README

Проект: Система управления заявками на ремонт в жилом комплексе

Краткое описание:

В рамках проекта реализуется система, с помощью которой жильцы могут подавать заявки на ремонт/обслуживание (например, "протекает кран", "не работает лифт"), а управляющая компания - обрабатывать, отслеживать и закрывать заявки.

Этап 1. Сбор и анализ требований

1.1. Бизнес-требования

1. Оптимизировать получение заявок от жителей:
   - Минимизация времени на оформление заявки (менее 3 минут).
   - Упрощение процесса подачи через мобильное приложение/Telegram бота.
2. Повышение удовлетворенности жильцов:
   - Внедрение системы оценки качества выполнения работ.
   - Уменьшение повторных обращений по одной проблеме на 20%.
3. Аналитика:
   - Генерация отчетов по затратам, времени выполнения, топ-5 частых проблем.

1.2. Пользовательские роли и потребности

Роль	Цели и задачи
Житель	Создавать заявки, оценивать качество предоставляемых услуг, просматривать статус заявки в реальном времени, получать уведомления
Сотрудник УК	Просматривать заявки, фильтровать заявки по срочности/категории, назначать ответственного сотрудника, изменять статус заявки, добавлять комментарии и фотоотчеты

Роль	Цели и задачи
Администратор	Управлять списком сотрудников, просматривать статистику, настраивать категории заявок, управлять правами доступа сотрудников, просматривать историю изменений заявки

1.3. User Stories

1. Как жилец, я хочу подать заявку за 5 кликов в Telegram-боте, чтобы сэкономить время.
2. Как жилец, я хочу прикрепить фото/видео к заявке, чтобы показать проблему.
3. Как жилец, я хочу выбрать срочность заявки (аварийная/ стандартная), чтобы приоритезировать ее выполнение.
4. Как жилец, я хочу видеть статус заявки и получать уведомления при изменении статуса, чтобы понимать, как продвигается ее обработка.
5. Как жилец, я хочу оценить качество обработки заявки оценкой от 1 до 5 звезд и оставить комментарий, чтобы сотрудники УК получали обратную связь.
6. Как сотрудник УК, я хочу просмотреть заявки, сортировать их по категориям, приоритету и дате, чтобы заявки обработать.
7. Как сотрудник УК, я хочу назначить ответственного за выполнение заявки, чтобы человек знал, что именно он должен ею заняться.
8. Как сотрудник УК, я хочу изменить статус заявки с автоматической отправкой уведомления жильцу, чтобы оповестить его об изменение статуса.
9. Как сотрудник УК, я хочу добавить комментарии, чтобы у жильца было больше понимания о статусе заявки.
10. Как администратор, я хочу настраивать роли и права доступа, управлять списком сотрудников, чтобы добавить/удалить нового/уволенного сотрудника.
11. Как администратор, я хочу посмотреть статистику по среднему времени выполнения заявок, затратам, рейтингу, чтобы сделать отчет начальству.
12. Как администратор, я хочу настраивать категории заявок, чтобы система автоматически их распределяла.
13. Как администратор, я хочу экспортировать данные в Excel, чтобы использовать их для бухгалтерии.
14. Как администратор, я хочу просматривать историю изменений заявки, чтобы понимать кто и когда ее обрабатывал.

1.4. Функциональные требования

1.4.1. Общие требования

1.4.1.1. Система должна обеспечивать авторизацию и аутентификацию пользователей по телефону, email или Telegram ID.
1.4.1.2. Система должна определять роль пользователя после входа: Жилец, Сотрудник, Администратор.
1.4.1.3. Система должна отображать интерфейс в зависимости от роли пользователя.
1.4.1.4. Все действия пользователей должны логироваться в системе.

1.4.2. Функциональность для жильца

1.4.2.1. Система должна позволять пользователю создать заявку, указав:

- категорию,
- краткое название,
- описание,
- приоритет,
- вложения (фото/видео).
  1.4.2.2. Система должна отправлять уведомление пользователю при изменении статуса его заявки.
  1.4.2.3. Система должна отображать пользователю список его заявок с возможностью фильтрации по статусу.
  1.4.2.3. Система должна позволять пользователю просматривать детали заявки.
  1.4.2.4. Система должна позволять пользователю оставлять оценку и комментарий после выполнения заявки.
  1.4.2.5. Система должна предотвращать отправку пустых или некорректных заявок.

1.4.3. Функциональность для сотрудника управляющей компании

1.4.3.1. Система должна предоставлять сотруднику доступ к списку всех заявок.
1.4.3.2. Система должна позволять фильтровать заявки по статусу, приоритету, категории и дате создания.
1.4.3.3. Система должна позволять сотруднику назначить ответственного за заявку.
1.4.3.4. Система должна позволять сотруднику изменять статус заявки:

- Новая → В работе,
- В работе → Выполнена / Отклонена.
  1.4.3.5. Система должна позволять сотруднику оставлять внутренние комментарии к заявке.
  1.4.3.6. Система должна уведомлять сотрудников о новых заявках, если они не были назначены вручную.

1.4.4. Функциональность для администратора

1.4.4.1. Система должна предоставлять администратору интерфейс управления списком сотрудников (создание, редактирование, деактивация).
1.4.4.2. Система должна позволять просматривать статистику:

- количество заявок по статусам,
- среднее время выполнения,
- рейтинг качества обслуживания,
- количество заявок по сотруднику, категории, приоритету.

1.4.4.3. Система должна позволять администратору формировать и экспортировать отчёты за заданный период в формате PDF/Excel.
1.4.4.4. Система должна позволять администратору просматривать историю действий пользователей в системе (аудит-лог).

1.4.5. Система уведомлений

1.4.5.1. Система должна автоматически отправлять уведомления пользователям о:

- новых комментариях,
- изменении статуса заявки
  1.4.5.2. Уведомления должны отправляться через email или Telegram-бота в зависимости от настроек пользователя.

1.4.6. Дополнительные требования

1.4.6.1. Система должна сохранять все данные в БД с возможностью последующего анализа.

1.4.6.2. Система должна быть адаптивной и корректно работать на мобильных устройствах.
1.4.6.3. Система должна иметь защиту от несанкционированного доступа.
1.4.6.4. Вся передача данных должна происходить по защищённому протоколу HTTPS.

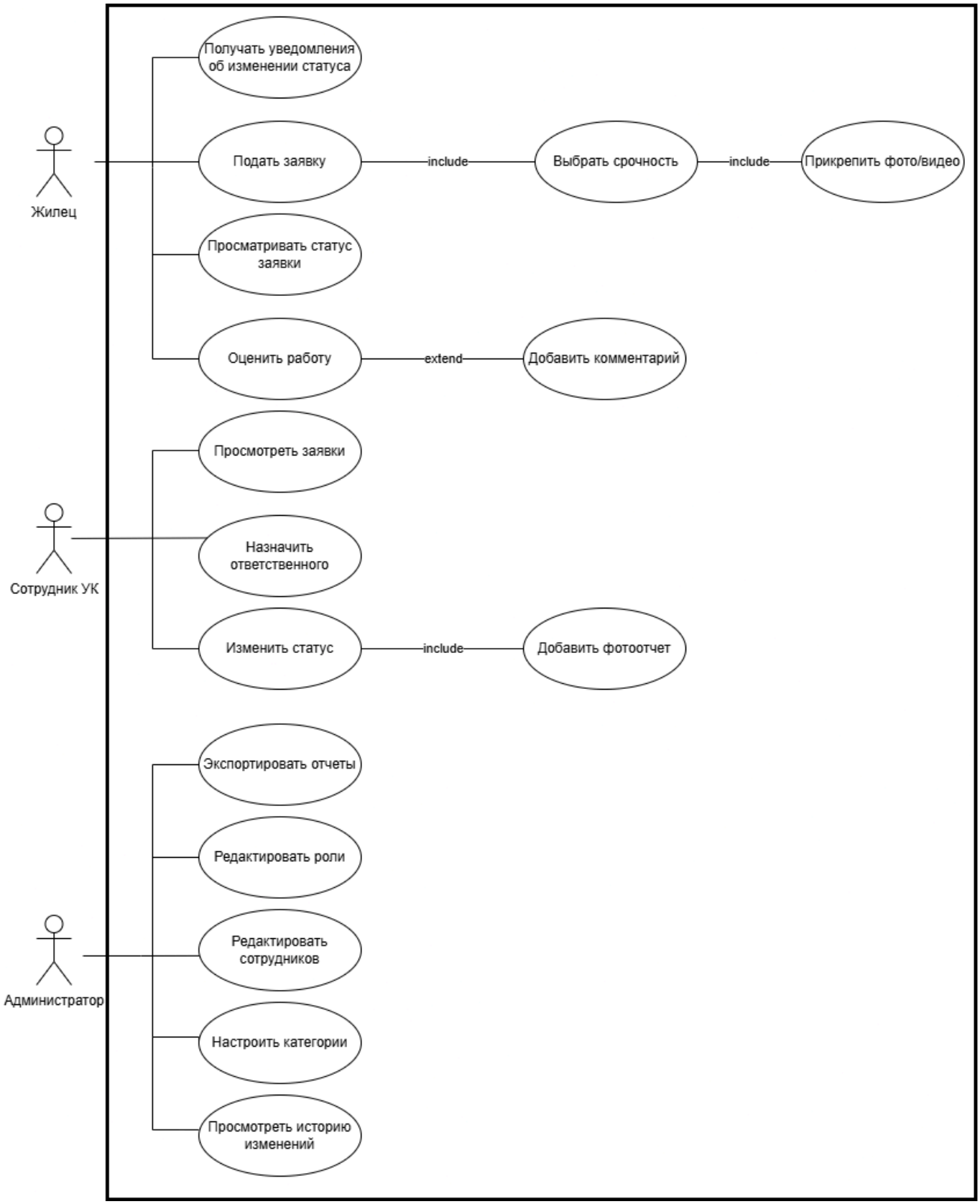
1.5. Нефункциональные требования

- Время отклика не более 3 секунд.
- Поддержка до 100 одновременных пользователей.
- Обработка до 20 новых заявок в минуту.
- Авторизация по ролям.
- Надежность (99,9% uptime)
- Хранение персональных данных только с согласия пользователей.
- Создание заявки — не более 5 шагов.

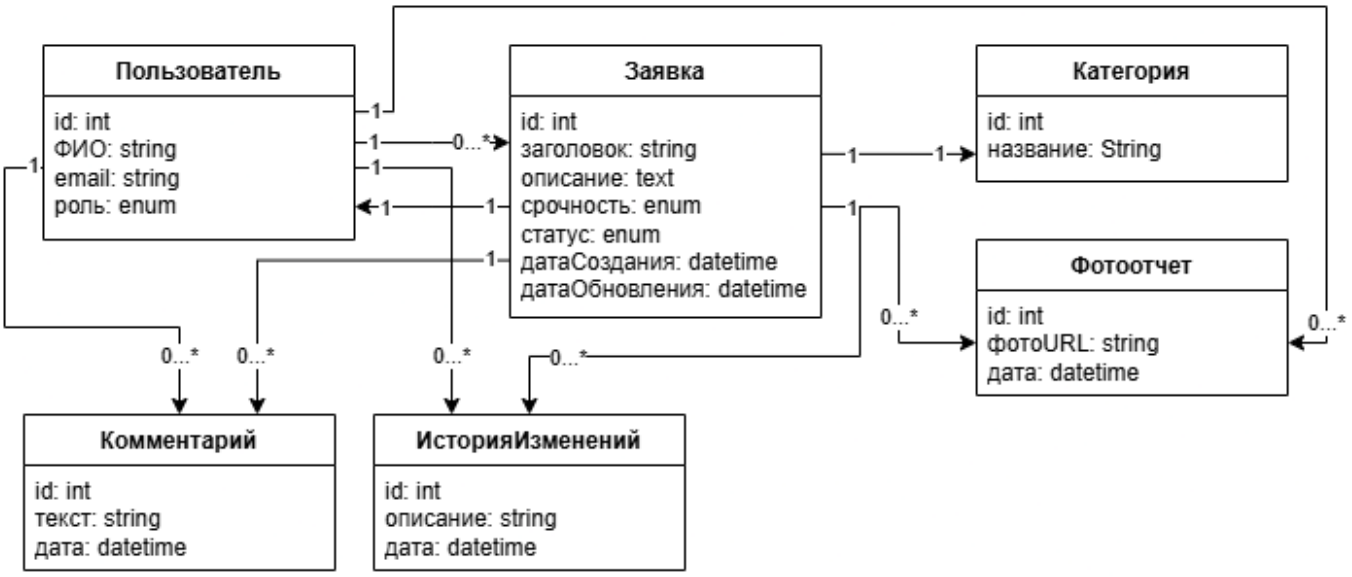
Этап 2. Моделирование

Все исходные диаграммы можно найти в docs/diagrams/source, изображения - docs/diagrams/png.

2.1. Use Case диаграмма

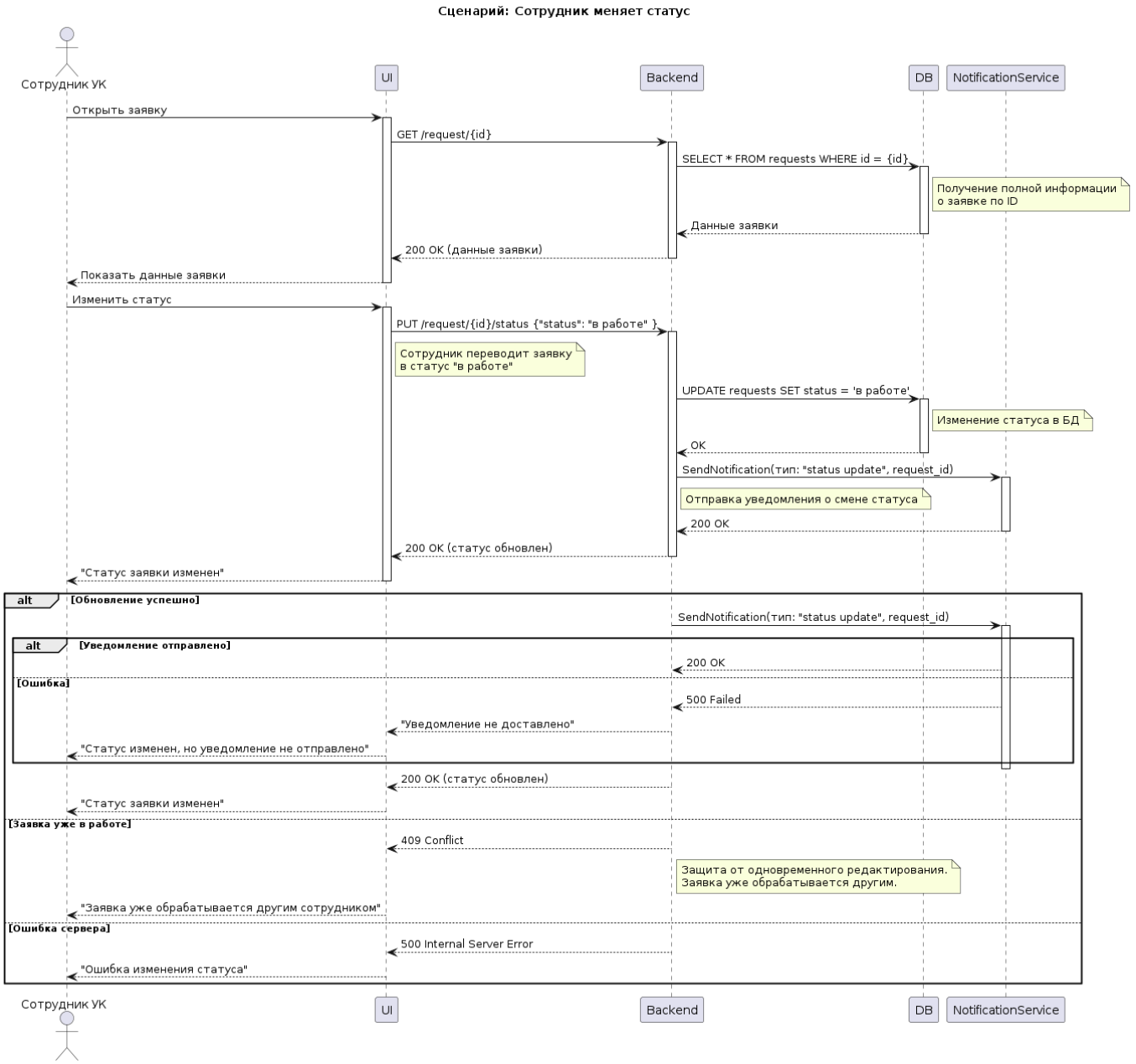


2.2. Class diagram (Диаграмма классов)

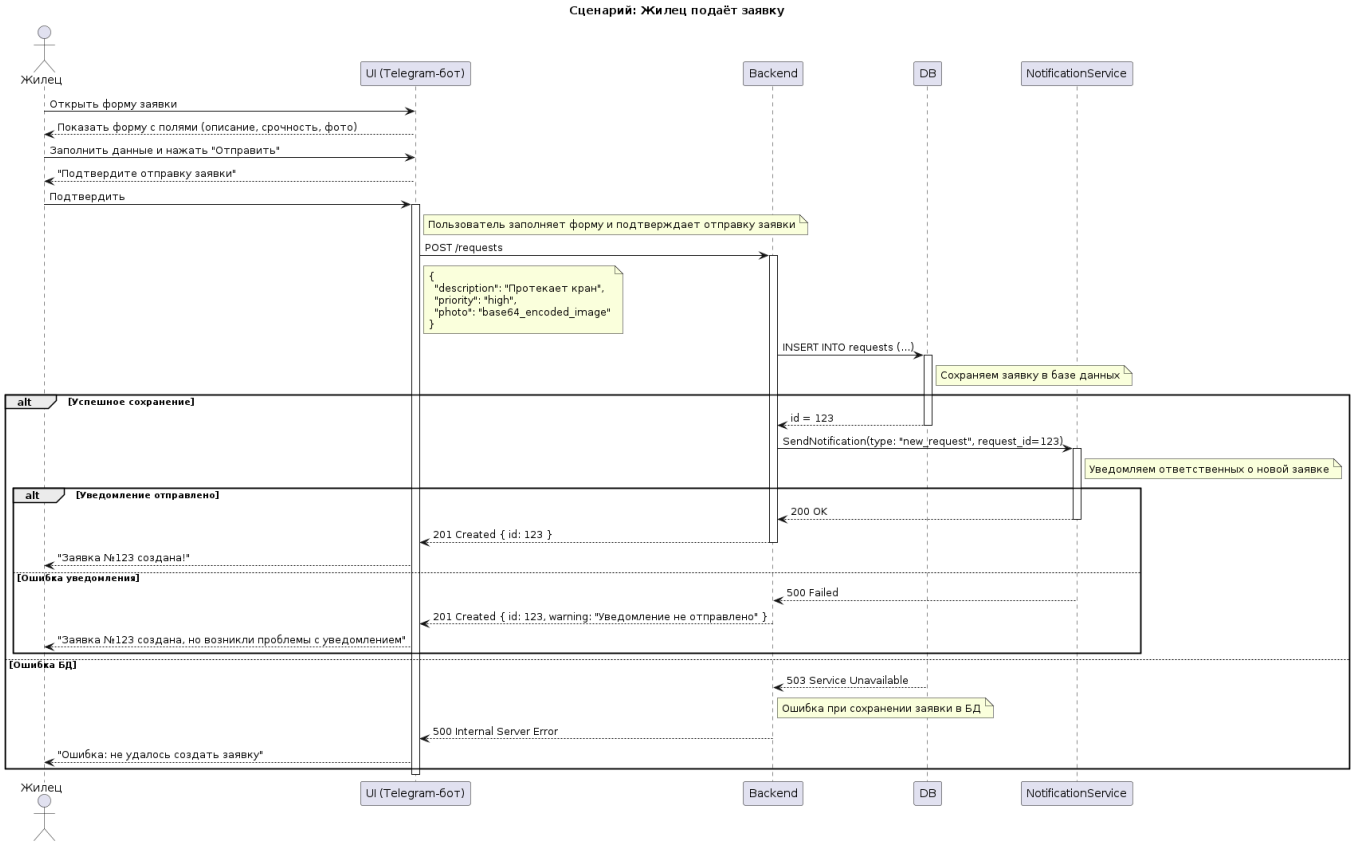


2.3. Sequence diagram (Диаграмма последовательности)

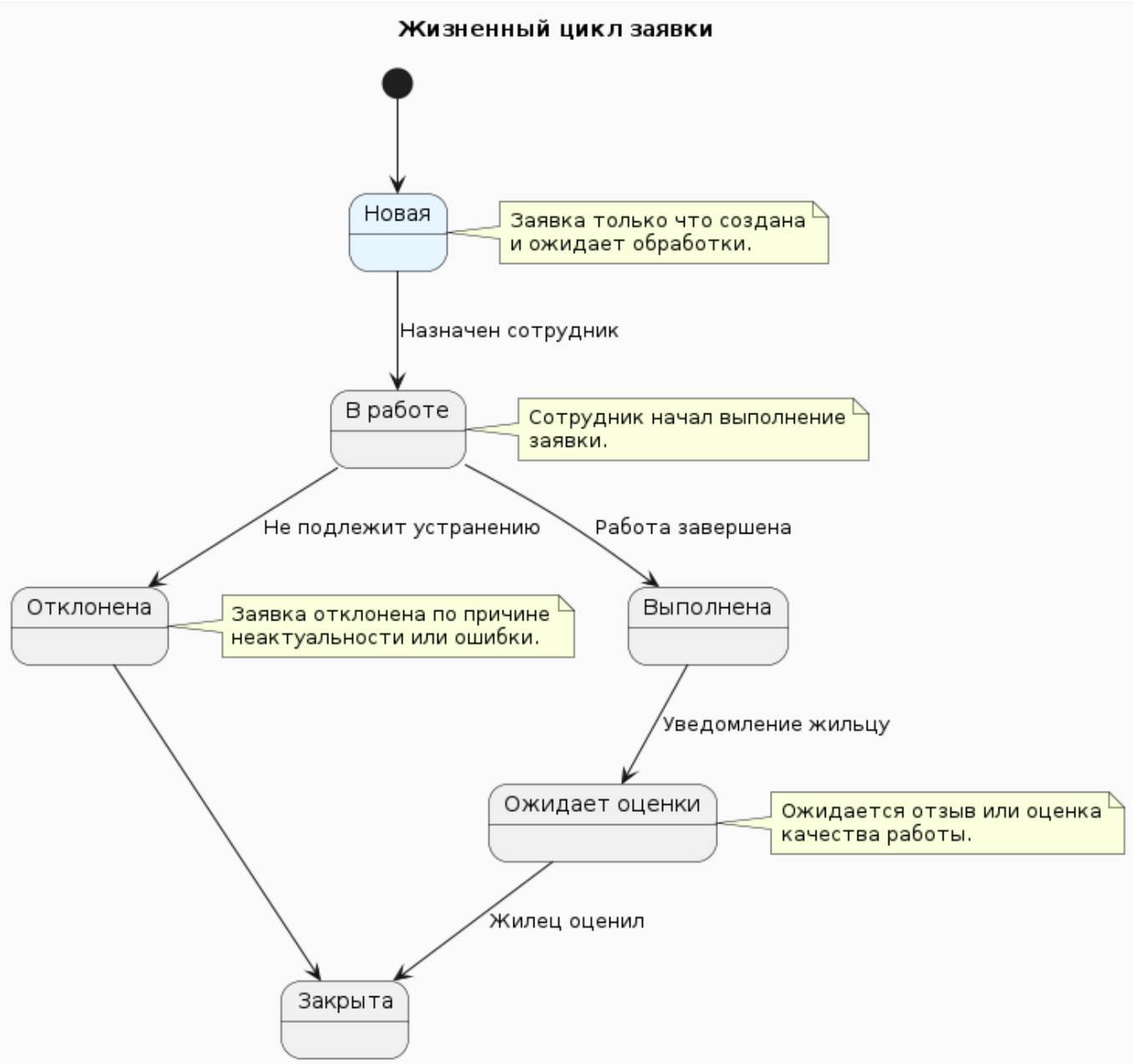
2.3.1. Сотрудник меняет статус



2.3.2. Жилец подает заявку



2.4. State diagram (Диаграмма состояний)



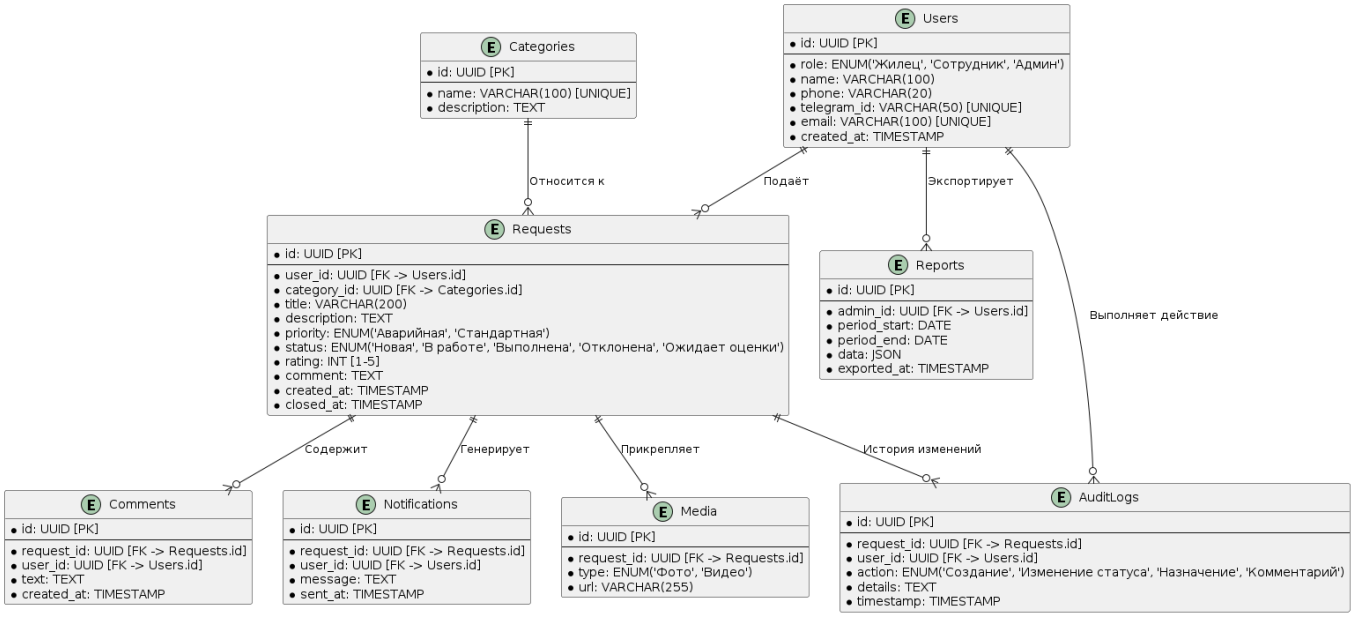
2.5. Activity diagram (Диаграмма активности)

2.5.1. Создание заявки

2.5.2. Обработка заявки сотрудником УК

2.5.3. Экспорт отчетов админом

2.6. ER-диаграмма



Этап 3. Проектирование API

3.1. Структура API

Основные сущности

Сущность	Описание	Методы
Заявки	Управление заявками на ремонт	GET, POST, DELETE
Пользователи	Управление аккаунтами	GET, POST, PUT, DELETE
Категории	Группировка заявок по типам	GET, POST, DELETE
Комментарии	Обсуждение задач	GET, POST, DELETE

3.2. Ключевые эндпоинты

3.2.1. Заявки

```
POST /requests
```

- **Описание**: Создание новой заявки с прикреплением файлов
- **Параметры**:

```
{
  "user_id": "uuid",
  "category_id": "uuid",
  "title": "Протекает кран",
  "priority": "Аварийная",
  "attachments": ["https://example.com/photo.jpg"]
}
```

```
GET /requests?status=В работе&category_id=uuid
```

- **Фильтры**:
  - `status` : Статус заявки (Новая/В работе/Выполнена/Отклонена/Ожидает оценки)
  - `category_id` : Фильтр по категории
  - Пагинация: `page=1` , `page_size=20`

3.2.2. Комментарии

```
POST /requests/{id}/comments
```

- **Требования**:
  - Текст комментария: до 1000 символов
  - Автоматическая привязка к автору через JWT

3.2.3. Пользователи

```
POST /users
```

- **Права**: Только для администраторов
- **Пример тела**:

```
{
  "email": "user@example.com",
  "role": "Сотрудник"
}
```

3.3. Модели данных

3.3.1. Заявка (Request)

```
id: uuid
status:
  type: string
  enum: [Новая, В работе, Выполнена, Отклонена, Ожидает оценки]
  Описание: |
    - Новая: Создана, но не назначена
    - В работе: Назначен ответственный
    - Выполнена: Требует подтверждения
created_at: datetime
```

3.3.2. Пользователь (User)

```
id: uuid
role:
  type: string
  enum: [Жилец, Сотрудник, Админ]
  Права:
    - Жилец: Создание заявок, просмотр своих задач
    - Сотрудник: Полный доступ к заявкам
    - Админ: Управление системой
email: string (format: email)
```

3.4. Безопасность

3.4.1. Аутентификация

```
BearerAuth:
  type: http
  scheme: bearer
  bearerFormat: JWT
  Требуемые scope:
    - Жилец: user
    - Сотрудник: user, employee
    - Админ: user, employee, admin
```

3.4.2. Права доступа

Роль	Доступные операции
Жилец	Свои заявки, комментарии
Сотрудник УК	Все заявки, изменение статусов
Админ	Полный доступ к системе

3.5. Обработка ошибок

3.5.1. Формат ответа

```
{
  "code": "validation_error",
  "message": "Ошибка в поле email",
  "errors": [
    {
      "field": "email",
      "error": "Некорректный формат",
      "rule": "RFC 5322"
    }
  ]
}
```

3.5.2. Типовые ошибки

Код состояния	Пример ошибки
400	Некорректный формат email
403	Недостаточно прав для операции
404	Заявка не найдена
500	Ошибка подключения к БД

3.6. Примеры сценариев

3.6.1. Создание заявки

Запрос:

```
POST /requests
Authorization: Bearer <JWT>
```

```
{
  "user_id": "550e8400-e29b-41d4-a716-446655440000",
```

```
  "category_id": "3d3e3f40-4a4b-4c4d-4e4f-505152535455",
  "title": "Сломан лифт",
  "priority": "Аварийная"
}
```

Ответ:

```
{
  "id": "6ba7b814-9dad-11d1-80b4-00c04fd430c8",
  "status": "Новая",
  "created_at": "2024-01-01T12:00:00Z"
}
```

3.6.2. Ошибка валидации

```
{
  "code": "validation_error",
  "message": "Обязательные поля отсутствуют",
  "errors": [
    {
      "field": "title",
      "error": "Поле обязательно для заполнения"
    }
  ]
}
```

Выводы

В рамках реализации проекта **«Система управления заявками на ремонт в жилом комплексе»** были достигнуты следующие результаты:

1. **Проведен полный анализ требований**, включая сбор бизнес-требований, пользовательских историй и функциональных ожиданий от ключевых ролей (житель, сотрудник УК, администратор). Это обеспечило четкое понимание целей системы и путей их достижения.
2. **Разработана подробная документация**, охватывающая:
   - бизнес-требования и пользовательские истории;
   - функциональные и нефункциональные требования;

- диаграммы моделей поведения и структуры (Use Case, Class, Activity, Sequence, State, ER-диаграмма);
- структуру API и основные эндпоинты.

3. **Определена архитектура системы**, обеспечивающая:
   - масштабируемость;
   - безопасную работу с персональными данными;
   - поддержку работы через Telegram-бота и веб-интерфейс;
   - адаптивность для мобильных устройств.
4. **Уделено внимание аналитике и автоматизации**, что позволит управляющей компании:
   - отслеживать эффективность работы сотрудников;
   - получать регулярные отчеты по заявкам;
   - проводить анализ частых проблем в жилом комплексе.
5. **Разработанная система поможет улучшить коммуникацию между жильцами и управляющей компанией**, повысит прозрачность обслуживания, сократит время отклика на заявки и повысит уровень удовлетворенности жильцов.

Таким образом, проект закладывает прочную основу для цифровизации процессов обслуживания в жилом комплексе и может быть масштабирован на другие объекты ЖКХ.