Система учета и проверки технического обслуживания оборудования.

Цель проекта:

Автоматизация процессов планирования, выполнения и контроля технического обслуживания оборудования с интеграцией в учетные системы предприятия (1С/ERP).
Инструменты: PostgreSQL, Swagger, JSON Schema, XML/XSD, 1С, Draw.io, Git, REST, Python.

Этап 1. Сбор и анализ требований

1.1 Бизнес-требования (цели проекта)

- Избавиться от ручного ведения ТО
- Исключить забытые ТО, просроченные задачи
- Автоматизировать уведомления подрядчикам
- Хранить документы и акты о выполненных работах
- Интегрировать учёт ТО с 1С или ERP

1.2 Пользовательские роли и потребности

Роль	Цели и задачи
Администратор	Назначать задачи ТО, контролировать подрядчиков, загружать документы
Подрядчик	Получать уведомления о задачах, подтверждать выполнение, прикладывать отчёт
Сотрудник	Проверять статус оборудования, создавать запросы на внеплановое ТО

1.3. User Stories

---

1. Как администратор, я хочу видеть все задачи ТО, чтобы контролировать исполнение.
2. Как подрядчик, я хочу получать уведомления о новых задачах, чтобы вовремя выполнять работы.
3. Как сотрудник, я хочу проверять статус оборудования, чтобы убедиться в его исправности.
4. Как администратор, я хочу прикреплять файлы актов, чтобы сохранить документацию.
5. Как подрядчик, я хочу загружать фото оборудования после выполнения ТО.
6. Как администратор, я хочу экспортировать данные о ТО для отчётности.
7. Как сотрудник, я хочу подать заявку на внеплановое ТО.
8. Как подрядчик, я хочу отклонить задачу, если она неактуальна.
9. Как администратор, я хочу фильтровать задачи по статусу и подрядчику.
10. Как администратор, я хочу назначать задачи на основании графика обслуживания.

1.4. Функциональные требования

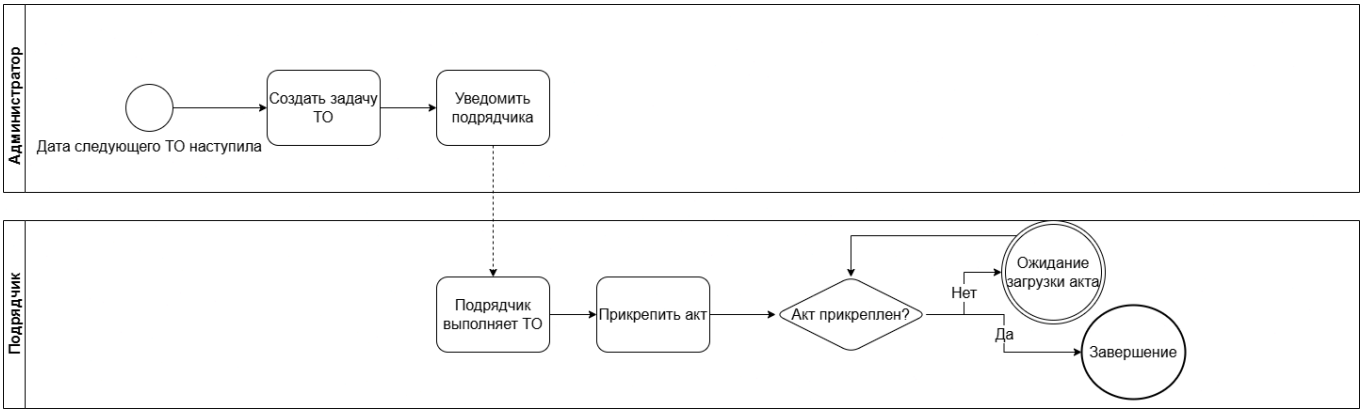
- Создание и редактирование оборудования
- Назначение задач ТО
- Назначение и уведомление подрядчика
- Прикрепление файлов (документы, фото)
- Уведомления по email / webhook
- Просмотр статуса оборудования
- Экспорт отчётов

1.5. Нефункциональные требования

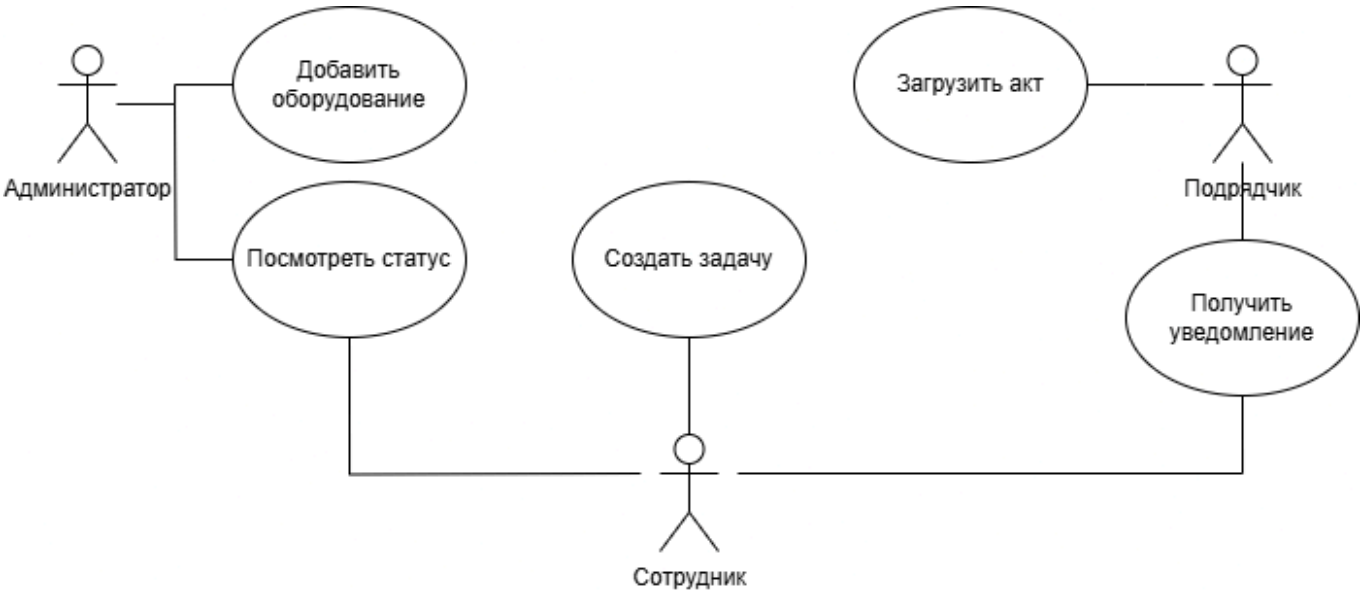
- Время отклика не более 2 секунд
- Надежность (99,9% uptime)
- Авторизация по ролям
- Кэширование запросов к списку задач
- Поддержка браузеров Chrome, Safari, Edge

Этап 2. Моделирование

2.1 BPMN: "Проведение ТО"

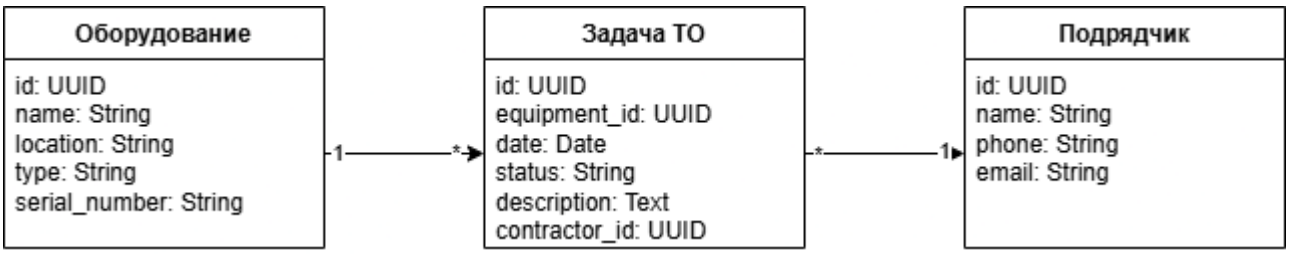


2.2 Use Case диаграмма



2.3 Sequence диаграмма

2.4 Class диаграмма



Этап 3. Проектирование базы данных и SQL

3.1 Анализ процессов

На основании **BPMN/Use Case** модели выделены ключевые процессы:

- Регистрация оборудования
- Создание и планирование задач ТО
- Назначение задач подрядчикам
- Отправка уведомлений
- Приём и проверка актов выполненных работ
- Хранение истории ТО
- Мониторинг и отчетность

3.2 Выделение сущностей

Основные сущности:

- Оборудование
- Задача ТО
- Тип ТО (регламентная, внеплановая и т.д.)
- Подрядчик
- Уведомление
- Акт выполненных работ
- Пользователь (внутренний сотрудник)

3.3 ER-диаграмма (сущности и связи)

Связи:

- Оборудование — 1 ко многим — Задачи ТО
- Задача ТО — 1 ко многим — Уведомления
- Задача ТО — 1 к 1 — Акт (может быть null)
- Задача ТО — многие к одному — Подрядчик
- Задача ТО — многие к одному — Тип ТО
- Задача ТО — многие к одному — Пользователь (создатель)

3.4 Атрибуты сущностей

---

equipment (Оборудование)

- id (PK)
- name
- inventory_numer
- location
- installed_at (дата установки)
- status (в эксплуатации, списано, на складе)
  maintenance_task (Задача ТО)
- id (PK)
- equipment_id (FK)

- contractor_id (FK)
- user_id (FK)
- type_id (FK)
- status (запланирована, выполнена, просрочена)
- planned_date
- completed_date
- description

maintenance_type (Тип ТО)

- id (PK)
- name (регламентное, внеплановое, разовое и т.д.)

contractor (Подрядчик)

- id (PK)
- name
- contact_name
- phone
- email

notification (Уведомление)

- id (PK)
- task_id (FK)
- type (email, sms, web)
- sent_at
- message

act (Акт выполненных работ)

- id (PK)
- task_id (FK)
- act_number
- file_url
- signed_at

user (Пользователь)

- id (PK)
- name
- role (admin, manager, viewer)
- email
- password_hash

3.5 Выбор СУБД

PostgreSQL выбран как СУБД, обеспечивающая:

- **Надёжность** для исключения потерь данных о ТО.
- **Гибкость** для интеграции с ERP и кастомизации.
- **Производительность** для работы с 1000+ задач в секунду.
- **Безопасность** ролевой модели, соответствующей требованиям (администратор, подрядчик, сотрудник).

3.6 SQL-структура (DDL)

```
CREATE TABLE equipment (
  id SERIAL PRIMARY KEY,
  name TEXT NOT NULL,
  inventory_number TEXT UNIQUE NOT NULL,
  location TEXT,
  installed_at DATE,
  status TEXT CHECK (status IN ('active', 'decommissioned', 'storage'))
);

CREATE TABLE maintenance_type (
  id SERIAL PRIMARY KEY,
  name TEXT NOT NULL
);

CREATE TABLE contractor (
  id SERIAL PRIMARY KEY,
  name TEXT NOT NULL,
  contact_name TEXT,
  phone TEXT,
  email TEXT
);

CREATE TABLE "user" (
  id SERIAL PRIMARY KEY,
  name TEXT NOT NULL,
  role TEXT CHECK (role IN ('admin', 'manager', 'viewer')),
  email TEXT UNIQUE NOT NULL,
  password_hash TEXT NOT NULL
);

CREATE TABLE maintenance_task (
  id SERIAL PRIMARY KEY,
```

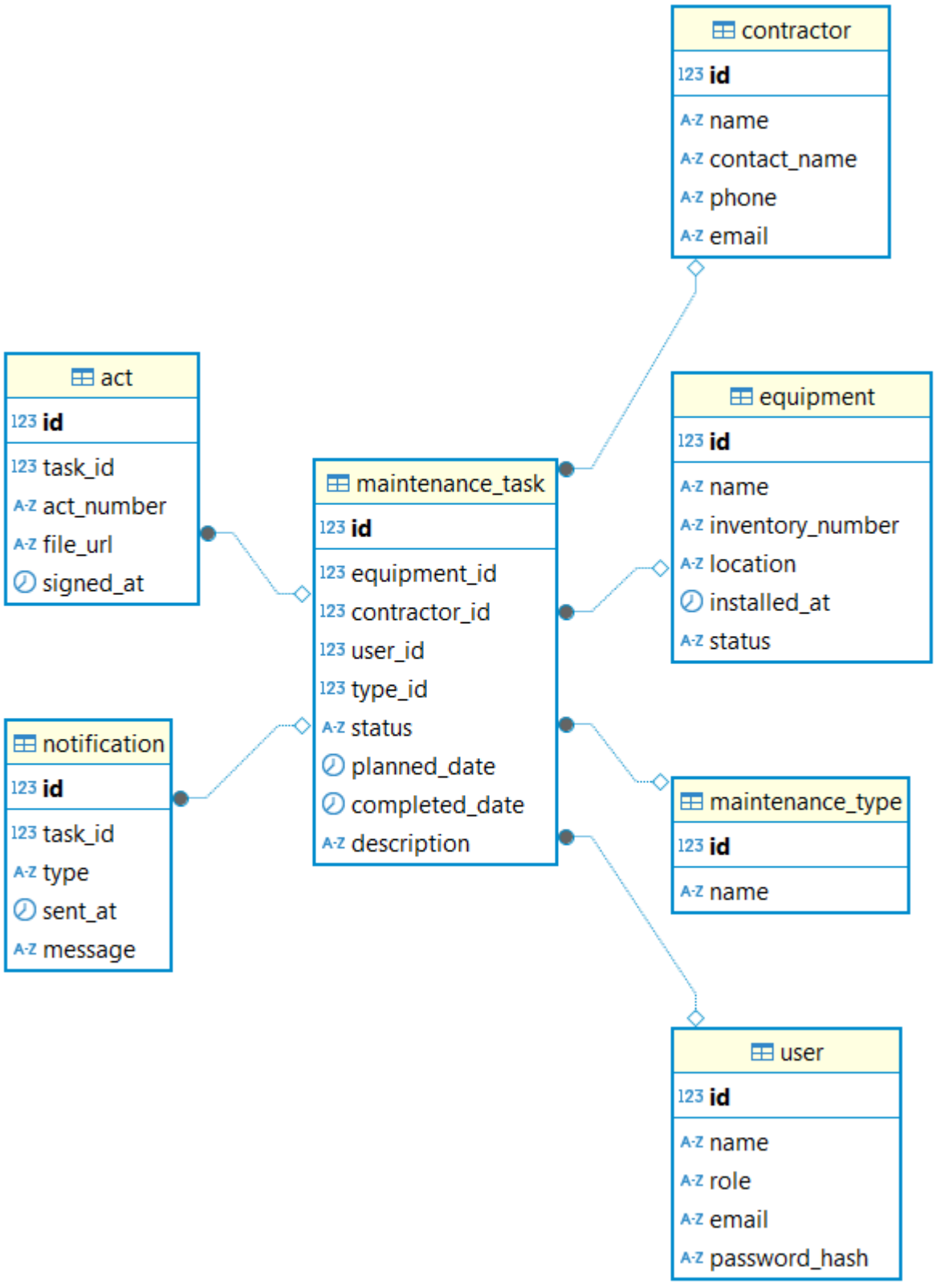
```
  equipment_id INT REFERENCES equipment(id),
  contractor_id INT REFERENCES contractor(id),
  user_id INT REFERENCES "user"(id),
  type_id INT REFERENCES maintenance_type(id),
  status TEXT CHECK (
    status IN (
      'planned',
      'in_progress',
      'completed',
      'overdue',
      'canceled',
      'awaiting_review'
    )
  ),
  planned_date DATE,
  completed_date DATE,
  description TEXT
);

CREATE TABLE notification (
  id SERIAL PRIMARY KEY,
  task_id INT REFERENCES maintenance_task(id),
  type TEXT CHECK (type IN ('email', 'sms', 'web')),
  sent_at TIMESTAMP,
  message TEXT
);

CREATE TABLE act (
  id SERIAL PRIMARY KEY,
  task_id INT UNIQUE REFERENCES maintenance_task(id),
  act_number TEXT,
  file_url TEXT,
  signed_at DATE
);
```

3.7 Документирование таблиц

- **equipment** — оборудование, подлежащее обслуживанию
  Поля: название, инвентарный номер, местоположение, дата установки, статус (`active`, `decommissioned`, `storage`).
- **maintenance_task** — задача ТО
  Поля:

    - Связи: оборудование (`equipment_id`), подрядчик (`contractor_id`), пользователь (`user_id`), тип ТО (`type_id`).
    - Статусы: `planned` , `in_progress` , `completed` , `overdue` , `canceled` , `awaiting_review` .
    - Даты: плановая (`planned_date`), фактическая (`completed_date`).
    - Описание: текстовое поле для деталей задачи.
- **maintenance_type** — тип задачи ТО (регламентное, внеплановое и др.).
- **contractor** — подрядная организация, выполняющая ТО.
  Поля: название, контактное лицо, телефон, email.
- **user** — внутренний пользователь системы.
  Поля: имя, роль (`admin` , `manager` , `viewer`), email, хэш пароля.
- **notification** — уведомление о задаче ТО.
  Поля: тип (`email` , `sms` , `web`), время отправки, текст сообщения.
- **act** — акт выполненных работ по задаче ТО.
  Поля: номер акта, ссылка на файл, дата подписания.

3.8 ER-диаграмма с атрибутами сущностей



Этап 4. Проектирование API (REST)

Цель этапа:
Создать API-интерфейс для взаимодействия с системой. REST API позволит:

- Управлять оборудованием и задачами ТО
- Интегрироваться с подрядчикам
- Загружать акты выполненных работ
- Получать уведомление и отчеты

4.1 Определение ключевых методов REST API

Методы для работы с оборудованием:

- `GET /equipment` — получить список оборудования
- `POST /equipment` — добавить новое оборудование
- `PATCH /equipment/:id` — отредактировать данные оборудования

Методы задач ТО:

- `GET /tasks` — получить все задачи ТО
- `POST /tasks` — создать новую задачу ТО
- `PATCH /tasks/:id` — изменить статус или подрядчика
- `DELETE /tasks/:id` — удалить задачу

Работа с файлами:

- `POST /tasks/:id/files` — загрузить файл (акт)
- `GET /tasks/:id/files` — получить список файлов по задаче

Уведомления и отчётность:

- `GET /alerts` — показать задачи с истекающим сроком
- `GET /reports?month=4` — выгрузка задач за месяц (фильтрация по параметрам)

4.2 Swagger документация (YAML)

Для интеграции и работы с системой управления ТО разработан REST API. Документация оформлена в формате OpenAPI 3.0 и включает следующие ключевые элементы:

4.2.1 Структура API

```yaml
servers:
  - url: https://api.maintenance.local/v1  # Базовый URL
paths:
  /equipment:           # Работа с оборудованием
  /tasks:               # Управление задачами ТО
  /tasks/{id}/files:    # Загрузка файлов
  /alerts:              # Просмотр просроченных задач
  /reports:             # Генерация отчётов
```

4.2.2 Основные эндпоинты

1. Управление оборудованием (`/equipment`)

Метод	Описание	Пример запроса
GET	Получить список оборудования	`GET /equipment`
POST	Добавить новое оборудование	`json {   "name": "Насос X200",   "inventory_number": "INV-045" }`

Схема Equipment:

```yaml
Equipment:
  type: object
  required: [name, inventory_number]
  properties:
    name: string          # Название
    inventory_number: string  # Инвентарный номер
    status:
      enum: [active, decommissioned, storage]  # Статус
```

---

2. Работа с задачами ТО (`/tasks`)

Метод	Описание	Параметры
GET	Фильтрация задач (по статусу/ дате)	`?status=completed&month=11`
POST	Создать задачу ТО	`json {   "equipment_id": 15,   "planned_date": "2023-12-01" }`

Схема MaintenanceTask:

```yaml
MaintenanceTask:
  type: object
  required: [equipment_id, planned_date]
  properties:
    status:
      enum: [planned, completed, overdue]  # Статус задачи
    planned_date: date  # Плановая дата выполнения
```

---

3. Работа с файлами (`/tasks/{id}/files`)

Метод	Описание	Формат данных
POST	Загрузить файл (акт, фото)	`multipart/form-data`
GET	Получить список файлов	`GET /tasks/45/files`

---

4. Отчёты (`/reports`)

```yaml
/reports:
  get:
    summary: Выгрузка задач за месяц
    parameters:
      - name: month  # Номер месяца (1-12)
        in: query
    responses:
```

```
      '200':
        description: CSV/Excel файл с данными
```

4.2.3 Пример запроса

```
POST /tasks HTTP/1.1
Content-Type: application/json

{
  "equipment_id": 15,
  "contractor_id": 8,
  "planned_date": "2023-12-05",
  "type_id": 2
}
```

Пример ответа

```
{
  "id": 189,
  "status": "planned",
  "created_at": "2023-11-20T14:30:00Z"
}
```

4.2.4 Интеграционные возможности

- Экспорт данных в 1С через `/reports?format=xml`
- Webhook-уведомления при изменении статуса задачи
- Поддержка авторизации через JWT (заголовок `Authorization: Bearer ...`)

Полная спецификация доступна в Swagger UI и приложенном YAML-файле.[]

4.3 Коды ответов (HTTP Status Codes)

Код	Описание
200	OK — запрос выполнен успешно
201	Created — создан новый ресурс

Код	Описание
400	Bad Request — ошибка клиента
401	Unauthorized — нет авторизации
404	Not Found — не найден ресурс
500	Internal Server Error — ошибка сервера

4.4 Обратная совместимость

Чтобы API оставался стабильным:

- Не удаляем или переименовываем существующие поля
- Все новые поля делаем **необязательными** (`nullable` , `optional`)
- Для кардинальных изменений создаём **новую версию API**, например: `/v2/tasks`

4.5 JSON Schema (валидация данных)

Для обеспечения целостности данных и защиты от некорректных запросов реализованы JSON-схемы. Они используются для валидации входных данных API и гарантируют:

- Проверку обязательных полей
- Корректность форматов (даты, email, URL)
- Валидацию статусов и типов через `enum`
- Защиту от SQL-инъекций и невалидных значений

4.5.1 Схема для оборудования (`Equipment`)

```
{
  "$schema": "http://json-schema.org/draft-07/schema#",
  "title": "Equipment",
  "description": "Данные оборудования",
  "type": "object",
  "required": ["name", "inventory_number"],
  "properties": {
    "name": {
```

```
      "type": "string",
      "minLength": 3,
      "maxLength": 255,
      "example": "Насос X200"
    },
    "inventory_number": {
      "type": "string",
      "pattern": "^INV-[A-Z0-9]{4}$",
      "example": "INV-045X"
    },
    "status": {
      "type": "string",
      "enum": ["active", "decommissioned", "storage"]
    }
  }
}
```

Поля:

- `name`: Название (3-255 символов)
- `inventory_number`: Инвентарный номер (формат `INV-XXXX`, где X — цифра/буква)
- `status`: Статус оборудования (только из списка)

4.5.2 Схема для задачи ТО (`MaintenanceTask`)

```
{
  "$schema": "http://json-schema.org/draft-07/schema#",
  "title": "MaintenanceTask",
  "description": "Задача технического обслуживания",
  "type": "object",
  "required": ["equipment_id", "planned_date"],
  "properties": {
    "equipment_id": {
      "type": "integer",
      "minimum": 1,
      "example": 15
    },
    "planned_date": {
      "type": "string",
      "format": "date",
      "example": "2023-12-01"
    },
```

```
    "status": {
      "type": "string",
      "enum": ["planned", "in_progress", "completed", "overdue"]
    }
  }
}
```

Правила:

- `equipment_id` — ID существующего оборудования
- `planned_date` — Дата в формате `YYYY-MM-DD`
- `status` — Только допустимые статусы

4.5.3 Схема для акта (`Act`)

```
{
  "$schema": "http://json-schema.org/draft-07/schema#",
  "title": "Act",
  "description": "Акт выполненных работ",
  "type": "object",
  "required": ["act_number", "file_url"],
  "properties": {
    "act_number": {
      "type": "string",
      "pattern": "^ACT-[0-9]{4}-[0-9]{2}$",
      "example": "ACT-2023-11"
    },
    "file_url": {
      "type": "string",
      "format": "uri",
      "pattern": "^https?://"
    }
  }
}
```

Поля:

- `act_number`: Номер акта (формат `АСТ-ГГГГ-ММ`)
- `file_url`: Ссылка на файл (только HTTP/HTTPS)

4.5.4 Интеграция с API

Схемы применяются на уровне обработки запросов:

```
// Пример валидации в Express.js
const validateSchema = (schema) => (req, res, next) => {
  const { error } = schema.validate(req.body);
  if (error) return res.status(400).json({ error: error.details[0].message });
  next();
};

// Создание задачи ТО
app.post('/tasks', validateSchema(taskSchema), (req, res) => {
  // Логика обработки
});
```

4.5.5 Пример ошибки валидации

```
{
  "error": "Validation failed",
  "details": [
    {
      "field": "inventory_number",
      "message": "must match pattern ^INV-[A-Z0-9]{4}$"
    }
  ]
}
```

4.5.6 Преимущества подхода

- **Стандартизация**: Единые правила для фронтенда и бэкенда.
- **Безопасность**: Защита от некорректных данных.
- **Документирование**: Схемы интегрированы с Swagger.
- **Тестирование**: Упрощение написания unit-тестов.

Этап 5. Интеграция с 1С/ERP

5.1. Формат данных

Для синхронизации данных с 1С/ERP используются:

- **XML** (основной формат для 1С).
- **JSON** (опционально для современных ERP).

Пример XML для экспорта задач ТО:

```xml
<MaintenanceTasks>
  <Task>
    <ExternalID>123</ExternalID>
    <Equipment>
      <InventoryNumber>INV-045X</InventoryNumber>
      <Name>Насос X200</Name>
    </Equipment>
    <Contractor>
      <ID>7</ID>
      <Name>ООО ТехСервис</Name>
    </Contractor>
    <PlannedDate>2023-12-01</PlannedDate>
    <Status>completed</Status>
    <Act>
      <Number>АСТ-2023-11</Number>
      <SignedAt>2023-12-05</SignedAt>
    </Act>
  </Task>
</MaintenanceTasks>
```

Пример JSON для REST-интеграции:

```json
{
  "tasks": [
    {
      "external_id": 123,
      "equipment": {
        "inventory_number": "INV-045X",
        "name": "Насос X200"
      },
      "planned_date": "2023-12-01",
      "status": "completed",
      "act": {
        "number": "АСТ-2023-11",
        "signed_at": "2023-12-05"
      }
```

```
    }
  ]
}
```

5.2. Механизм синхронизации

Сценарии:

1. **Периодическая выгрузка (раз в сутки):**
   - Система генерирует XML-файл с задачами, завершенными за последние 24 часа.
   - Файл отправляется через **SFTP** в папку для импорта 1С.
2. **Событийная синхронизация (вебхуки):**
   - При изменении статуса задачи система отправляет POST-запрос на URL 1С.
   - Пример вебхука:

```
{
  "event_type": "task.updated",
  "task_id": 123,
  "status": "completed",
  "timestamp": "2023-12-05T14:30:00Z"
}
```

3. **Ручной экспорт:**
   - Администратор может запустить выгрузку данных за произвольный период через интерфейс.

5.3. Обработка ошибок

---

1. **Повторные попытки:**
   - При ошибке HTTP ≥400 система повторяет отправку через 1, 5, 15 минут.
2. **Логирование:**
   - Все попытки фиксируются в таблице `sync_logs`:

```
CREATE TABLE sync_logs (
    id SERIAL PRIMARY KEY,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    status TEXT CHECK (status IN ('success', 'error')),
```

```
    error_message TEXT,
    payload TEXT
);
```

3. **Уведомления:**
   - При трех неудачных попытках администратор получает email.

5.4. XSD-схема для валидации XML

```
<!-- maintenance_task.xsd -->
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="MaintenanceTasks">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="Task" maxOccurs="unbounded">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="ExternalID" type="xs:int"/>
              <xs:element name="Equipment">
                <xs:complexType>
                  <xs:sequence>
                    <xs:element name="InventoryNumber" type="xs:string"/>
                    <xs:element name="Name" type="xs:string"/>
                  </xs:sequence>
                </xs:complexType>
              </xs:element>
              <xs:element name="PlannedDate" type="xs:date"/>
              <xs:element name="Status" type="xs:string"/>
              <xs:element name="Act" minOccurs="0">
                <xs:complexType>
                  <xs:sequence>
                    <xs:element name="Number" type="xs:string"/>
                    <xs:element name="SignedAt" type="xs:date"/>
                  </xs:sequence>
                </xs:complexType>
              </xs:element>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
```

```
  </xs:element>
</xs:schema>
```

5.5. Пример кода для экспорта данных

```
import requests
from lxml import etree

def export_to_1c(task_ids):
    tasks =
MaintenanceTask.query.filter(MaintenanceTask.id.in_(task_ids)).all()

    # Генерация XML
    root = etree.Element("MaintenanceTasks")
    for task in tasks:
        task_elem = etree.SubElement(root, "Task")
        etree.SubElement(task_elem, "ExternalID").text = str(task.id)
        equipment_elem = etree.SubElement(task_elem, "Equipment")
        etree.SubElement(equipment_elem, "InventoryNumber").text =
task.equipment.inventory_number
        etree.SubElement(equipment_elem, "Name").text = task.equipment.name
        etree.SubElement(task_elem, "PlannedDate").text =
task.planned_date.isoformat()
        etree.SubElement(task_elem, "Status").text = task.status

        if task.act:
            act_elem = etree.SubElement(task_elem, "Act")
            etree.SubElement(act_elem, "Number").text = task.act.act_number
            etree.SubElement(act_elem, "SignedAt").text =
task.act.signed_at.isoformat()

    # Валидация по XSD
    xmlschema = etree.XMLSchema(etree.parse("maintenance_task.xsd"))
    xmlschema.assertValid(root)

    # Отправка в 1С
    response = requests.post(
        "https://1c.company.com/api/maintenance",
        data=etree.tostring(root, encoding="utf-8"),
        headers={"Content-Type": "application/xml"}
    )
    response.raise_for_status()
```

5.6. Требования к 1С/ERP

---

1. **Авторизация:**
   - Использование API-ключа в заголовке `X-API-Key`.
2. **Идемпотентность:**
   - Повторная отправка одного и того же `ExternalID` не должна создавать дублирующих записей.
3. **Ответы:**
   - Успех: `200 OK` с телом `<Status>OK</Status>`.
   - Ошибка: `400 Bad Request` с описанием в `<Error>Invalid date format</Error>`.

Итоги интеграции

---

1. **Форматы:** XML (основной), JSON (опционально).
2. **Синхронизация:** Периодическая + событийная.
3. **Надежность:** Повторы, логирование, уведомления.
4. **Безопасность:** Валидация XSD, API-ключи.

Ключевые достижения:

1. **Анализ требований:**
   - Выявлены бизнес-цели: устранение ручного управления ТО, автоматизация уведомлений, интеграция с ERP.
   - Определены роли пользователей (администратор, подрядчик, сотрудник) и их потребности.
   - Сформулированы User Stories и функциональные/нефункциональные требования.
2. **Моделирование процессов:**
   - Разработаны диаграммы: BPMN (процесс ТО), Use Case (сценарии использования), Sequence (взаимодействие компонентов), Class (сущности), ER-диаграмма (база данных).
   - Визуализация упростила коммуникацию с заказчиком и командой разработки.
3. **Проектирование БД:**
   - Спроектирована структура на PostgreSQL с нормализацией, ограничениями (`CHECK`, `REFERENCES`) и индексами.

- Реализованы сущности: оборудование, задачи ТО, подрядчики, акты, уведомления.
- Документирование таблиц и связей для прозрачности структуры.

4. **REST API:**
   - Создана Swagger-документация с примерами запросов/ответов.
   - Реализованы методы для управления оборудованием, задачами, файлами и отчетами.
   - Валидация данных через JSON Schema для защиты от ошибок.
5. **Интеграция с 1С/ERP:**
   - Поддержка XML/JSON для экспорта данных.
   - Реализованы сценарии синхронизации: периодическая, событийная (вебхуки), ручная.
   - Обработка ошибок с повторными попытками и логированием.