**Техническая спецификация для проекта DoctorChecker**

**1. ВВЕДЕНИЕ**

Настоящая техническая спецификация разработана для создания веб-приложения DoctorChecker, предназначенного для управления документами Microsoft Word (.docx). Приложение позволяет загружать документы, хранить их метаданные в базе данных PostgreSQL, применять стили оформления в соответствии со стандартами ГОСТ и предоставлять возможность скачивания как оригинальных, так и отформатированных документов. Цель системы:  
· Обеспечить автоматизацию форматирования документов по ГОСТ.  
· Упростить управление документами через веб-интерфейс.  
· Демонстрировать навыки работы с Java и Spring Boot.

**2. ОСНОВАНИЯ ДЛЯ РАЗРАБОТКИ**

Разработка выполнена в рамках практического проекта для демонстрации навыков работы с Java, Spring Boot и PostgreSQL. Основания:  
· Требование реализации веб-приложения для управления документами.  
· Необходимость применения стилей ГОСТ к документам Microsoft Word.  
· Использование Java как основного языка программирования.

**3. НАЗНАЧЕНИЕ РАЗРАБОТКИ**

Приложение DoctorChecker предназначено для:  
· Загрузки, хранения и скачивания документов Microsoft Word (.docx).  
· Применения стилей оформления в соответствии со стандартами ГОСТ.  
· Хранения метаданных документов в базе данных PostgreSQL.  
· Демонстрации навыков разработки на Java (Spring Boot, JPA/Hibernate, Apache POI).

**4. ТРЕБОВАНИЯ К ВЕБ-ПРИЛОЖЕНИЮ**

**4.1 Требования к функциональным характеристикам**

· Управление документами:  
· Загрузка документов через веб-интерфейс (HTML-форма).  
· Отображение списка загруженных документов (имя, дата загрузки).  
· Скачивание оригинальных документов.  
· Применение стилей ГОСТ и скачивание отформатированных документов.  
· Обработка документов:  
· Форматирование документов по ГОСТ (шрифт Times New Roman, размер 12, межстрочный интервал 1.5, абзацный отступ 1.25 см, поля: левое — 30 мм, правое — 10 мм, верхнее — 20 мм, нижнее — 25 мм).  
· Хранение данных:  
· Сохранение метаданных документов (имя, дата загрузки) в PostgreSQL.  
· Хранение файлов в локальной директории (./uploads).  
· Веб-интерфейс:  
· Простая HTML-страница для загрузки и просмотра документов.

**4.2 Требования к надежности**

· Безопасность данных:  
· Хранение файлов в защищённой директории.  
· Логирование операций с документами.  
· Обработка ошибок:  
· Уведомление пользователя о неподдерживаемых форматах файлов.  
· Проверка на наличие загружаемого файла.

**4.3 Условия эксплуатации**

· Клиентская часть:  
· Поддержка браузеров: Chrome, Firefox, Safari, Edge.  
· Минималистичный веб-интерфейс (HTML с Thymeleaf).  
· Серверная часть:  
· Совместимость с Linux/Windows Server.  
· Локальный запуск или деплой в облаке (опционально).

**4.4 Требования к техническим средствам**

· Клиент:  
· Процессор: Intel Core i3 и выше.  
· ОЗУ: 2 ГБ и более.  
· Подключение к интернету (минимум 5 Мбит/с).  
· Сервер:  
· Минимум 4 ГБ ОЗУ, 1 ядро CPU.  
· SSD-накопители для хранения файлов и базы данных.

**4.5 Совместимость**

· Технологии:  
· Frontend: HTML, Thymeleaf.  
· Backend: Spring Boot (Spring Web, Spring Data JPA).  
· Базы данных: PostgreSQL.  
· Форматы файлов:  
· Поддержка Microsoft Word (.docx) через Apache POI.

**4.6 Специальные требования**

· Логирование: Логирование операций загрузки и обработки документов (SLF4J).  
· Форматирование: Соответствие ГОСТ 7.32-2017 (например, размеры полей, шрифт, межстрочный интервал).

**5. ЭТАПЫ РАЗРАБОТКИ**

 **Сбор и анализ требований** — 23 мая  
· Определение основных функций: загрузка, форматирование по ГОСТ, скачивание документов.

 **Проектирование архитектуры приложения** — 23 мая  
· Создание структуры проекта и схемы базы данных PostgreSQL.

 **Разработка backend (контроллеры, сервисы, репозитории)** — 24 мая  
· Настройка Spring Boot и подключение PostgreSQL.

 **Реализация форматирования документов по ГОСТ** — 24 мая  
· Использование Apache POI для работы с .docx и применение стилей ГОСТ.

 **Разработка веб-интерфейса** — 25 мая  
· Создание HTML-шаблона с Thymeleaf.

 **Тестирование и отладка** — 25 мая  
· Проверка загрузки, форматирования и скачивания документов.

**6. ТРЕБОВАНИЯ К ПРОГРАММНОЙ ДОКУМЕНТАЦИИ**

· Комментарии в коде: Использование JavaDoc для ключевых классов и методов.  
· Руководства:  
· Руководство пользователя (инструкция по загрузке и скачиванию документов).  
· Руководство администратора (настройка приложения и базы данных).  
· Описание проекта: README с инструкцией по запуску.

**7. ПРИЕМКА РАБОТ**

· Финальное тестирование функциональности загрузки, форматирования и скачивания документов.  
· Подписание акта приемки после демонстрации работы приложения.

**8. ПРИМЕЧАНИЕ**

Все изменения требований оформляются в виде дополнительных соглашений.

Пример структуры репозитория:

├── src/ # Исходный код приложения

│ ├── main/ # Основной код

│ │ ├── java/ # Java-пакеты

│ │ │ └── com/example/doctorchecker/

│ │ │ ├── controller/ # Контроллеры

│ │ │ │ └── DocumentController.java

│ │ │ ├── dto/ # DTO-классы

│ │ │ │ └── DocumentDTO.java

│ │ │ ├── model/ # Модель данных

│ │ │ │ └── Document.java

│ │ │ ├── repository/ # Репозитории

│ │ │ │ └── DocumentRepository.java

│ │ │ ├── service/ # Сервисы

│ │ │ │ └── DocumentService.java

│ │ │ ├── util/ # Утилиты

│ │ │ │ └── StyleApplier.java

│ │ │ └── DoctorCheckerApplication.java # Точка входа

│ │ └── resources/ # Ресурсы приложения

│ │ ├── static/ # Статические файлы (CSS, JS)

│ │ ├── templates/ # HTML-шаблоны

│ │ │ └── document-list.html

│ │ └── application.properties # Конфигурация

└── pom.xml # Файл сборки Maven

**Технологии**

· Frontend: HTML, Thymeleaf.  
· Backend: Spring Boot (Spring Web, Spring Data JPA).  
· Базы данных: PostgreSQL.  
· Обработка документов: Apache POI.

**Ключевые преимущества**

· Автоматизация: Применение стилей ГОСТ к документам.  
· Простота: Удобный веб-интерфейс для управления документами.  
· Соответствие стандартам: Форматирование документов выполнено с учётом ГОСТ 7.32-2017.