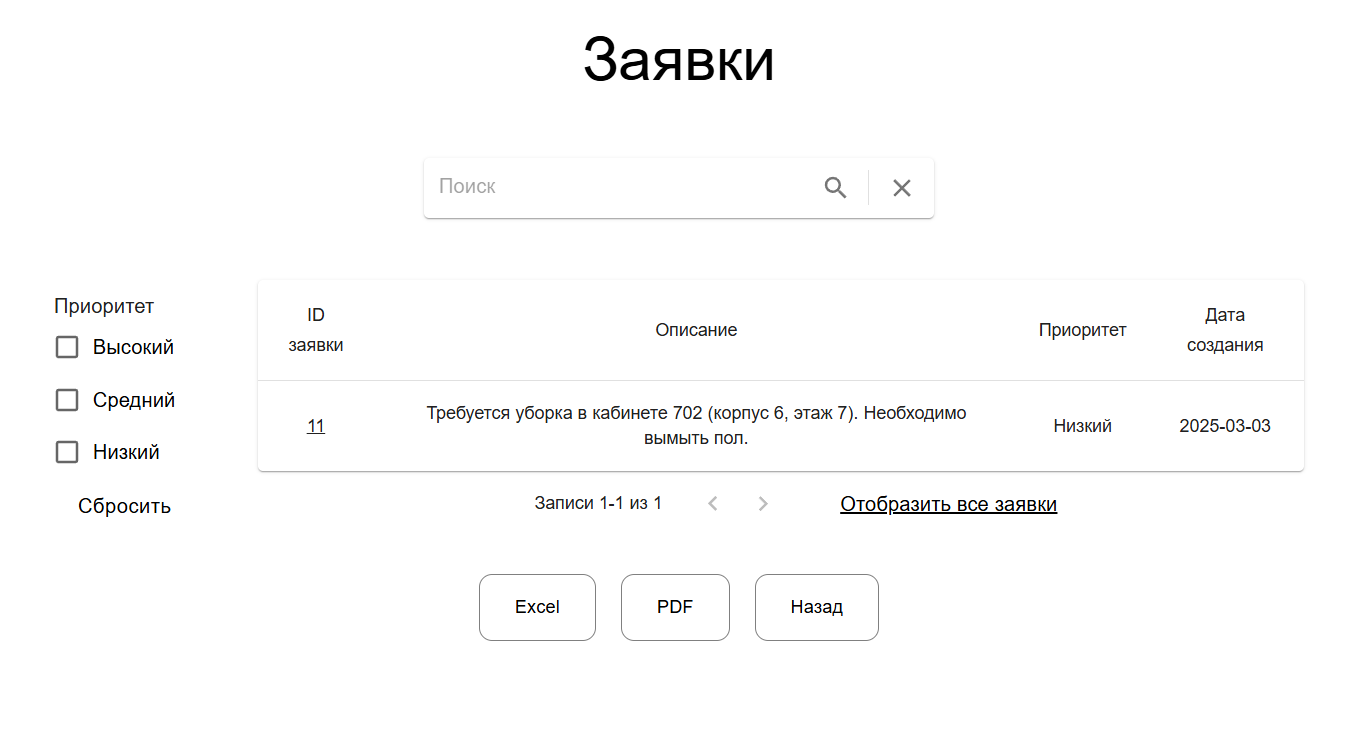


Рисунок 5.24 – Страница «Requests» для руководителя

При нажатии на ссылку «11» в столбце «ID заявки» таблицы, откроется модальное окно с подробным описанием заявки. Результат представлен на рисунке 5.25

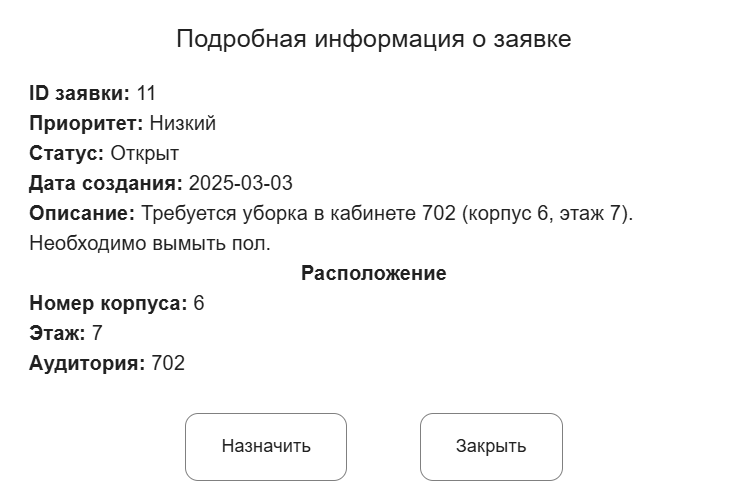


Рисунок 5.25 – Модальное окно с подробным описанием заявки

При нажатии на кнопку «Назначить» выбранному сотруднику назначается заявка для выполнения и модальное окно закрывается. Результат назначения заявки представлен на рисунке 5.26.

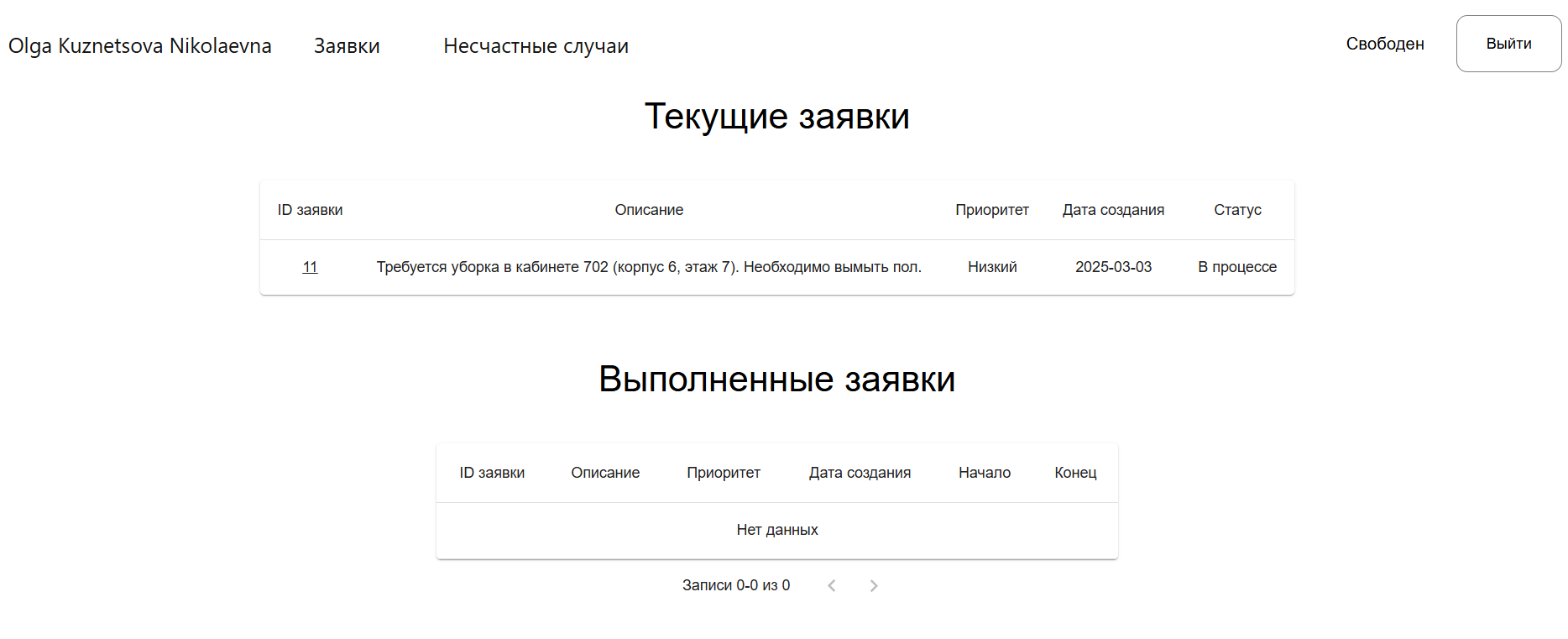


Рисунок 5.26 – Отображение назначенной заявки в текущих заявках

Работник технического персонала может изменить статус заявки на «Открыта», «В процессе» и «Закрыта». Результат смены статуса заявки на «Закрыта» представлен на рисунке 5.27

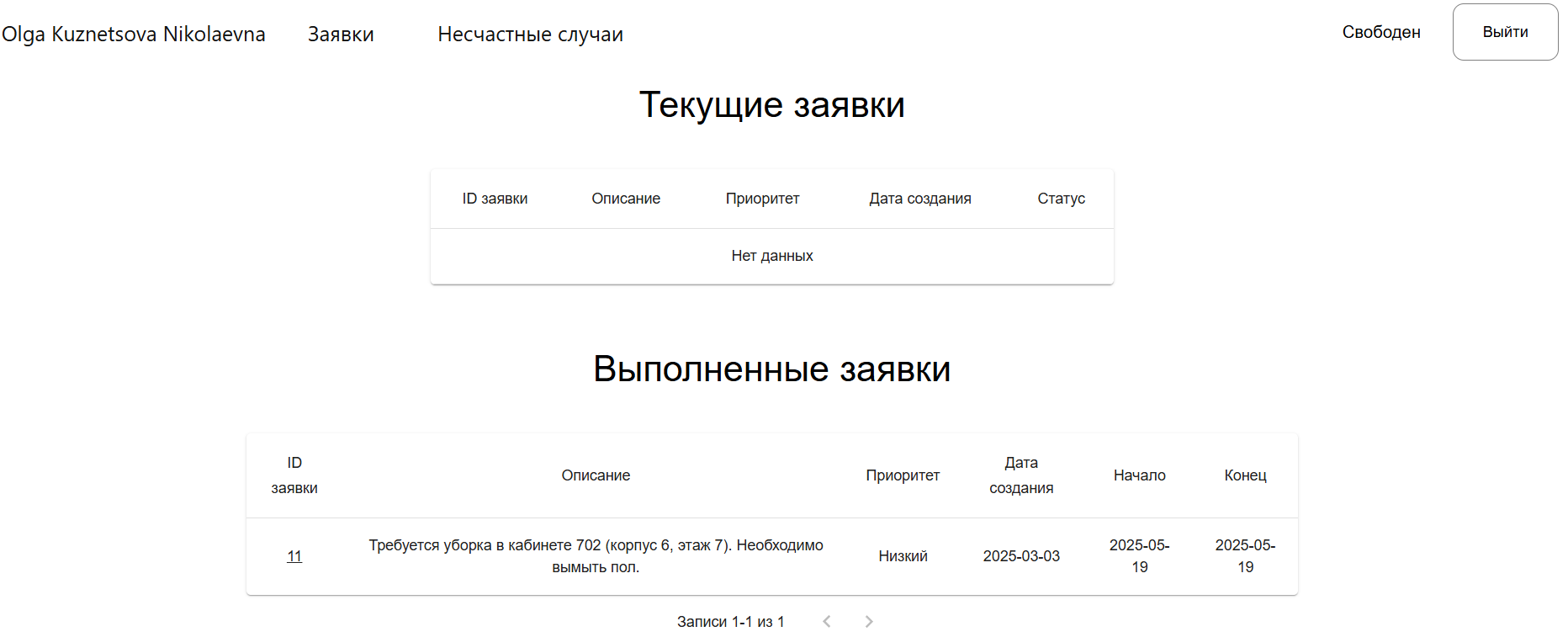


Рисунок 5.27 – Отображение заявки в выполненных заявках

При нажатии на ссылку «Несчастные случаи» работник перенаправляется на страницу «Incidents». Результат представлен на рисунке 5.28

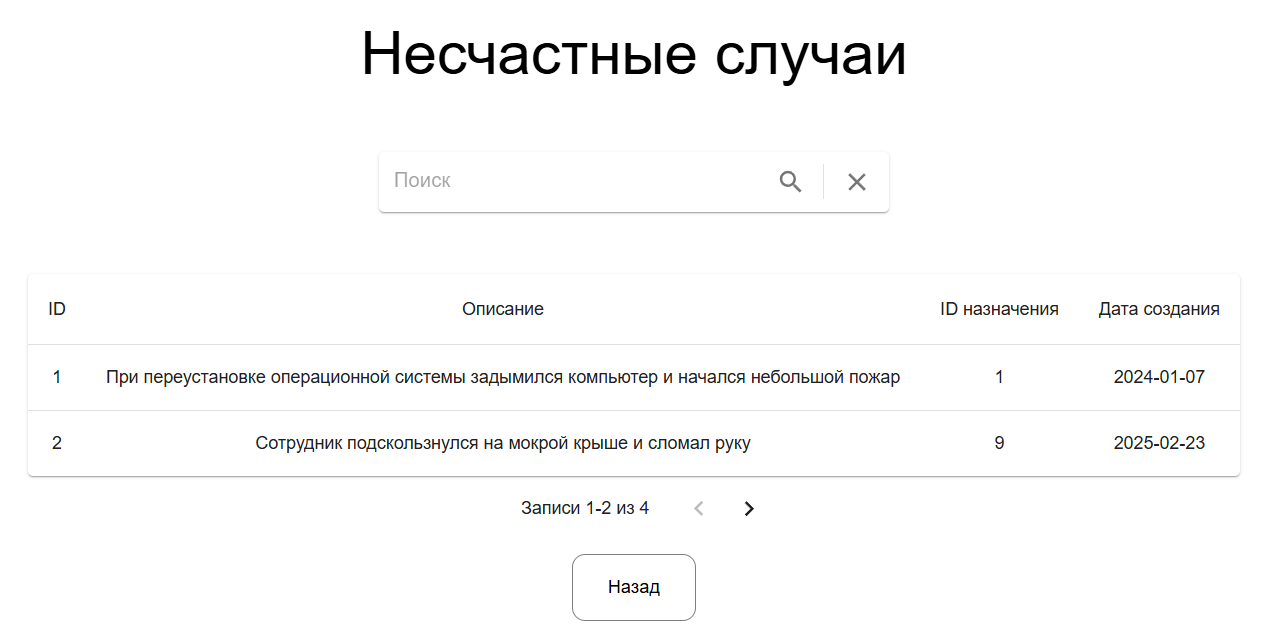


Рисунок 5.28 – Страница «Incidents» для сотрудника

При нажатии на ФИО на странице Employee of technical staff сотрудник попадает на страницу «Full info». Результат представлен на рисунке 5.29

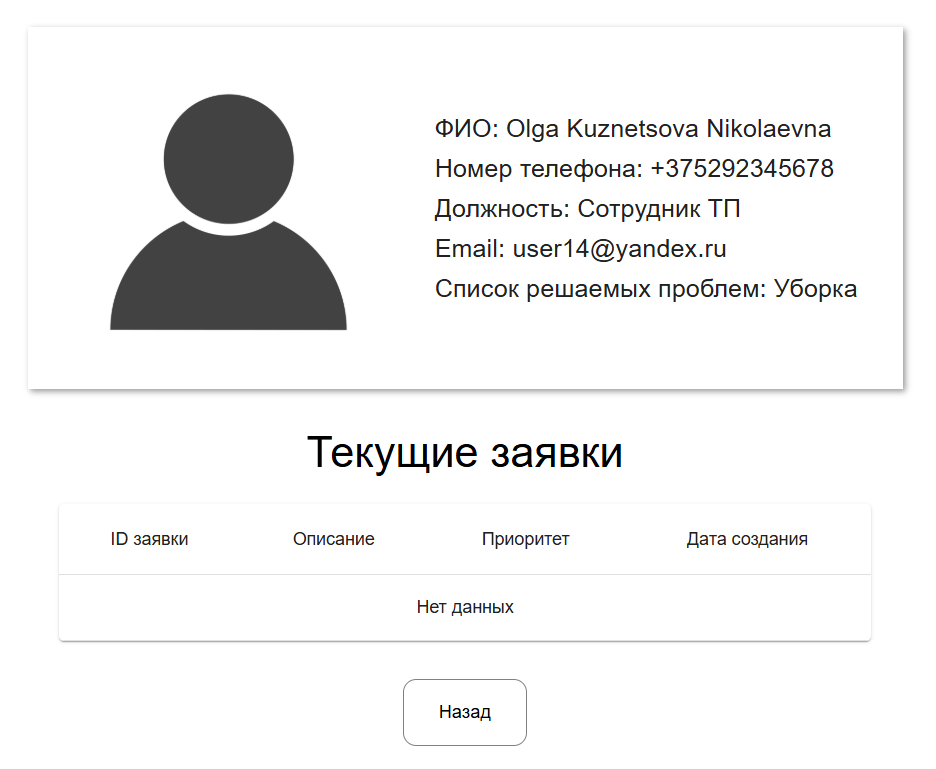


Рисунок 5.29 – Страница «Full info» для работника

# Заключение

В рамках выполнения курсовой работы были достигнуты все поставленные цели, а именно: разработано WEB-приложение для помощи в обслуживании имущества в учреждениях образования, предназначенное для автоматизация работы технического персонала.

Приложение предоставляет сотруднику технического персонала удобный интерфейс для просмотра информации о заявках, что позволяет легко отслеживать статус каждой заявки, включая детали о времени создания, приоритете и текущем состоянии выполнения. Так же работник может просматривать информацию о несчастных случаях, о своих данных и о выполненных заявках.

Приложение также предоставляет функционал для генерации документов в форматах Excel и PDF, что позволяет руководителю легко экспортировать данные о заявках, работниках и календаре выполнения заявок для дальнейшего анализа и отчетности. Эта возможность значительно упрощает процесс подготовки отчетов и делает его более эффективным.

Были выбраны современные и подходящие технологии для реализации проекта, спроектированы и реализованы клиентская и серверная части приложения, а также создана система взаимодействия с базой данных.

Среди основных преимуществ разработанного приложения можно выделить удобный и интуитивно понятный интерфейс и современный внешний вид. Применение таких технологий, как JavaScript, React для клиентской части и Express, Sequelize, PostgreSQL на серверной стороне, обеспечило гибкость архитектуры и возможность масштабирования проекта.

Разработанное решение значительно оптимизирует процессы технического обслуживания, сокращает время реагирования на неисправности и способствует поддержанию комфортных условий для учебного процесса. Автоматизация рутинных операций позволяет техническому персоналу сосредоточиться на решении важных задач, а преподавателям – уделять больше времени образовательной деятельности.

# Список литературы

1. Express.js: Руководство для начинающих [Электронный ресурс] — Режим доступа: https://expressjs.com/ – Дата доступа: 15.05.2025;
2. Основы React [Электронный ресурс] — Режим доступа: https://react.dev/learn – Дата доступа: 15.05.2025;
3. React Router: Docs [Электронный ресурс] – Режим доступа: https://reactrouter.com/en/main – Дата доступа: 15.05.2025;
4. PostgreSQL: Руководство [Электронный ресурс] — Режим доступа: https://www.postgresql.org/docs/ – Дата доступа: 15.05.2025;
5. Немцова, Т. И. Компьютерная графика и web-дизайн: учебное пособие / Т. И. Немцова, Т. В. Казанкова, А. В. Шнякин; под ред. Л. Г. Гагариной. – Москва: ИМ «ФОРУМ»: ИНФРА-М, 2020. – 400 с;
6. React и JavaScript: Совместное использование [Электронный ресурс] — Режим доступа: https://react.dev/learn/thinking-in-react – Дата доступа: 15.05.2025;
7. Взаимодействие PostgreSQL и Node.js [Электронный ресурс] — Режим доступа: https://node-postgres.com/ – Дата доступа: 15.05.2025;
8. REST API и Express.js [Электронный ресурс] — Режим доступа: https://developer.mozilla.org/ru/docs/Learn/Server-side/Express\_Nodejs – Дата доступа: 15.05.2025;
9. Обработка исключений [Электронный ресурс] — Режим доступа: https://doka.guide/js/try-catch/ – Дата доступа: 15.05.2025.
10. Работа с Axios [Электронный ресурс] — Режим доступа: https://axios-http.com/docs/intro – Дата доступа: 15.05.2025.

# Приложение 1 – Программный код приложения

class authController {

async login(req, res) {

try {

const { login, password } = req.body;

const user = await User.findOne({

where: {

[Op.or]: [

{ Email: login },

{ PhoneNumber: login }

],

}

})

if (!user) {

return res.status(404).json({ message: `Пользователь ${username} не найден` });

}

const validPassword = bcrypt.compareSync(password, user.Password);

if (!validPassword) {

return res.status(401).json({ message: `Введен неверный пароль` });

}

const roleName = user.Role;

const userId = user.id

//const token = generateAccessToken(user.id, roleNames)

return res.status(200).json({ userId, roleName });

} catch (e) {

console.log(e);

res.status(500).json({ message: "Ошибка авторизации" })

}

}

}

class EmployeeOfTechnicalStaffController {

async getAllEmployees(req, res) {

try {

const employees = await EmployeeOfTechnicalStaffService.gettAllEmployees();

res.status(200).json(employees);

} catch (error) {

res.status(500).json({ message: error.message });

}

}

async getEmployeeById(req, res) {

const { employee\_id } = req.params;

try {

const employee = await EmployeeOfTechnicalStaffService.getEmployeeById(employee\_id);

if (!employee) {

return res.status(404).json({ message: "Сотрудник технического персонала не найден" });

}

res.status(200).json(employee);

} catch (error) {

res.status(500).json({ message: error.message });

}

}

async getEmployeesFullInfoWithPagination(req, res) {

const { page = 1, limit = 10 } = req.query;

const offset = (page - 1) \* limit;

try {

const employees = await EmployeeOfTechnicalStaffService.getAllEmployeesWithPagination(limit, offset);

res.status(200).json({

total: employees.count,

pages: Math.ceil(employees.count / limit),

data: employees.rows

});

}

catch (error) {

res.status(500).json({ message: error.message })

}

}

async getEmployeesFullInfoSorteredByBusyStatusAndSearch(req, res) {

const { page = 1, limit = 10, is\_busy, search } = req.query;

const offset = (page - 1) \* limit;

try {

let boolean\_busy;

if(Number(is\_busy) === 0){

boolean\_busy = false

}

else{

boolean\_busy = true

}

const employees = await EmployeeOfTechnicalStaffService.getAllEmployeesSortedByBusyStatusAndSearch(limit, offset, boolean\_busy, search);

res.status(200).json({

total: employees.count,

pages: Math.ceil(employees.count / limit),

data: employees.rows

});

}

catch (error) {

res.status(500).json({ message: error.message })

}

}

async getFullInfoByEmployee(req, res) {

const { employee\_id } = req.params;

try {

const employee = await EmployeeOfTechnicalStaffService.getFullEmployeeInfoById(employee\_id);

if (!employee) {

return res.status(404).json({ message: "Сотрудник технического персонала не найден" });

}

res.status(200).json(employee);

}

catch (error) {

res.status(500).json({ message: error.message })

}

}

async getEmployeeByUserId(req, res) {

const { user\_id } = req.params;

try {

const employee = await EmployeeOfTechnicalStaffService.getEmployeeByUserId(user\_id);

if (!employee) {

return res.status(404).json({ message: "Сотрудник технического персонала не найден" });

}

res.status(200).json(employee);

}

catch (error) {

res.status(500).json({ message: error.message })

}

}

async setBusy(req, res) {

const { employee\_id } = req.params;

const { busy } = req.body;

try {

const employee = await EmployeeOfTechnicalStaffService.setBusy(employee\_id, busy);

if (!employee) {

return res.status(404).json({ message: "Сотрудник технического персонала не найден" });

}

res.status(200).json(employee);

} catch (error) {

res.status(500).json({ message: error.message });

}

}

}

class EmployeeProblemsCategoryController {

async getAllEmployeeProblemsCategories(req, res) {

try {

const employeeProblemCategories = await EmployeeProblemsCategoryService.getAllEmployeeProblemCategories();

res.status(200).json(employeeProblemCategories);

} catch (error) {

res.status(500).json({ message: error.message });

}

}

async getEmployeeProblemsCategoryByIds(req, res) {

const { employee\_id, category\_id } = req.params;

try {

const employeeProblemCategory = await EmployeeProblemsCategoryService.getEmployeeProblemCategoryByIds(employee\_id, category\_id);

if (!employeeProblemCategory) {

return res.status(404).json({ message: "Категория у данного сотрудника не найдена" });

}

res.status(200).json(employee);

} catch (error) {

res.status(500).json({ message: error.message });

}

}

async getAllProblemsCategoriesOfEmployee(req, res) {

const { employee\_id } = req.params;

try {

const problemsCategoriesOfEmployee = await EmployeeProblemsCategoryService.getallProblemsCategoriesOfEmployee(employee\_id);

if (!problemsCategoriesOfEmployee) {

return res.status(404).json({ message: "Нет решаемых проблем" });

}

res.status(200).json(problemsCategoriesOfEmployee);

} catch (error) {

res.status(500).json({ message: error.message });

}

}

}

class ExcelController {

async generateReport(req, res) {

try {

const data = req.body || [];

const buffer = await ExcelService.generateExcelReport(data);

res.set({

'Content-Disposition': 'attachment; filename="report.xlsx"',

'Content-Type': 'application/vnd.openxmlformats-officedocument.wordprocessingml.document',

});

res.send(buffer);

} catch (error) {

console.error("Ошибка создания отчёта:", error);

res.status(500).json({ error: error.message });

}

}

}

class IncidentController {

async createIncident(req, res) {

const { incident\_date, description, req\_assignment } = req.body;

try {

const incident = await IncidentService.createIncident({ incident\_date, description, req\_assignment });

res.status(201).json(incident);

} catch (error) {

res.status(500).json({ message: error.message });

}

}

async getAllIncidents(req, res) {

try {

const incidents = await IncidentService.getAllIncidents();

res.status(200).json(incidents);

} catch (error) {

res.status(500).json({ message: error.message });

}

}

async getAllIncidentsWithPag(req, res) {

const { page = 1, limit = 10 } = req.query;

const offset = (page - 1) \* limit;

try {

const incidents = await IncidentService.getAllIncidentsWithPag(limit, offset);

return res.status(200).json({

total: incidents.count,

pages: Math.ceil(incidents.count / limit),

data: incidents.rows,

});

} catch (error) {

return res.status(500).json({ message: `Ошибка при получении несчастных случаев: ${error.message}`, });

}

}

async getIncidentById(req, res) {

const { incident\_id } = req.params;

try {

const incident = await IncidentService.getIncidentById(incident\_id);

if (!incident) {

return res.status(404).json({ message: "Инцидент не найден" });

}

res.status(200).json(incident);

} catch (error) {

res.status(500).json({ message: error.message });

}

}

async searchIncidentsByDescription(req, res) {

const { text, page = 1, limit = 10 } = req.query;

const offset = (page - 1) \* limit;

try {

const incidents = await IncidentService.searchIncidentsByDescription(limit, offset, text);

return res.status(200).json({

total: incidents.count,

pages: Math.ceil(incidents.count / limit),

data: incidents,

});

} catch (error) {

console.error("Ошибка при поиске несчастных случаев:", error);

return res

.status(500)

.json({ message: `Ошибка при поиске несчастных случаев: ${error}` });

}

}

async searchIncidentsByIds(req, res) {

const { id, page = 1, limit = 10 } = req.query;

const offset = (page - 1) \* limit;

try {

const incidents = await IncidentService.searchIncidentsByIds(limit, offset, id);

return res.status(200).json({

total: incidents.count,

pages: Math.ceil(incidents.count / limit),

data: incidents,

});

} catch (error) {

console.error("Ошибка при поиске несчастных случаев:", error);

return res

.status(500)

.json({ message: `Ошибка при поиске несчастных случаев: ${error}` });

}

}

async deleteIncident(req, res) {

const { incident\_id } = req.params;

try {

await IncidentService.deleteIncident(incident\_id);

return res.status(204).send()

} catch (error) {

if (error.message.includes('не найден')) {

res.status(404).json({ message: error.message })

}

else {

return res.status(500).json({ message: error.message });

}

}

}

}

class PDFController {

async generateReport(req, res) {

try {

const {data, name} = req.body || [];

const pdfBuffer = await PDFService.generatePDF(data, name);

res.setHeader('Content-Type', 'application/pdf');

res.setHeader('Content-Disposition', 'attachment; filename="report.pdf"');

res.send(pdfBuffer);

} catch (error) {

console.error("Ошибка создания отчёта:", error);

res.status(500).json({ error: error.message });

}

}

}

class ProblemCategoryController {

async getAllProblemCategories(req, res) {

try {

const problemCategories = await ProblemCategoryService.getAllProblemCategorys();

res.status(200).json(problemCategories);

} catch (error) {

res.status(500).json({ message: error.message });

}

}

async getProblemCategoryById(req, res) {

const { id } = req.params;

try {

const problemCategory = await ProblemCategoryService.getProblemCategoryById(id);

if (!problemCategory) {

return res.status(404).json({ message: "Категория проблемы не найден" });

}

res.status(200).json(problemCategory);

} catch (error) {

res.status(500).json({ message: error.message });

}

}

async getProblemCategoriesByIds(req, res){

const categoryIds = req.query.categoryIds.split(',').map(Number);

try {

const problemCategories = await ProblemCategoryService.getProblemCategoriesByIds(categoryIds);

if (!problemCategories) {

return res.status(404).json({ message: "Категории проблемы не найден" });

}

res.status(200).json(problemCategories);

} catch (error) {

res.status(500).json({ message: error.message });

}

}

async updateProblemCategory(req, res) {

const { id } = req.params;

const { name, description } = req.body;

try {

const updatedProblemCategory = await ProblemCategoryService.updateProblemCategory(id, {

name,

description

});

res.status(200).json(updatedProblemCategory);

} catch (error) {

res.status(500).json({ message: error.message });

}

}

async setName(req, res) {

const { id } = req.params;

const { name } = req.body;

try {

const updatedProblemCategory = await ProblemCategoryService.setName(id, name);

res.status(200).json(updatedProblemCategory);

} catch (error) {

res.status(500).json({ message: error.message });

}

}

async setDescription(req, res) {

const { id } = req.params;

const { description } = req.body;

try {

const updatedProblemCategory = await ProblemCategoryService.setDescription(id, description)

res.status(200).json(updatedProblemCategory);

} catch (error) {

res.status(500).json({ message: error.message });

}

}

}

class RequestAssignmentController {

async createRequestAssignment(req, res) {

const { assigned\_date, employee\_id, request\_id } = req.body;

try {

const requestAssignment = await RequestAssignmentService.createRequestAssignments({ assigned\_date, employee\_id, request\_id});

res.status(201).json(requestAssignment);

} catch (error) {

res.status(500).json({ message: error.message });

}

}

async getAllRequestAssignments(req, res) {

try {

const requestAssignments = await RequestAssignmentService.getAllRequestAssignments();

res.status(200).json(requestAssignments);

} catch (error) {

res.status(500).json({ message: error.message });

}

}

async getRequestAssignmentById(req, res) {

const { id } = req.params;

try {

const requestAssignment = await RequestAssignmentService.getRequestAssignmentsById(id);

if (!requestAssignment) {

return res.status(404).json({ message: "Назначение заявки сотруднику не найдено" });

}

res.status(200).json(requestAssignment);

} catch (error) {

res.status(500).json({ message: error.message });

}

}

async getRequestAssignmentByEmployeeAndRequestIds(req, res) {

const { request\_id, employee\_id } = req.params;

try {

const requestAssignment = await RequestAssignmentService.getRequestAssignmentsByEmployeeAndRequestIds(request\_id, employee\_id);

if (!requestAssignment) {

return res.status(404).json({ message: "Назначение заявки сотруднику не найдено" });

}

res.status(200).json(requestAssignment);

} catch (error) {

res.status(500).json({ message: error.message });

}

}

async deleteRequestAssignmentByEmployeeAndRequestIds(req, res) {

const { request\_id, employee\_id } = req.params;

try {

await RequestAssignmentService.deleteRequestAssignmentsByEmployeeAndRequestIds(request\_id, employee\_id);

return res.status(204).send()

} catch (error) {

if (error.message.includes('не найден')) {

res.status(404).json({ message: error.message })

}

else {

return res.status(500).json({ message: error.message });

}

}

}

}

class RequestController {

async getAllRequestsWithPag(req, res) {

const { page = 1, limit = 10 } = req.query;

const offset = (page - 1) \* limit;

try {

const requests = await RequestService.getAllRequestsWithPag(limit, offset);

return res.status(200).json({

total: requests.count,

pages: Math.ceil(requests.count / limit),

data: requests.rows,

});

} catch (error) {

return res.status(500).json({ message: `Ошибка при получении заявок: ${error.message}`, });

}

}

async getFilteredByPriorityAndSearchRequests(req, res) {

const {

priorities = ["Высокий", "Средний", "Низкий"],

search,

request\_status,

page = 1, limit = 10

} = req.query;

const offset = (page - 1) \* limit;

try {

let prioritiesArray;

if (!Array.isArray(priorities)) {

prioritiesArray = priorities.split(',').map(item => item.trim()).filter(item => item !== '');

}

else {

prioritiesArray = priorities

}

const requests = await RequestService.getFilteredByPriorityAndSearchRequests(limit, offset, prioritiesArray, search, request\_status);

return res.status(200).json({

total: requests.count,

pages: Math.ceil(requests.count / limit),

data: requests.rows,

});

} catch (error) {

return res.status(500).json({ message: `Ошибка при получении заявок: ${error.message}`, });

}

}

async getFilteredByPriorityAndSearchRequestsForEmployee(req, res) {

const {

employee\_id,

priorities = ["Высокий", "Средний", "Низкий"],

search,

request\_status,

page = 1, limit = 10

} = req.query;

const offset = (page - 1) \* limit;

try {

let prioritiesArray

if (!Array.isArray(priorities)) {

prioritiesArray = priorities.split(',').map(item => item.trim()).filter(item => item !== '');

}

else {

prioritiesArray = priorities

}

const requests = await RequestService.getFilteredByPriorityAndSearchRequestsForEmployee(limit, offset, prioritiesArray, search, employee\_id, request\_status);

return res.status(200).json({

total: requests.count,

pages: Math.ceil(requests.count / limit),

data: requests.rows,

});

} catch (error) {

return res.status(500).json({ message: `Ошибка при получении заявок: ${error.message}`, });

}

}

async getRequestById(req, res) {

const { request\_id } = req.params;

try {

const request = await RequestService.getRequestById(request\_id);

if (!request) {

return res.status(404).json({ message: "Заявка не найдена" });

}

res.status(200).json(request);

} catch (error) {

res.status(500).json({ message: error.message });

}

}

async getFullRequestInfoById(req, res) {

const { request\_id } = req.params;

try {

const request = await RequestService.getFullRequestInfoById(request\_id);

if (!request) {

return res.status(404).json({ message: "Заявка не найдена" });

}

res.status(200).json(request);

} catch (error) {

res.status(500).json({ message: error.message });

}

}

async getEmployeeRequests(req, res) {

const { employee\_id } = req.params;

try {

const requests = await RequestService.getEmployeeActiveRequests(employee\_id);

if (!requests) {

return res.status(404).json({ message: "Заявки не найдена" });

}

res.status(200).json(requests);

} catch (error) {

res.status(500).json({ message: error.message });

}

}

async getEmployeeDoneRequests(req, res) {

const { employee\_id } = req.params;

const { limit = 10, page = 1, search, start\_date, end\_date } = req.query;

const offset = (page - 1) \* limit;

try {

const requests = await RequestService.getEmployeeDoneRequests(employee\_id, limit, offset, search, start\_date, end\_date);

if (!requests) {

return res.status(404).json({ message: "Заявки не найдены" });

}

return res.status(200).json({

total: requests.count,

pages: Math.ceil(requests.count / limit),

data: requests.data,

});

} catch (error) {

res.status(500).json({ message: error.message });

}

}

async setStatus(req, res) {

const { request\_id } = req.params;

const { status } = req.body;

try {

const updatedRequest = await RequestService.setStatus(request\_id, status);

res.status(200).json(updatedRequest);

} catch (error) {

res.status(500).json({ message: error.message });

}

}

}

class UserController {

async getAllUsers(req, res) {

try {

const users = await UserService.getAllUsers();

res.status(200).json(users);

} catch (error) {

res.status(500).json({ message: error.message });

}

}

async getUserById(req, res) {

const { user\_id } = req.params;

try {

const user = await UserService.getUserById(user\_id);

if (!user) {

return res.status(404).json({ message: "Пользователь не найден" });

}

res.status(200).json(user);

} catch (error) {

res.status(500).json({ message: error.message });

}

}

async getFullName(req, res) {

const { user\_id } = req.params;

try {

const fullname = await UserService.getFullName(user\_id);

if (!fullname) {

return res.status(404).json({ message: "Пользователь не найден" });

}

res.status(200).json(fullname);

} catch (error) {

res.status(500).json({ message: error.message });

}

}

}

class WorkTimeLogController {

async createWorkTimeLog(req, res) {

const { start\_time, end\_time, req\_assignments } = req.body;

try {

const workTimeLog = await WorkTimeLogService.createWorkTimeLog({ start\_time, end\_time, req\_assignments });

res.status(201).json(workTimeLog);

} catch (error) {

res.status(500).json({ message: error.message });

}

}

async getAllWorkTimeLogs(req, res) {

try {

const workTimeLogs = await WorkTimeLogService.getAllWorkTimeLogs();

res.status(200).json(workTimeLogs);

} catch (error) {

res.status(500).json({ message: error.message });

}

}

async getWorkTimeLogById(req, res) {

const { id } = req.params;

try {

const workTimeLog = await WorkTimeLogService.getWorkTimeLogById(id);

if (!workTimeLog) {

return res.status(404).json({ message: "Запись о времени работы не найдена" });

}

res.status(200).json(workTimeLog);

} catch (error) {

res.status(500).json({ message: error.message });

}

}

async getDurationInMinutes(req, res) {

const { id } = req.params;

try {

const duration = await WorkTimeLogService.getDurationInMinutes(id);

if (!duration) {

return res.status(404).json({ message: "Запись о времени работы не найдена" });

}

res.status(200).json(duration);

} catch (error) {

res.status(500).json({ message: error.message });

}

}

async deleteWorkTimeLog(req, res) {

const { id } = req.params;

try {

await WorkTimeLogService.deleteWorkTimeLog(id);

return res.status(204).send()

} catch (error) {

if (error.message.includes('не найден')) {

res.status(404).json({ message: error.message })

}

else {

return res.status(500).json({ message: error.message });

}

}

}

}