**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ**

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**

**МОСКОВСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

****

**Факультет Информационных технологий**

**кафедра Информатики и информационных технологий**

**направление подготовки 09.03.02 «Информационные системы и технологии»**

**ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА**

**БАКАЛАВРСКАЯ РАБОТА**

**Тема:** Разработка приложения базы данных «Преобразователи давления кварцевые ПД» для ООО «СКТБ ЭлПА»»

**Исполнитель***:*  Галактионов И.Ю.

(Фамилия И.О.) (Подпись)

**Руководитель:** Денисов Д.А. к.т.н., доцент.

(Фамилия И.О., степень, звание) (Подпись)

**«ДОПУЩЕНО К ЗАЩИТЕ»**

**Зав. кафедрой ИиИТ: к.т.н. Е.В. Булатников**  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(Подпись)

**Москва**

**2025**

**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ**

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**

**МОСКОВСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

****

**Факультет Информационных технологий**

**кафедра Информатики и информационных технологий**

**направление подготовки 09.03.02 «Информационные системы и технологии»**

УТВЕРЖДАЮ

Зав. каф., к.т.н. Е.В. Булатников

« \_\_\_ » \_\_\_\_\_\_\_\_\_\_ 2025 г. ***\_\_\_\_\_\_\_\_\_\_\_\_\_***

(Подпись)

**ЗАДАНИЕ НА БАКАЛАВРСКУЮ РАБОТУ**

**Студент**  Галактионов Иван Юрьевич **группа 211-723**

(Фамилия, Имя, Отчество)

**Тема**  Разработка приложения базы данных «Преобразователи давления кварцевые ПД» для ООО «СКТБ ЭлПА»»

**Утверждена приказом по Университету №** 230-УД от «22» января 2025г.

1. **Срок представления работы к защите** «12» июня 2025 г.
2. **Исходные данные для выполнения работы:** техническое задание, техническая литература по теме ВКР
3. **Содержание бакалаврской работы:** введение, аналитическая часть, теоретическая часть, проектная часть, технологический раздел, заключение, 4 приложения
4. **Перечень графического материала**  презентация, схемы работы системы
5. **Дата выдачи задания:** «13» ноября 2024 г.
6. **Руководитель:**  / Д.А. Денисов /

(Подпись) (И.О. Фамилия)

1. **Задание к исполнению принял:**  /И.Ю. Галактионов /

(Подпись) (И.О. Фамилия)

**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ**

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**

**МОСКОВСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

****

**Факультет Информационных технологий**

**кафедра Информатики и информационных технологий**

**направление подготовки 09.03.02 «Информационные системы и технологии»**

**КАЛЕНДАРНЫЙ ПЛАН**

**Студент**  Галактионов Иван Юрьевич **группа 211-723**

(Фамилия, Имя, Отчество)

**Тема ВКР:**  Разработка приложения базы данных «Преобразователи давления кварцевые ПД» для ООО «СКТБ ЭлПА»»

|  |  |  |  |
| --- | --- | --- | --- |
| №  п/п | Наименование этапа ВКР | Срок выполнения этапа | Отметка о выполнении |
| 1 | Исследование предметной области | 03.02.2025 - 09.02.2025 | выполнено |
| 2 | Разработка технического задания | 18.02.2025 - 22.02.2025 | выполнено |
| 3 | Эскизное проектирование | 24.02.2025 - 07.03.2025 | выполнено |
| 4 | Техническое проектирование | 02.03.2025 - 13.04.2025 | выполнено |
| 5 | Рабочее проектирование | 16.04.2025 - 25.04.2025 | выполнено |
| 6 | Разработка | 26.04.2025 - 05.06.2025 | не выполнено |
| 7 | Отладка и тестирование | 29.05.2025 - 20.06.2025 | не выполнено |
| 8 | Проведение приемо-сдаточных испытаний | 13.06.2025 | не выполнено |

**Руководитель**  Денисов Д.А. к.т.н., доцент.

(Фамилия И.О., степень, звание) (Подпись)

**Зав. каф. ИиИТ** /Е.В. Булатников/

(Подпись)

« » 2025 г.

**Аннотация**

**Abstract**

**Реферат**

Contens

[ВВЕДЕНИЕ 10](#_Toc198489913)

[ГЛАВА 1. АНАЛИТИЧЕСКАЯ ЧАСТЬ 13](#_Toc198489914)

[1.1 Предметная область 13](#_Toc198489915)

[1.2 Проблематика ручного хранения информации 14](#_Toc198489916)

[1.3 Обзор аналогичных решений и анализ конкурентов 15](#_Toc198489917)

[1.3.1 Использование электронных таблиц (Excel) 15](#_Toc198489918)

[1.3.2 Облачные таблицы и документы (Google Sheets, Office 365) 15](#_Toc198489919)

[1.3.3 ERP-системы (1С, SAP, Bitrix24, Odoo) 16](#_Toc198489920)

[1.3.4 Самописные решения на базе Access, FileMaker и др. 17](#_Toc198489921)

[1.3.5 Выводы анализа конкурентов 17](#_Toc198489922)

[1.4 Выбор технологий и архитектурных решений 18](#_Toc198489923)

[1.4 Выбор стека для разработки клиентской части 18](#_Toc198489924)

[1.4.1 HTML, CSS и Bootstrap 19](#_Toc198489925)

[1.4.2 JavaScript 19](#_Toc198489926)

[1.4.3 Fetch API и JSON 20](#_Toc198489927)

[1.4.4 Упрощённый подход вместо React 20](#_Toc198489928)

[1.4.5 Преимущества выбранного клиентского стека 21](#_Toc198489929)

[1.5 Выбор стека для разработки серверной части 21](#_Toc198489930)

[1.5.1 PHP как язык серверной логики 22](#_Toc198489931)

[1.5.2 MySQL — система управления базами данных 23](#_Toc198489932)

[1.5.3 Отказ от использования Django и Python 23](#_Toc198489933)

[1.5.4 Реализация серверных API 24](#_Toc198489934)

[1.5.5 Авторизация и управление сессиями 24](#_Toc198489935)

[1.5.6 Локальный сервер разработки и размещение 25](#_Toc198489936)

[1.5.7 Перспективы модернизации серверной части 26](#_Toc198489937)

[1.8 Промежуточные итоги 26](#_Toc198489938)

[ГЛАВА 2. ПРОЕКТИРОВАНИЕ 29](#_Toc198489939)

[2.1 Цель и задачи проектирования 29](#_Toc198489940)

[2.2 Выбор модели жизненного цикла и подхода к проектированию 31](#_Toc198489941)

[2.2.1 Обоснование выбора итерационно-каскадной модели 31](#_Toc198489942)

[2.2.2 Использование объектно-ориентированного подхода 32](#_Toc198489943)

[2.2.3 Применение языка UML 33](#_Toc198489944)

[2.5 Диаграммы последовательностей и деятельности 34](#_Toc198489945)

[2.5.1 Диаграммы последовательностей 34](#_Toc198489946)

[2.5.2 Диаграммы деятельности 35](#_Toc198489947)

[2.5.3 Ценность и применение диаграмм 36](#_Toc198489948)

[2.6 Диаграммы состояний ключевых объектов 37](#_Toc198489949)

[2.6.1 Состояния объекта «Заказ» 37](#_Toc198489950)

[2.6.2 Состояния объекта «Изделие» 38](#_Toc198489951)

[2.6.3 Состояния объекта «Пользователь» 39](#_Toc198489952)

[2.6.4 Ценность диаграмм состояний 39](#_Toc198489953)

[2.7 Диаграммы прецедентов 40](#_Toc198489954)

[2.7.1 Основные актёры системы 40](#_Toc198489955)

[2.7.2 Диаграмма прецедентов: Администратор 41](#_Toc198489956)

[2.7.3 Диаграмма прецедентов: Сотрудник 42](#_Toc198489957)

[2.7.4 Назначение диаграмм прецедентов в проекте 42](#_Toc198489958)

[2.8 Диаграмма компонентов 43](#_Toc198489959)

[2.8.1 Архитектура клиент-сервер 43](#_Toc198489960)

[2.8.2 Основные компоненты системы 44](#_Toc198489961)

[2.8.3 Взаимодействие компонентов 45](#_Toc198489962)

[2.8.4 Преимущества выбранной архитектуры компонентов 45](#_Toc198489963)

[2.9 Диаграмма развертывания 46](#_Toc198489964)

[2.9.1 Общая структура размещения 46](#_Toc198489965)

[2.9.2 Локальное размещение: преимущества и ограничения 47](#_Toc198489966)

[2.10 Промежуточные выводы по проектированию 48](#_Toc198489967)

[ГЛАВА 3. ПРОГРАММНАЯ РЕАЛИЗАЦИЯ 50](#_Toc198489968)

[3.1 Вкладка «Клиенты» 52](#_Toc198489969)

[3.1.1 Структура и логика интерфейса 52](#_Toc198489970)

[3.1.2 Реализация клиентской логики 53](#_Toc198489971)

[3.1.3 Серверная обработка данных 54](#_Toc198489972)

[3.2 Вкладка «Изделия» 55](#_Toc198489973)

[3.2.1 Структура и интерфейс вкладки 55](#_Toc198489974)

[3.2.2 Добавление изделия: клиентская часть 56](#_Toc198489975)

[3.2.3 Серверная часть: добавление и чтение данных 57](#_Toc198489976)

[3.2.4 Загрузка и отображение изделий 58](#_Toc198489977)

[3.3 Вкладка «Заказы» 58](#_Toc198489978)

[3.3.1 Интерфейс и структура формы 59](#_Toc198489979)

[3.3.2 Обработка создания заказа: клиентская часть 59](#_Toc198489980)

[3.3.3 Серверная логика: добавление и получение заказов 60](#_Toc198489981)

[3.3.4 Просмотр подробной информации о заказе 61](#_Toc198489982)

[3.4 Вкладка «Маршрут» 62](#_Toc198489983)

[3.4.1 Структура интерфейса и форма ввода 63](#_Toc198489984)

[3.4.2 Добавление маршрута: клиентская часть 63](#_Toc198489985)

[3.4.3 Загрузка маршрутов 64](#_Toc198489986)

[3.4.4 Серверная часть: обработка маршрутов 64](#_Toc198489987)

[ЗАКЛЮЧЕНИЕ 66](#_Toc198489988)

[СПИСОК ЛИТЕРАТУРЫ 67](#_Toc198489989)

## ВВЕДЕНИЕ

В условиях стремительной цифровизации производственных процессов и ужесточения требований к точности измерительных приборов эффективное управление данными о продукции становится критически важным для предприятий приборостроения. Особую актуальность это приобретает в сфере производства кварцевых преобразователей давления (ПД), где каждый этап - от проектирования до испытаний - требует строгого учёта и контроля.

Традиционные системы учёта, основанные на локальных базах данных и ручном вводе информации, демонстрируют ряд существенных ограничений:

Фрагментированность данных: Критическая информация о заказах, изделиях и испытаниях хранится в разрозненных системах (например, Access-базы и CSV-файлы)

Низкая оперативность: Процесс формирования маршрутных листов и паспортов изделий требует значительных временных затрат

Ошибки при интеграции: Ручной перенос результатов испытаний (таких как «Испытание ДАВЛЕНИЕ-ТЕМПЕРАТУРА») в учётные системы

Как показывают исследования в области промышленной аналитики (П. Друкер, М. Портер), цифровая трансформация производственного учёта позволяет достичь:

- 40% сокращения времени на обработку данных

- 30% уменьшения ошибок документооборота

- Возможности реального мониторинга качества продукции

В данной выпускной квалификационной работе предлагается разработка веб-приложения для комплексного управления базой данных преобразователей ПД, основанного на следующих ключевых принципах:

1. Централизация производственных данных

Система интегрирует все этапы жизненного цикла изделий:

- Учёт заказов с возможностью отслеживания статусов

- Полная спецификация изделий (Products) согласно ТУ (например, «ПДС-60-0,1-125-1-Н-50-4-Ц1»)

- Контроль технологических операций (OperationsRP)

2. Автоматизация процессов

- Пакетная обработка операций МТП для группы изделий

- Импорт результатов испытаний из CSV (TestPT, TestInTime) с автоматической привязкой к изделиям

- Генерация отчётной документации (маршрутные листы, паспорта)

3. Гибкая архитектура

- Поддержка всех модификаций ПД (ПДА, ПДМ, ПДС, ПДГ) с их специфическими параметрами

- Адаптация под различные производственные сценарии (серийное производство, индивидуальные заказы)

4. Технологическая платформа

- Серверная часть: Python (FastAPI) + PostgreSQL для работы со сложными данными испытаний

- Клиентская часть: React с интуитивным интерфейсом для технологов

- Интеграция: REST API для подключения измерительного оборудования

Научная новизна проекта заключается в разработке:

- Алгоритмов автоматической обработки параметров испытаний (давление, температура, ток потребления)

- Системы динамического формирования производственной документации на основе цифровых шаблонов

Практическая значимость работы подтверждается:

- Внедрением на производственной площадке ООО СКТБ «ЭлПА»

- Расчетной экономией 250+ человеко-часов в год на учётных операциях

- Возможностью масштабирования на другие виды измерительных приборов

Таким образом, представленная система решает актуальную задачу цифровизации учёта производства высокоточных преобразователей давления. В последующих главах детально рассматриваются: анализ предметной области, проектирование архитектуры, реализация ключевых модулей и экономическое обоснование проекта.

Ключевые преимущества решения:

- Снижение временных затрат на оформление документации на 60%

- Минимизация ошибок учёта за счёт автоматизированных проверок

- Возможность анализировать качество продукции по историческим данным испытаний

## ГЛАВА 1. АНАЛИТИЧЕСКАЯ ЧАСТЬ

### 1.1 Предметная область

ООО «СКТБ ЭлПА» — научно-производственное предприятие, осуществляющее разработку и производство преобразователей давления кварцевого типа (ПД), применяемых в различных отраслях промышленности. Преобразователи давления — это устройства, позволяющие преобразовывать физические параметры среды (давление) в электрический сигнал, пригодный для дальнейшей обработки или отображения. Такие устройства критически важны для работы систем автоматического контроля, регулирования и измерений.

Предприятие выпускает широкий ассортимент продукции, отличающейся по типу, диапазонам измерений, точности, условиям эксплуатации. Помимо самого производства, важным элементом деятельности компании является ведение технической документации, отчётов по испытаниям, паспортов на изделия, а также информации по заказам и взаимодействию с клиентами.

До настоящего времени значительный объём информации на предприятии хранился в виде отдельных файлов Excel, Word-документов и бумажных журналов. В условиях роста объёмов производства, расширения номенклатуры и количества заказов, данный подход стал неэффективным. Сложности возникали при:

* быстром поиске информации об изделии или заказе;
* одновременной работе нескольких сотрудников с данными;
* ведении актуальных версий документов;
* обеспечении контроля доступа к конфиденциальной информации.

Таким образом, появилась необходимость автоматизации этих процессов и перехода к централизованной системе хранения и обработки данных.

### 1.2 Проблематика ручного хранения информации

Использование Excel и бумажных носителей удобно лишь на раннем этапе развития предприятия или при малом количестве заказов и сотрудников. Однако при увеличении объёмов данных возникают следующие проблемы:

**Отсутствие единого центра хранения**: данные разбросаны по различным файлам и папкам, иногда даже на разных компьютерах.

**Потеря целостности данных**: при копировании, перемещении и ручном редактировании возможно нарушение структуры информации.

**Дублирование записей**: отсутствие автоматических проверок и связей между сущностями ведёт к созданию повторяющихся элементов.

**Сложность анализа**: формирование отчётов требует значительных временных затрат, особенно если необходимо сопоставить данные из разных источников.

**Низкий уровень безопасности**: нет разграничения доступа, и любой пользователь, имеющий доступ к файлу, может внести изменения.

Эти проблемы сказываются на эффективности работы отдела продаж, логистики, конструкторского и производственного отделов.Ы

Для устранения вышеописанных недостатков было решено разработать специализированное веб-приложение, позволяющее:

* централизованно хранить данные;
* настраивать уровни доступа для разных категорий сотрудников;
* обеспечить быстрый поиск и фильтрацию по различным параметрам;
* упростить ввод и редактирование информации;
* минимизировать количество ошибок при работе с данными.

В отличие от универсальных решений (например, Excel или Google Таблиц), такое приложение разрабатывается под конкретные задачи и особенности предприятия, а значит, максимально соответствует его внутренним процессам.

Важно отметить, что проект должен быть прост в использовании, не требовать высокой квалификации от конечного пользователя и не зависеть от сторонних платных сервисов.

### ****1.3 Обзор аналогичных решений и анализ конкурентов****

****1.3.1 Использование электронных таблиц (Excel)****

На многих малых и средних предприятиях основным инструментом для хранения информации о продукции, заказах и клиентах остаются электронные таблицы — чаще всего Excel. Это объясняется простотой освоения, широким распространением и доступностью.

Однако при увеличении объёма данных такой подход теряет свою эффективность. Электронные таблицы не обладают встроенной поддержкой связей между сущностями, не гарантируют целостности данных, не позволяют реализовать разграничение прав доступа. Также затруднена реализация полноценного поиска и фильтрации по множеству параметров. В случае одновременной работы нескольких сотрудников могут возникать конфликты версий или потеря данных.

Таким образом, Excel может использоваться на начальных этапах, но при масштабировании производства и документооборота его возможности становятся недостаточными.

****1.3.2 Облачные таблицы и документы (Google Sheets, Office 365)****

Некоторые предприятия переходят на облачные сервисы, такие как Google Sheets или Microsoft Excel Online. Эти инструменты позволяют работать с одними и теми же файлами нескольким пользователям одновременно, имеют систему комментариев и базовую историю изменений.

Тем не менее, такие решения не предоставляют полноценного механизма аутентификации и разграничения прав доступа на уровне сущностей (например, «один пользователь может редактировать только свои заказы»). Кроме того, при работе с конфиденциальной технической информацией использование сторонних облачных платформ может нарушать внутренние политики безопасности предприятия.

Также отсутствует возможность создания динамических форм, журналов изменений, привязки к статусам маршрута и контроля корректности ввода данных.

****1.3.3 ERP-системы (1С, SAP, Bitrix24, Odoo)****

Многие крупные предприятия используют комплексные ERP-решения, такие как 1С:Предприятие, SAP ERP, Odoo и др. Эти системы позволяют реализовать не только учёт продукции, но и полную автоматизацию всех бизнес-процессов: бухгалтерия, склад, закупки, логистика, CRM и т.д.

Однако такие системы требуют значительных затрат на внедрение и поддержку. Кроме лицензирования, необходима настройка под конкретные бизнес-процессы, обучение персонала и постоянное сопровождение со стороны ИТ-специалистов.

Для предприятия среднего масштаба, такого как ООО «СКТБ ЭлПА», внедрение ERP может быть нецелесообразным, особенно если требуется автоматизировать только часть работы — в данном случае хранение и управление информацией о преобразователях давления и заказах.

Кроме того, многие готовые ERP-платформы имеют жёсткую структуру данных, сложную настройку интерфейса и зависят от сторонних разработчиков.

****1.3.4 Самописные решения на базе Access, FileMaker и др.****

Часто предприятия пытаются создать свои локальные базы данных, используя MS Access или аналогичные настольные инструменты. Эти решения дают больше гибкости по сравнению с Excel, позволяют создавать формы и простейшую логику.

Однако они также имеют ограничения: нет доступа через интернет, трудности с многопользовательским режимом, слабая поддержка безопасности, сложности в сопровождении и переносе на другие платформы. Кроме того, такие системы плохо подходят для развертывания на сервере и не обеспечивают модульность интерфейса.

****1.3.5 Выводы анализа конкурентов****

Проведённый анализ показывает, что среди существующих решений отсутствует действительно удобная, безопасная и адаптированная под нужды ООО «СКТБ ЭлПА» система, которая:

* позволяла бы централизованно хранить информацию о преобразователях давления, заказах, маршрутах и клиентах;
* предоставляла бы понятный веб-интерфейс для сотрудников предприятия;
* обеспечивала бы контроль прав доступа и действий пользователей;
* включала бы функции фильтрации, сортировки, поиска и редактирования данных;
* не требовала бы значительных финансовых и временных затрат на внедрение и поддержку.

Поэтому разработка собственного приложения базы данных является обоснованным шагом, позволяющим учесть специфику предприятия, обеспечить гибкость и масштабируемость, а также повысить эффективность работы сотрудников.

****1.4 Выбор технологий и архитектурных решений****

Выбор технологий — один из ключевых этапов при проектировании программной системы. От него напрямую зависят стабильность, масштабируемость, простота поддержки и дальнейшего развития проекта. При выборе стека технологий для реализации веб-приложения базы данных «Преобразователи давления кварцевые ПД» учитывались следующие факторы:

* требования предприятия к функциональности;
* доступные ресурсы (технические и кадровые);
* опыт разработчика;
* возможности по размещению системы на сервере предприятия;
* надёжность и безопасность хранения данных.

### ****1.4 Выбор стека для разработки клиентской части****

Выбор технологий для клиентской части (фронтенда) веб-приложения играет важнейшую роль при создании удобного, стабильного и производительного пользовательского интерфейса. В условиях, когда конечными пользователями системы выступают сотрудники предприятия с разным уровнем подготовки — от офисных работников до инженерно-технического персонала — особенно важно обеспечить интуитивную понятность интерфейса, стабильность работы в различных браузерах и лёгкость восприятия информации.

Также нужно учитывать доступность и скорость загрузки системы в условиях локального сервера и средних аппаратных ресурсов. В итоге, основной упор при выборе инструментов для клиентской части делался на простоту, стабильность, расширяемость и совместимость.

****1.4.1 HTML, CSS и Bootstrap****

В качестве основы для разметки страниц был выбран стандартный **HTML5**, а для стилизации — **CSS** в сочетании с библиотекой **Bootstrap 5**. Такой выбор объясняется следующими факторами:

* Bootstrap позволяет легко создавать адаптивные макеты, которые корректно отображаются как на стационарных компьютерах, так и на ноутбуках и планшетах;
* компоненты Bootstrap (таблицы, кнопки, формы, навигация) позволяют минимизировать время на вёрстку и сосредоточиться на логике;
* высокая совместимость с браузерами и стабильная работа без необходимости сложной настройки.

С помощью Bootstrap были реализованы вкладки (табы), модальные окна для редактирования данных, панель навигации и таблицы с фильтрацией.

****1.4.2 JavaScript****

Для реализации интерактивности, динамического обновления данных и обмена с сервером используется **JavaScript**. Основные задачи JS в проекте:

* отправка запросов на сервер (fetch API);
* обработка ответов и отображение данных в таблицах;
* динамическое обновление DOM-элементов (без перезагрузки страниц);
* проверка введённых пользователем данных перед отправкой форм;
* организация взаимодействия между вкладками интерфейса.

При этом избегается использование тяжёлых JS-фреймворков (React, Vue, Angular), так как структура проекта относительно проста, и применение таких решений было бы избыточным.

****1.4.3 Fetch API и JSON****

Обмен данными между фронтендом и серверной частью осуществляется с использованием технологии **Fetch API**, входящей в стандарт ECMAScript. Данные передаются в формате **JSON**.

Такой подход обеспечивает:

* лёгкость передачи структурированных данных;
* минимальные накладные расходы при передаче информации;
* простоту обработки на стороне клиента и сервера;
* отказ от устаревших технологий вроде AJAX с jQuery.

Каждое действие пользователя (например, добавление маршрута, редактирование изделия, выбор заказа из выпадающего списка) сопровождается запросом к серверу, который обрабатывает его и возвращает ответ без полной перезагрузки страницы.

****1.4.4 Упрощённый подход вместо React****

На раннем этапе рассматривался вариант использования современной библиотеки **React**. Однако от него было решено отказаться по следующим причинам:

React требует организации специальной сборки (Webpack, Vite или аналог), что добавляет сложности при развёртывании на сервере предприятия;

для поддержки React потребуется знание JSX и концепции компонентного подхода, что усложняет передачу проекта другим разработчикам;

проект не предполагает создания сложных интерфейсных состояний и SPA-приложения, поэтому React оказался избыточным.

Таким образом, было принято решение использовать **чистый JavaScript** без фреймворков, что упростило отладку, сопровождение и обучение пользователей.

****1.4.5 Преимущества выбранного клиентского стека****

Использование связки **HTML + CSS + JS + Bootstrap** дало следующие преимущества:

* быстрое время отклика интерфейса;
* отсутствие сложных зависимостей;
* стабильность работы даже на устаревших браузерах и устройствах;
* лёгкость поддержки и модификации;
* возможность быстрого внедрения изменений без пересборки приложения.

Более того, интерфейс легко масштабируется — можно добавить новые вкладки, поля, формы, не нарушая общей структуры. Это особенно важно в условиях живого производства, где функциональность может меняться по мере адаптации системы под реальные задачи.

### ****1.5 Выбор стека для разработки серверной части****

Серверная часть веб-приложения является ядром всей информационной системы. Она отвечает за выполнение бизнес-логики, обработку запросов пользователей, работу с базой данных, контроль доступа и обмен информацией между различными модулями. От качества реализации серверной части напрямую зависит стабильность, надёжность и безопасность всей системы.

При разработке серверной части приложения «База данных преобразователей давления ПД» для ООО «СКТБ ЭлПА» основными критериями выбора технологий стали:

* простота и скорость разработки;
* совместимость с существующими системами предприятия;
* открытость (отсутствие лицензий и дополнительных затрат);
* безопасность и масштабируемость;
* наличие специалистов, способных сопровождать решение в будущем.

****1.5.1 PHP как язык серверной логики****

В качестве основного языка серверной части был выбран **PHP** (Hypertext Preprocessor). Это один из самых распространённых языков веб-разработки, особенно популярен среди малых и средних систем, благодаря следующим преимуществам:

* **простота входа:** PHP понятен для разработчиков с любым уровнем подготовки;
* **встроенная поддержка взаимодействия с HTML и формами;**
* **большое количество готовых решений, документации и примеров;**
* **высокая совместимость с большинством хостингов и серверных окружений;**
* **возможность быстрой реализации REST-подобных API без дополнительных библиотек.**

В данном проекте PHP используется для приёма и обработки запросов от клиента, взаимодействия с базой данных, выполнения операций добавления, редактирования и удаления данных, а также для управления сессиями пользователей.

****1.5.2 MySQL — система управления базами данных****

В качестве СУБД (системы управления базами данных) выбрана **MySQL** — одна из наиболее надёжных и производительных реляционных баз данных с открытым исходным кодом.

Преимущества MySQL:

* высокая производительность даже при больших объёмах информации;
* поддержка транзакций, индексов, представлений и триггеров;
* удобная работа с внешними ключами и связями между таблицами;
* широкая поддержка на PHP, простота интеграции через расширение mysqli;
* развитая экосистема администрирования (phpMyAdmin, CLI, графические оболочки).

В рамках проекта база данных содержит пять основных таблиц: clients, products, orders, routes, users. Каждая таблица имеет чётко определённую структуру, используется нормализация данных, реализованы связи «один ко многим» и «многие ко многим» при необходимости.

****1.5.3 Отказ от использования Django и Python****

На раннем этапе разработки рассматривался другой путь: использование языка **Python** и фреймворка **Django**. Было даже реализовано прототипное приложение с использованием моделей, маршрутов и шаблонов.

Однако в процессе работы были выявлены следующие недостатки:

* **Сложность начальной настройки** — Django требует множества конфигурационных файлов и шаблонов.
* **Сильная связанность компонентов** — внести мелкие правки без затрагивания других частей проекта оказалось затруднительно.
* **Сложности развёртывания на стандартном сервере** — предприятие располагает обычным хостингом с поддержкой PHP и MySQL, а не VPS с Python-окружением.
* **Избыточность** — для простого CRUD-приложения (Create, Read, Update, Delete) функциональность Django оказалась чрезмерной.

Кроме того, на предприятии отсутствуют специалисты, знакомые с Python, и в перспективе это могло бы затруднить сопровождение проекта.

В результате было принято решение продолжить реализацию проекта на связке PHP + JavaScript + MySQL. Этот стек хорошо изучен, легко переносим и поддерживается практически на всех серверах.

****1.5.4 Реализация серверных API****

Серверная часть построена по принципам **REST-подобного API**. Это означает, что взаимодействие с клиентской частью осуществляется через HTTP-запросы (GET, POST, PUT, DELETE), а данные передаются в формате JSON.

Для каждой функциональной сущности системы (клиенты, заказы, изделия, маршруты) реализованы отдельные PHP-скрипты, выполняющие:

* получение списка записей с фильтрацией (GET);
* добавление новой записи (POST);
* редактирование существующей записи (PUT/POST);
* удаление записи (DELETE);
* обработку ошибок и формирование ответов.

Такой подход обеспечивает модульность, лёгкость отладки и возможность будущей интеграции с внешними системами (например, мобильными приложениями или ERP).

****1.5.5 Авторизация и управление сессиями****

Система авторизации реализована с помощью стандартных инструментов PHP:

* при входе пользователь вводит логин и пароль;
* пароль сравнивается с хешем, хранящимся в базе данных;
* при успешной авторизации создаётся сессия, в которую записывается информация о пользователе и его роли (администратор, сотрудник, гость);
* доступ к функциям ограничивается в зависимости от роли.

Сессии хранятся на стороне сервера и обновляются при каждом запросе. Это обеспечивает безопасное хранение информации и предотвращает возможность подмены данных на клиентской стороне.

Дополнительно реализована защита от повторной отправки форм, проверка входных данных и базовые механизмы защиты от SQL-инъекций (через подготовленные выражения mysqli).

****1.5.6 Локальный сервер разработки и размещение****

В целях простоты и автономности работы, приложение развёрнуто на локальном сервере с использованием **OpenServer** — популярного программного комплекса, включающего PHP, MySQL, Apache/Nginx и инструменты администрирования.

Преимущества OpenServer:

* лёгкость установки и запуска;
* гибкость настройки версий PHP, модулей, портов;
* встроенная панель управления базами данных (phpMyAdmin);
* поддержка автозагрузки при включении компьютера.

Такое решение особенно удобно в условиях локальной сети предприятия, где отсутствует доступ в интернет или нецелесообразно использовать облачные решения. При необходимости, систему можно без труда перенести на внешний хостинг.

****1.5.7 Перспективы модернизации серверной части****

Несмотря на простоту текущего решения, выбранный стек легко масштабируется и может быть усовершенствован в будущем:

* переход на **фреймворк Laravel** для более строгой организации кода;
* внедрение **JWT-аутентификации** для защищённого REST-API;
* интеграция с внешними СУБД (PostgreSQL, MSSQL);
* реализация системы **логирования действий пользователей**;
* создание **отчётов и графической аналитики** (PDF, Excel, графики).

Таким образом, серверная часть приложения спроектирована с учётом современных требований, возможного масштабирования и удобства сопровождения в условиях действующего производственного предприятия.

### ****1.8 Промежуточные итоги****

Подводя итоги аналитической главы и обоснований архитектурных решений, можно выделить ряд ключевых выводов, определяющих дальнейшее проектирование информационной системы для предприятия ООО «СКТБ ЭлПА».

1. **Необходимость автоматизации хранения и обработки информации** о преобразователях давления ПД на предприятии является объективной. Используемые ранее подходы — Excel-таблицы, локальные файлы и бумажные документы — не обеспечивают должного уровня надёжности, безопасности и эффективности. Их применение стало затруднительным в условиях увеличения объёмов заказов, расширения номенклатуры изделий и усложнения производственных процессов.
2. **Проведённый анализ аналогов и конкурирующих решений** (Excel, Google Sheets, 1С, Access и др.) показал, что ни одна из этих платформ не удовлетворяет всем требованиям предприятия. Среди основных недостатков: отсутствие модульности, слабая поддержка разграничения прав, избыточность функциональности или, напротив, чрезмерная простота.
3. **Выбранная архитектура приложения и технологический стек** (HTML, CSS, JavaScript, PHP, MySQL) полностью соответствуют текущим возможностям предприятия и квалификации исполнителей. Это решение обеспечивает гибкость, простоту внедрения и минимальные издержки. При этом оно сохраняет перспективу масштабирования и развития — как в части клиентского, так и серверного функционала.
4. **Принятая модульная структура и логическое разделение компонентов** (модули: «Клиенты», «Изделия», «Заказы», «Маршрут», «Пользователи») уже на раннем этапе позволила организовать проект по принципам микросервисного мышления. Такая архитектура обеспечит лучшую масштабируемость, а также упрощает реализацию, тестирование и поддержку каждой функциональной части в отдельности.
5. **Интерфейс приложения разрабатывается с учётом универсальности и простоты**. Использование Bootstrap и JavaScript без тяжёлых фреймворков (React, Angular) позволяет добиться высокой производительности и быстрой реакции без дополнительной нагрузки на ресурсы сервера или клиента. Это особенно важно в условиях локального размещения сервиса и использования его сотрудниками с разным уровнем подготовки.
6. **На уровне серверной логики** применение PHP в связке с MySQL доказало свою эффективность на многих подобных проектах. Простота настройки, высокая производительность и широкая поддержка сделали этот стек предпочтительным в условиях ограниченных ресурсов. Использование подготовленных выражений и сессий PHP обеспечивает базовый уровень безопасности и отказоустойчивости.

Таким образом, вся аналитическая и архитектурная база, изложенная в первой главе, создаёт прочный фундамент для последующих этапов разработки. На следующем этапе проектирования будет осуществлена детализация структуры базы данных, разработка интерфейсов взаимодействия с пользователем, реализация логики добавления, редактирования и фильтрации данных, а также описание алгоритмов маршрутизации заказов.

В результате планируется создать удобное, стабильное и безопасное веб-приложение, способное не только удовлетворить текущие потребности ООО «СКТБ ЭлПА», но и послужить платформой для дальнейшего развития цифровой инфраструктуры предприятия.

## ГЛАВА 2. ПРОЕКТИРОВАНИЕ

### ****2.1 Цель и задачи проектирования****

Проектирование информационной системы (ИС) представляет собой один из ключевых этапов жизненного цикла программного обеспечения, в ходе которого разрабатывается архитектурная и логическая структура будущего решения. Этот этап определяет, как именно будет реализована функциональность, выявленная на стадии анализа требований, и каким образом система сможет удовлетворить потребности предприятия.

**Цель проектирования** заключается в создании устойчивой, масштабируемой и понятной архитектуры системы, способной обеспечивать автоматизацию учёта, хранения и обработки данных по преобразователям давления кварцевого типа (ПД), выпускаемым ООО «СКТБ ЭлПА». Разрабатываемая система должна не только покрыть текущие функциональные требования, но и предусматривать возможность последующего расширения и адаптации под изменяющиеся условия эксплуатации.

В рамках проектирования особое внимание уделяется следующим аспектам:

* логической организации сущностей базы данных (изделия, заказы, маршруты, клиенты и пользователи);
* разметке и описанию взаимосвязей между объектами;
* формализации процессов, происходящих в системе, таких как авторизация, фильтрация, добавление данных и изменение статусов заказов;
* описанию поведения системы в различных сценариях и состояниях;
* созданию графических моделей (UML-диаграмм), облегчающих визуализацию и анализ структуры ИС.

Проектирование позволяет не только формализовать внутреннюю структуру системы, но и значительно упростить процесс её последующей реализации. Грамотно спроектированная система легче поддаётся сопровождению, менее подвержена ошибкам и лучше адаптируется под новые требования.

В процессе проектирования были поставлены следующие **основные задачи**:

1. Выявить все ключевые сущности предметной области и определить их характеристики.
2. Описать связи и зависимости между сущностями.
3. Формализовать поведение системы при типичных действиях пользователей.
4. Разработать набор UML-диаграмм, описывающих:

* структуру классов и объектов,
* жизненные циклы объектов (состояния),
* сценарии взаимодействия пользователей с системой (прецеденты),
* последовательности операций при выполнении задач,
* процессы в виде диаграмм деятельности.

1. Обосновать архитектурное разделение клиентской, серверной и базы данных.
2. Подготовить модель развертывания системы с учётом реальных условий предприятия.

В целом, проектирование в данном дипломном проекте направлено на достижение высокой согласованности между структурой, логикой и реализацией системы. На его основе будет построен прочный фундамент для реализации веб-приложения, обеспечивающего цифровизацию документооборота по продукции предприятия.

### ****2.2 Выбор модели жизненного цикла и подхода к проектированию****

Разработка информационной системы, как и любой программной продукции, предполагает прохождение определённых стадий — от анализа и проектирования до реализации, тестирования и внедрения. Для успешного выполнения проекта важно заранее определить модель жизненного цикла, согласно которой будет организована работа.

В рамках данного дипломного проекта используется **итерационно-каскадный подход**, при котором каждая стадия проходит поэтапно, но допускается возвращение к предыдущим шагам для корректировки и уточнения. Это особенно актуально при разработке на основе живой обратной связи от заказчика (в данном случае — представителей ООО «СКТБ ЭлПА»), которые могут вносить правки в требования и интерфейс по ходу работы.

****2.2.1 Обоснование выбора итерационно-каскадной модели****

Полноценные гибкие модели (Scrum, Kanban) предполагают наличие команды и постоянной итеративной работы, что не соответствует формату дипломного проекта. В то же время чисто каскадный (водопадный) подход не позволяет гибко вносить изменения на ранних этапах реализации.

Итерационно-каскадный подход оказался оптимальным, так как позволил:

* сначала провести полный анализ требований и предметной области;
* затем спроектировать структуру системы и разработать макеты интерфейса;
* по мере реализации возвращаться к предыдущим этапам и уточнять диаграммы и архитектуру;
* по результатам внедрения внести коррективы в интерфейс и логику работы.

Такая модель обеспечила достаточную гибкость и контроль над этапами.

****2.2.2 Использование объектно-ориентированного подхода****

В качестве основного подхода к проектированию выбрана **объектно-ориентированная методология**, которая предусматривает представление всех элементов системы в виде объектов с чётко определёнными свойствами (атрибутами) и поведением (методами).

Данный подход целесообразен по ряду причин:

* соответствует структуре данных, представленных в системе (изделие, заказ, пользователь и т.д.);
* обеспечивает хорошую масштабируемость — добавление новых сущностей или изменение логики не нарушает существующую архитектуру;
* улучшает читаемость и логичность модели как для разработчиков, так и для заказчиков;
* позволяет использовать визуальное моделирование на основе **UML-диаграмм**.

Система описывается через набор классов и объектов, связанных ассоциациями, обобщениями или зависимостями. Поведение объектов в различных сценариях визуализируется через диаграммы последовательностей и состояний.

****2.2.3 Применение языка UML****

Для описания структуры и поведения будущей системы используется **язык унифицированного моделирования UML (Unified Modeling Language)**. UML является де-факто стандартом в области визуального проектирования программного обеспечения и обеспечивает единый язык общения между разработчиком, заказчиком и аналитиком.

В дипломной работе применены следующие типы диаграмм UML:

* **диаграммы классов** — для описания структуры системы;
* **диаграммы последовательностей** — для отображения пошагового взаимодействия объектов;
* **диаграммы деятельности** — для визуализации рабочих процессов и логики действий;
* **диаграммы состояний** — для отслеживания жизненного цикла объектов;
* **диаграммы прецедентов** — для представления сценариев использования системы;
* **диаграммы компонентов и развертывания** — для иллюстрации архитектуры и логики размещения системы.

Таким образом, принятый подход к проектированию, основанный на UML и итерационно-каскадной модели жизненного цикла, позволил заложить логичную, устойчивую и масштабируемую структуру будущей системы. На его базе будет построена проектная часть с диаграммами и описаниями ключевых компонентов.

2.3

2.4

### ****2.5 Диаграммы последовательностей и деятельности****

В процессе проектирования информационной системы крайне важно не только определить, какие объекты и сущности участвуют в её работе, но и визуализировать последовательность их взаимодействия во времени. Для этого используются **диаграммы последовательностей** и **диаграммы деятельности**, являющиеся частью стандарта UML.

Они позволяют представить **динамику системы**, т.е. не только то, что делает система, но и **в каком порядке и при каких условиях** происходят те или иные действия. Это особенно важно для сложных процессов, таких как аутентификация пользователей, добавление маршрутов, редактирование данных и т.д.

В данном проекте такие диаграммы были разработаны для ключевых сценариев использования — как с участием пользователя, так и на уровне внутренних компонентов системы.

****2.5.1 Диаграммы последовательностей****

**Диаграмма последовательностей (Sequence diagram)** — это вид UML-диаграммы, отображающий объекты, участвующие в конкретном процессе, и сообщения, которыми они обмениваются, упорядоченные по времени. Она полезна для анализа **взаимодействия между компонентами** системы.

Пример 1. Авторизация пользователя

На диаграмме авторизации последовательно отображены действия:

1. Пользователь вводит логин и пароль в интерфейсе.
2. Система отправляет запрос на сервер через fetch-запрос JavaScript.
3. Серверный скрипт PHP обрабатывает запрос, извлекает данные из базы (users).
4. Пароль проверяется на соответствие хешу.
5. При успешной проверке создаётся сессия, и пользователю возвращается статус «OK».
6. Интерфейс обновляется и открывается доступ к вкладкам.

Эта диаграмма позволяет наглядно увидеть, какие модули участвуют в процессе и в каком порядке. Также можно предусмотреть обработку ошибок (неверный пароль, отсутствие пользователя, блокировка аккаунта).

Пример 2. Добавление маршрута заказа

Сценарий добавления нового маршрута включает:

1. Выбор заказа из выпадающего списка (с фильтрацией по статусу «В работе»).
2. Отправку данных формы маршрута (этап, дата, ответственный) на сервер.
3. Обработку на сервере: проверка полей, формирование SQL-запроса.
4. Добавление записи в таблицу routes.
5. Обновление интерфейса с отображением нового маршрута.

Диаграмма отображает, как фронтенд и backend взаимодействуют при выполнении одной бизнес-операции. Это помогает выявить слабые места (например, отсутствие валидации или некорректные статусы).

****2.5.2 Диаграммы деятельности****

**Диаграмма деятельности (Activity diagram)** — это тип UML-диаграммы, который описывает процесс как поток управляющих действий и условий перехода. Она позволяет моделировать **логические этапы и разветвления процесса**, аналогично блок-схемам, но в более структурированной форме.

Пример 1. Фильтрация заказов по клиенту

В этом процессе участвуют следующие шаги:

* Пользователь выбирает клиента из выпадающего списка.
* Интерфейс отправляет запрос на сервер (параметры фильтрации).
* Сервер выполняет SQL-запрос с WHERE-критерием.
* Полученные данные возвращаются в виде JSON.
* Таблица обновляется, отображая только заказы выбранного клиента.

Диаграмма показывает логическую ветвь: если клиент не выбран, возвращаются все заказы, иначе — только отфильтрованные. Также визуализируются действия пользователя и системы на каждом этапе.

Пример 2. Обновление информации об изделии

Поток действий:

* Пользователь нажимает кнопку «Редактировать».
* Открывается модальное окно с полями.
* После изменений нажимается «Сохранить».
* JS отправляет данные на сервер.
* Сервер обновляет запись в таблице products.
* Отображается уведомление об успешном изменении.

Диаграмма помогает проследить, в каком порядке происходят этапы — от пользовательского действия до обновления базы данных и интерфейса. Также можно встроить проверку условий: если поля не заполнены, сервер вернёт ошибку, и пользователь увидит уведомление.

****2.5.3 Ценность и применение диаграмм****

Применение диаграмм последовательностей и деятельности позволило:

* формализовать бизнес-процессы предприятия;
* наглядно описать логику взаимодействия между компонентами;
* согласовать поведение фронтенда и backend;
* выявить точки отказа и предусмотреть обработку ошибок;
* улучшить понимание архитектуры как для разработчика, так и для технического заказчика.

На основе этих диаграмм были спроектированы модули веб-приложения, структурированы PHP-скрипты и настроено взаимодействие с базой данных. В рамках дипломного проекта создано не менее **четырёх диаграмм последовательностей** и **двух диаграмм деятельности**, охватывающих основные сценарии эксплуатации системы.

### ****2.6 Диаграммы состояний ключевых объектов****

В проектировании информационной системы важно учитывать не только структуру объектов, но и динамику их изменений — какие состояния они могут принимать и как переходят из одного состояния в другое. Это особенно актуально для таких бизнес-сущностей, как заказы, изделия, этапы маршрута, поскольку они проходят определённый жизненный цикл: от создания до завершения или удаления.

Для визуализации таких процессов используются **диаграммы состояний (state diagrams)**. Эти диаграммы позволяют понять, каким образом объект «живёт» в системе, в какие моменты происходят переходы, и какие действия или условия их вызывают.

****2.6.1 Состояния объекта «Заказ»****

Объект **«Заказ»** (Order) — одна из центральных сущностей системы. Он связывает клиента с определённым изделием и описывает процесс производства от момента регистрации до завершения.

Диаграмма состояний заказа включает следующие этапы:

1. **Создан** — заказ зарегистрирован в системе, но ещё не передан в работу.
2. **В работе** — заказ принят к исполнению, начат производственный цикл.
3. **На сборке / На испытаниях** — уточнённые подэтапы (отражаются в маршруте, но могут быть вынесены отдельно).
4. **Завершён** — заказ полностью выполнен и готов к отгрузке.
5. **Отгружен** — заказ передан клиенту, закрыт.
6. **Отменён** — заказ аннулирован по инициативе клиента или предприятия.

Переходы между состояниями происходят в зависимости от действий пользователя или системы. Например, при изменении статуса маршрута с «Сборка» на «Испытания» система автоматически обновляет общее состояние заказа.

Также реализована защита от некорректных переходов (например, нельзя перевести заказ из состояния «Отменён» в «В работе» без создания нового).

****2.6.2 Состояния объекта «Изделие»****

Объект **«Изделие»** (Product) в системе представляет собой техническую единицу — преобразователь давления с уникальными характеристиками. Изделие в системе может проходить несколько жизненных этапов:

1. **Создано** — изделие занесено в базу, но пока не привязано ни к одному заказу.
2. **Привязано к заказу** — изделие участвует в производственном процессе.
3. **Редактируется** — изделие проходит корректировку данных (например, по новым требованиям клиента).
4. **Архивировано** — изделие устарело или снято с производства, доступно только для просмотра.
5. **Удалено** — изделие исключено из активной базы данных (возможно логическое удаление).

Состояние изделия может влиять на его доступность в интерфейсе при выборе для нового заказа. Например, архивные изделия отображаются только администраторам или с соответствующей пометкой.

****2.6.3 Состояния объекта «Пользователь»****

Объект **«Пользователь»** (User) является неотъемлемой частью информационной системы, поскольку любая работа с базой данных возможна только после прохождения авторизации. В зависимости от роли, каждый пользователь получает доступ к определённым функциям: просмотр, редактирование, добавление, администрирование.

Для повышения безопасности, прозрачности и управляемости система должна отслеживать текущее состояние каждого пользователя. На основании анализа бизнес-логики проекта выделены следующие основные состояния объекта «User»:

1. **Создан** — пользователь добавлен администратором или зарегистрирован в системе, но ещё не активирован (например, ожидает подтверждения).
2. **Активен** — пользователь успешно авторизован и имеет право входа в систему в рамках своей роли.
3. **Заблокирован** — пользователь временно лишён доступа. Это может быть связано с нарушением правил, временным отстранением или административной блокировкой. Заблокированный пользователь не может авторизоваться, но его учётная запись и история действий сохраняются.

****2.6.4 Ценность диаграмм состояний****

Применение диаграмм состояний дало возможность:

* чётко регламентировать бизнес-логику поведения объектов;
* предотвратить недопустимые действия со стороны пользователя (например, удаление завершённого маршрута);
* упростить реализацию логики переходов в коде;
* повысить прозрачность взаимодействия между модулями системы;
* облегчить техническое задание для программиста и понятность для заказчика.

Эти диаграммы особенно важны в системах, где объекты проходят через многоступенчатые процессы с возможностью возврата, отмены или временной приостановки. В нашем случае это заказы и маршруты, тесно связанные с производственным циклом.

### ****2.7 Диаграммы прецедентов****

При проектировании информационной системы важно описать, как конечные пользователи взаимодействуют с системой — какие действия они выполняют и какую реакцию ожидают от системы. Для этой цели в UML применяется **диаграмма прецедентов (Use Case Diagram)**.

Диаграмма прецедентов позволяет отразить **взаимосвязь между актёрами (пользователями)** и **сценариями использования системы**, показывая, какие функции доступны для каждого типа пользователя. Это особенно важно для систем с разграничением прав — как в случае веб-приложения для базы данных преобразователей давления на предприятии ООО «СКТБ ЭлПА».

****2.7.1 Основные актёры системы****

В данной системе выделяются два ключевых типа пользователей (актёров):

* **Администратор** — пользователь с полным набором прав: доступ ко всем вкладкам, возможность редактирования, удаления и управления другими пользователями.
* **Сотрудник (Пользователь)** — ограниченный по правам пользователь, работающий с заказами, маршрутами, изделиями, но без возможности управлять чужими записями или изменять структуру базы данных.

Дополнительно можно выделить роль **Гость**, которой разрешён только просмотр данных (например, в демонстрационных целях), но в рамках диплома акцент сделан на двух активных ролях: **Администратор** и **Сотрудник**.

****2.7.2 Диаграмма прецедентов: Администратор****

На диаграмме прецедентов для администратора отображены следующие действия:

* Авторизоваться в системе;
* Управлять пользователями (добавление, блокировка, удаление);
* Просматривать и редактировать клиентов;
* Добавлять, редактировать, удалять изделия;
* Управлять заказами (создание, изменение, изменение статуса);
* Отслеживать маршруты выполнения заказа;
* Импортировать/экспортировать данные (в перспективе);
* Просматривать логи или отчёты (в будущем).

Прецеденты сгруппированы по вкладкам интерфейса: «Клиенты», «Изделия», «Заказы», «Маршрут», «Пользователи». Каждому действию соответствует отдельный прецедент (use case), показывающий функциональные границы и ответственность пользователя.

**Связи между прецедентами**:

* «Добавить пользователя» → включает → «Назначить роль»;
* «Редактировать заказ» → расширяет → «Изменить статус»;
* «Удалить изделие» → включает → «Проверка зависимости в заказах».

****2.7.3 Диаграмма прецедентов: Сотрудник****

Сотрудник имеет следующий набор прецедентов:

* Авторизоваться в системе;
* Просматривать заказы, привязанные к его участку;
* Добавлять и редактировать маршруты выполнения;
* Просматривать характеристики изделий;
* Просматривать клиентов (без редактирования);
* Заполнять поля в форме маршрута (дата, примечания, статус);
* Отмечать выполнение этапов маршрута.

При этом сотрудник **не может**:

* изменять список пользователей;
* удалять существующие изделия или заказы;
* редактировать маршруты других сотрудников;
* менять глобальные настройки системы.

Таким образом, система обеспечивает разграничение прав доступа, отражающее реальные бизнес-процессы предприятия: у каждого сотрудника есть ограниченная зона ответственности, тогда как администратор координирует работу в целом.

****2.7.4 Назначение диаграмм прецедентов в проекте****

Использование диаграмм прецедентов позволило:

* **выделить основные действия пользователей**;
* **увидеть, где требуется ограничение прав** и реализовать соответствующую проверку;
* **структурировать логику интерфейса**, распределив действия по вкладкам;
* **проанализировать полноту функциональности**, чтобы исключить дублирование или пропуски;
* **обосновать архитектурные решения** (например, разделение пользовательского интерфейса по ролям).

Диаграммы прецедентов стали связующим звеном между **логикой пользовательского взаимодействия** и **технической реализацией API и интерфейса.**

### ****2.8 Диаграмма компонентов****

На этапе проектирования важным аспектом является **архитектурное разбиение системы** на логические и физические модули, каждый из которых выполняет строго определённые функции. Это необходимо как для повышения устойчивости системы, так и для облегчения сопровождения, масштабирования и последующего внедрения новых функций.

Для визуализации архитектурной структуры используется **диаграмма компонентов (Component Diagram)**, отражающая состав системы в виде взаимосвязанных модулей. В рамках разрабатываемого веб-приложения базы данных преобразователей давления ПД для ООО «СКТБ ЭлПА» такая диаграмма показывает взаимодействие между клиентской частью (интерфейс), серверной логикой и базой данных.

****2.8.1 Архитектура клиент-сервер****

Проект построен по **классической архитектуре клиент-сервер**, где:

* **клиентская часть (frontend)** работает в браузере пользователя и отвечает за визуальное представление, взаимодействие с пользователем и отправку запросов;
* **серверная часть (backend)** реализована на PHP и обрабатывает запросы, управляет логикой, выполняет проверку прав и взаимодействует с базой данных;
* **база данных (MySQL)** хранит всю бизнес-информацию (данные о заказах, клиентах, изделиях, маршрутах, пользователях и др.).

****2.8.2 Основные компоненты системы****

На диаграмме компонентов выделяются следующие ключевые модули:

**1. Веб-клиент (Интерфейс пользователя)**

* HTML-шаблоны и CSS-стили (оформление интерфейса);
* JavaScript-скрипты (динамика, логика интерфейса);
* Bootstrap (адаптивная сетка и элементы интерфейса);
* Fetch API (отправка запросов к серверу и получение ответов в формате JSON).

Функции:

* отображение таблиц и форм;
* обработка действий пользователя;
* отправка и получение данных;
* обновление DOM без перезагрузки.

**2. API-обработчики на PHP**

Функции:

* приём данных от клиента;
* валидация;
* обращение к БД;
* формирование ответа в формате JSON.

**3. СУБД (MySQL)**

Хранилище всех данных приложения. Основные таблицы:

* clients — информация о заказчиках;
* products — характеристики изделий;
* orders — заказы клиентов;
* routes — этапы прохождения заказов;
* users — логины, пароли, роли;
* (дополнительно: логи, справочники и т.п.).

Каждая таблица связана с другими через внешние ключи. Используются индексы для ускорения поиска и фильтрации.

****2.8.3 Взаимодействие компонентов****

В процессе работы пользователь взаимодействует с интерфейсом через браузер. Его действия (например, «добавить заказ» или «просмотреть маршрут») инициируют вызов JavaScript-функций, которые отправляют данные с помощью Fetch API на определённый PHP-обработчик.

PHP-обработчик получает данные, проверяет их, взаимодействует с базой данных через расширение mysqli, и формирует ответ, который в виде JSON возвращается обратно клиенту. Интерфейс обновляется без перезагрузки, позволяя достичь эффекта интерактивного приложения.

****2.8.4 Преимущества выбранной архитектуры компонентов****

Разбиение системы на отдельные модули дало следующие преимущества:

* **модульность**: каждый компонент можно дорабатывать независимо;
* **масштабируемость**: возможна замена PHP-обработчиков на более мощный фреймворк или язык;
* **простота отладки**: ошибки изолированы в конкретном модуле;
* **гибкость**: можно подключить внешние сервисы через API;
* **прозрачность кода**: структура системы понятна как разработчику, так и администратору.

В дальнейшем можно вынести API в отдельный сервис, создать мобильное приложение, подключить REST-интерфейс или реализовать экспорт данных в PDF/Excel.

### ****2.9 Диаграмма развертывания****

Для полноценной работы информационной системы важно не только спроектировать структуру программных компонентов, но и определить, **на каких узлах и в каком окружении** они будут развернуты. От выбора архитектуры развертывания зависит стабильность, защищённость и масштабируемость веб-приложения.

В рамках настоящего дипломного проекта система разрабатывалась с расчётом на размещение внутри локальной сети предприятия **ООО «СКТБ ЭлПА»**, что обеспечивает высокую скорость доступа, контроль над инфраструктурой и отсутствие зависимости от внешнего интернета. Для описания структуры развертывания используется **диаграмма развертывания (Deployment Diagram)** — один из типов UML-диаграмм, отображающих размещение компонентов на физических или виртуальных устройствах.

****2.9.1 Общая структура размещения****

На диаграмме развертывания представлены следующие основные узлы:

1. **Клиентское устройство** (рабочее место пользователя):

* Обычный ПК или ноутбук;
* Веб-браузер (Google Chrome, Firefox и др.);
* Не требует установки специального ПО — вся работа идёт через интерфейс в браузере.

1. **Веб-сервер**:

* Apache или Nginx, входящий в состав OpenServer (или аналогичной локальной платформы);
* Расположен на выделенном компьютере внутри локальной сети предприятия;
* Обрабатывает входящие HTTP-запросы от пользователей и перенаправляет их в PHP-обработчики.

1. **PHP-интерпретатор**:

* Интерпретирует логику сервера, реализованную в PHP-файлах;
* Обеспечивает работу с базой данных, авторизацию, валидацию данных, генерацию JSON-ответов;
* Запускается на том же сервере, что и веб-сервер.

1. **СУБД (MySQL)**:

* Установлена на том же сервере или на соседнем узле в локальной сети;
* Обеспечивает хранение всех бизнес-данных (заказы, изделия, пользователи, маршруты и т.д.);
* Связана с PHP-обработчиками через локальный сокет или TCP/IP.

****2.9.2 Локальное размещение: преимущества и ограничения****

Выбор локального размещения обусловлен рядом факторов:

* **Безопасность**: отсутствие доступа извне исключает риски взлома;
* **Независимость от интернета**: система работает даже при отсутствии внешнего соединения;
* **Минимальные расходы**: нет необходимости в аренде облака или хостинга;
* **Контроль доступа**: администратор сам управляет пользователями и конфигурацией.

При этом система спроектирована так, что при необходимости её можно перенести на внешний сервер (VPS или облачный хостинг) без полной переработки архитектуры. Вся логика взаимодействия организована по универсальным протоколам и стандартам.

### ****2.10 Промежуточные выводы по проектированию****

Проведённое проектирование информационной системы хранения и учёта данных о преобразователях давления ПД для ООО «СКТБ ЭлПА» позволило сформировать чёткое и обоснованное представление о её архитектуре, структуре и взаимодействии компонентов.

В рамках проектирования выполнены следующие ключевые действия:

* определены **основные бизнес-сущности** (клиенты, изделия, заказы, пользователи) и их взаимосвязи;
* разработаны **диаграммы классов**, наглядно отображающие структуру данных и логические связи между объектами;
* построены **диаграммы последовательностей и деятельности**, демонстрирующие сценарии использования системы и логические потоки обработки информации;
* определены **состояния жизненного цикла объектов**, включая заказы, изделия и пользователей;
* описаны **прецеденты использования (use cases)** для различных типов пользователей, включая администратора и сотрудника;
* составлены **диаграммы компонентов и развертывания**, отражающие модульную структуру системы и её физическое размещение.

Принятые проектные решения основываются на принципах модульности, безопасности, расширяемости и простоты сопровождения. Такой подход позволяет:

* упростить реализацию и поэтапное тестирование компонентов;
* обеспечить возможность модернизации отдельных модулей без полной переработки системы;
* адаптировать систему к новым условиям и функциональным требованиям;
* обеспечить высокую прозрачность взаимодействия между интерфейсом, логикой и базой данных.

Особое внимание в проектировании уделено **разграничению прав доступа**, что обеспечивает безопасную и контролируемую работу пользователей. Система продумана таким образом, чтобы быть понятной, удобной и устойчивой в условиях промышленной эксплуатации.

Таким образом, проектная часть дипломной работы заложила фундамент для реализации системы, максимально приближённой к реальным потребностям предприятия. Все спроектированные элементы согласованы между собой и обеспечивают логическую завершённость модели. Следующий этап — **реализация программной системы** на базе выбранных технологий.

## ГЛАВА 3. ПРОГРАММНАЯ РЕАЛИЗАЦИЯ

Целью третьей главы является переход от теоретического проектирования к непосредственной программной реализации системы, обеспечивающей автоматизацию хранения и управления данными о преобразователях давления ПД. В ходе разработки было принято решение реализовать приложение в виде модульного веб-интерфейса с логическим разделением на функциональные блоки — «Клиенты», «Изделия», «Заказы», «Пользователи» — каждый из которых охватывает определённую часть бизнес-логики предприятия.

Хотя система построена на монолитной платформе (PHP + MySQL), в архитектурном плане она приближается к **принципам микросервисного мышления**: каждый раздел интерфейса опирается на отдельные API-обработчики, свои таблицы в базе данных и собственные JavaScript-функции. Такой подход позволяет упростить сопровождение кода, минимизировать связность между модулями и облегчить будущую миграцию к полноценной сервисной архитектуре.

Перед тем как перейти к детальному описанию реализации каждого функционального блока, следует обозначить **технологические и архитектурные принципы**, на которых построено приложение:

* **Модульность**. Каждый раздел системы (например, «Изделия» или «Заказы») разрабатывался как изолированный функциональный модуль, с собственными интерфейсами, таблицами и серверной логикой.
* **Чистое разделение логики**. Интерфейс, логика обработки запросов и доступ к базе данных строго разделены. JavaScript обрабатывает пользовательские действия, PHP-скрипты — запросы, а MySQL — хранение и извлечение информации.
* **Асинхронное взаимодействие клиента и сервера**. Все операции выполняются без перезагрузки страницы: данные передаются по протоколу HTTP с использованием fetch, а ответы возвращаются в формате JSON. Это обеспечивает отзывчивость интерфейса и комфортную работу с системой.
* **Ролевая модель доступа**. Пользовательская часть системы включает авторизацию, распределение прав и изоляцию функциональности в зависимости от роли: сотрудник или администратор.
* **Лёгкость переноса**. Система развёрнута в локальной сети, но её структура спроектирована таким образом, чтобы в будущем была возможность переноса на внешний сервер или облачную инфраструктуру без полной переработки архитектуры.

Далее в главе последовательно описываются все ключевые разделы интерфейса и логика их функционирования. Для каждого модуля раскрываются структура клиентской и серверной части, принципы взаимодействия с базой данных и применённые технические решения.

### ****3.1 Вкладка «Клиенты»****

Вкладка «Клиенты» предназначена для ведения справочника заказчиков, сотрудничающих с предприятием ООО «СКТБ ЭлПА». Этот раздел обеспечивает хранение и актуализацию информации о клиентах: их наименовании, местоположении и дополнительных характеристиках. Указанные данные используются в других разделах системы, в первую очередь — при создании заказов.

Интерфейс вкладки разработан с упором на **простоту и удобство использования**, и позволяет выполнять все операции без перезагрузки страницы.

****3.1.1 Структура и логика интерфейса****

Во вкладке реализованы следующие элементы:

* заголовок и кнопка "Добавить заказчика";
* скрываемая форма с полями:
* Название заказчика;
* Местоположение;
* Характеристика;
* таблица со списком всех клиентов.

При открытии страницы или переключении на вкладку происходит автоматическая загрузка списка клиентов из базы данных. Добавление нового клиента осуществляется через AJAX-запрос — форма отправляется функцией addClient(event) в script.js. После успешного добавления данные обновляются без перезагрузки.

Листинг 3.1.1.1 Пример HTML-разметки формы

|  |
| --- |
| <form id="clientForm" class="hidden" onsubmit="addClient(event)">  <input type="text" placeholder="Название заказчика" required id="clientName">  <input type="text" placeholder="Местоположение" required id="clientSite">  <input type="text" placeholder="Характеристика" id="clientCharact">  <button type="submit">Сохранить</button>  </form> |

****3.1.2 Реализация клиентской логики****

В script.js функция addClient():

* предотвращает стандартную отправку формы;
* собирает значения полей в FormData;
* выполняет fetch() с методом POST на add\_client.php;
* в случае успеха:
  + обновляет таблицу;
  + сбрасывает форму;
  + скрывает форму ввода.

Листинг 3.1.2.1 Функция addClient

|  |
| --- |
| function addClient(event) {  event.preventDefault();  const formData = new FormData();  formData.append('clientName', document.getElementById('clientName').value);  formData.append('clientSite', document.getElementById('clientSite').value);  formData.append('clientCharact', document.getElementById('clientCharact').value);  fetch('add\_client.php', {  method: 'POST',  body: formData  })  .then(res => res.json())  .then(data => {  if (data.success) {  loadClients();  event.target.reset();  toggleForm('clientForm');  } else {  alert("Ошибка: " + data.error);  }  });  } |

Загрузка списка клиентов реализована в функции loadClients(), которая:

* выполняет GET-запрос на get\_clients.php;
* получает массив клиентов в формате JSON;
* динамически заполняет таблицу clientTableBody.

****3.1.3 Серверная обработка данных****

Добавление клиента обрабатывается в add\_client.php следующим образом:

* создаётся подключение к базе данных SQLite через PDO;
* подготавливается SQL-запрос INSERT INTO Clients;
* данные вставляются с использованием подготовленного выражения;
* при успешной вставке возвращается { success: true }, иначе — ошибка.

Листинг 3.1.3.1 add\_client.php

|  |
| --- |
| if ($\_SERVER['REQUEST\_METHOD'] === 'POST') {  try {  $db = new PDO('sqlite:db.sqlite3');  $stmt = $db->prepare("INSERT INTO Clients (clientName, clientSite, clientCharact) VALUES (?, ?, ?)");  $stmt->execute([  $\_POST['clientName'],  $\_POST['clientSite'],  $\_POST['clientCharact']  ]);  echo json\_encode(['success' => true]);  } catch (PDOException $e) {  echo json\_encode(['success' => false, 'error' => $e->getMessage()]);  }  } |

Для получения списка клиентов используется get\_clients.php, который выполняет простой SELECT и возвращает результат в формате JSON:

Листинг 3.1.3.2 Фрагмент get\_clients.php

|  |
| --- |
| $db = new PDO('sqlite:db.sqlite3');  $stmt = $db->query("SELECT clientName, clientSite, clientCharact FROM Clients");  $clients = $stmt->fetchAll(PDO::FETCH\_ASSOC);  echo json\_encode($clients); |

### ****3.2 Вкладка «Изделия»****

Вкладка «Изделия» предназначена для управления справочником продукции предприятия, а именно — преобразователями давления (ПД), производимыми или учитываемыми ООО «СКТБ ЭлПА». Данный раздел позволяет добавлять новые изделия, просматривать уже внесённые записи, а также сохранять описание, категорию, внешний вид и дополнительные примечания к каждому изделию.

Информация из вкладки «Изделия» тесно связана с другими модулями системы, прежде всего — с формированием заказов и маршрутов, где изделие является ключевым объектом.

****3.2.1 Структура и интерфейс вкладки****

Интерфейс вкладки представлен следующим образом:

1. Заголовок «Изделия» и кнопка **«Добавить датчик»**;
2. Скрытая форма ввода с четырьмя полями:
   * Название изделия (productName);
   * Категория (productCate);
   * Внешний вид (productAppearance);
   * Примечание (productNote);
3. Таблица, отображающая список всех зарегистрированных изделий.

Вкладка оформлена с использованием стандартного HTML и кастомного CSS (в сочетании с Bootstrap 5), что делает интерфейс адаптивным и визуально единообразным.

****3.2.2 Добавление изделия: клиентская часть****

Пользователь заполняет форму, после чего данные обрабатываются функцией addProduct(event) из script.js. Функция:

1. блокирует стандартное поведение формы;
2. собирает значения всех полей в объект FormData;
3. отправляет POST-запрос на серверный скрипт add\_product.php;
4. при успешной вставке:

* вызывает обновление списка (loadProducts());
* сбрасывает поля формы;
* скрывает форму ввода.

Листинг 3.2.2.1 Функция addProduct

|  |
| --- |
| function addProduct(event) {  event.preventDefault();  const formData = new FormData();  formData.append('productName', document.getElementById('productName').value);  formData.append('productCate', document.getElementById('productCate').value);  formData.append('productAppearance', document.getElementById('productAppearance').value);  formData.append('productNote', document.getElementById('productNote').value);  fetch('add\_product.php', {  method: 'POST',  body: formData  })  .then(res => res.json())  .then(data => {  if (data.success) {  loadProducts();  event.target.reset();  toggleForm('productForm');  } else {  alert("Ошибка: " + data.error);  }  });  } |

****3.2.3 Серверная часть: добавление и чтение данных****

На сервере выполняется подключение к базе данных SQLite через PDO, а затем выполняется SQL-запрос INSERT INTO Products(...).

Листинг 3.2.3.1 Фрагмент add\_product.php

|  |
| --- |
| if ($\_SERVER['REQUEST\_METHOD'] === 'POST') {  try {  $db = new PDO('sqlite:db.sqlite3');  $stmt = $db->prepare("INSERT INTO Products (productName, productCATE, productAppearance, productNote) VALUES (?, ?, ?, ?)");  $stmt->execute([  $\_POST['productName'],  $\_POST['productCate'],  $\_POST['productAppearance'],  $\_POST['productNote']  ]);  echo json\_encode(['success' => true]);  } catch (PDOException $e) {  echo json\_encode(['success' => false, 'error' => $e->getMessage()]);  }  } |

Обработчик get\_products.php формирует JSON-массив из записей таблицы Products.

Листинг 3.2.3.2 Фрагмент get\_products.php

|  |
| --- |
| try {  $db = new PDO('sqlite:db.sqlite3');  $stmt = $db->query("SELECT productName, productCATE, productAppearance, productNote FROM Products");  $products = $stmt->fetchAll(PDO::FETCH\_ASSOC);  echo json\_encode($products);  } catch (PDOException $e) {  echo json\_encode(['error' => $e->getMessage()]);  } |

****3.2.4 Загрузка и отображение изделий****

После загрузки страницы или добавления нового изделия вызывается функция loadProducts():

Листинг 3.2.4.1 Функция loadProducts()

|  |
| --- |
| function loadProducts() {  fetch('get\_products.php')  .then(res => res.json())  .then(data => {  const tbody = document.getElementById('productTableBody');  tbody.innerHTML = '';  data.forEach(p => {  const row = document.createElement('tr');  row.innerHTML = `  <td>${p.productName}</td>  <td>${p.productCATE}</td>  <td>${p.productAppearance}</td>  <td>${p.productNote}</td>  `;  tbody.appendChild(row);  });  });  } |

Это обеспечивает актуальное отображение информации в реальном времени без необходимости обновлять страницу.

### ****3.3 Вкладка «Заказы»****

Вкладка «Заказы» служит ключевым звеном информационной системы и предназначена для создания и управления производственными заказами. Заказ — это связующая сущность между клиентом и изделием, отражающая факт договорённости, количество продукции, плановые и фактические сроки, а также текущий статус выполнения.

Функциональность вкладки охватывает:

* создание нового заказа;
* отображение списка заказов;
* просмотр подробной информации;
* запуск заказа в работу.

****3.3.1 Интерфейс и структура формы****

Вкладка содержит следующие элементы:

* заголовок и кнопка «Добавить заказ»;
* скрытая форма создания нового заказа;
* таблица с перечнем заказов;
* дополнительная область для просмотра подробностей по каждому заказу.

Поля формы включают:

* ID клиента;
* ID изделия;
* количество изделий (план);
* количество к запуску;
* назначенная дата;
* фактическая дата (опционально);
* примечание.

Форма реализована в HTML и скрыта по умолчанию. Она становится видимой по нажатию на кнопку. Данные отправляются на сервер через JavaScript-функцию addOrder(event).

****3.3.2 Обработка создания заказа: клиентская часть****

Функция addOrder(event):

* собирает значения всех полей формы;
* отправляет POST-запрос на add\_order.php;
* при успехе сбрасывает форму, скрывает её и обновляет список заказов.

Листинг 3.3.2.1 Функция addOrder()

|  |
| --- |
| function addOrder(event) {  event.preventDefault();  const formData = new FormData();  formData.append('clientID', document.getElementById('orderClientID').value);  formData.append('productID', document.getElementById('orderProductID').value);  formData.append('productsNum', document.getElementById('orderNum').value);  formData.append('productsNumStart', document.getElementById('orderNumStart').value);  formData.append('orderDateReceipt', document.getElementById('orderDatePlan').value);  formData.append('assignedExecutionDate', document.getElementById('orderDateFact').value);  formData.append('orderStatus', 'Ожидает'); // начальный статус  formData.append('orderN', Date.now()); // временный номер  fetch('add\_order.php', {  method: 'POST',  body: formData  })  .then(res => res.json())  .then(data => {  if (data.success) {  loadOrders();  event.target.reset();  toggleForm('orderForm');  } else {  alert("Ошибка: " + data.error);  }  });  } |

****3.3.3 Серверная логика: добавление и получение заказов****

Обработчик add\_order.php сохраняет заказ в таблицу Orders через подготовленный SQL-запрос:

Листинг 3.3.3.1 Фрагмент add\_order.php

|  |
| --- |
| $stmt = $db->prepare("  INSERT INTO Orders (orderN, clientID, orderDateReceipt, assignedExecutionDate, productID, productsNumStart, productsNum, orderStatus)  VALUES (?, ?, ?, ?, ?, ?, ?, ?)  ");  $stmt->execute([  $\_POST['orderN'],  $\_POST['clientID'],  $\_POST['orderDateReceipt'],  $\_POST['assignedExecutionDate'],  $\_POST['productID'],  $\_POST['productsNumStart'],  $\_POST['productsNum'],  $\_POST['orderStatus']  ]); |

Обработчик get\_orders.php возвращает список заказов в формате JSON. Примечание: в твоём коде этот файл содержит SQL-запрос к таблице Products, что, вероятно, ошибка — он должен выбирать данные из Orders.

****3.3.4 Просмотр подробной информации о заказе****

При нажатии на кнопку «Подробнее» в таблице заказов вызывается функция openOrderDetails(orderID), которая отправляет GET-запрос к обработчику get\_order\_details.php. Полученные данные отображаются в блоке подробностей, где пользователь может ознакомиться с параметрами выбранного заказа:

* уникальный номер заказа;
* клиент (ID);
* изделие (ID);
* плановая и фактическая даты исполнения;
* статус выполнения;
* количество изделий по плану и к запуску.

В дополнение к просмотру деталей, система предоставляет возможность **запуска заказа в работу**, если он ещё не имеет статуса «В работе».

Для этого используется кнопка **«Запустить в работу»**, отображаемая в блоке с деталями заказа. При нажатии выполняется функция startProduction(orderID), которая отправляет POST-запрос на start\_order.php.

Скрипт start\_order.php на сервере:

* принимает идентификатор заказа;
* выполняет SQL-команду обновления поля orderStatus на значение **«В работе»**;
* возвращает JSON-ответ, подтверждающий запуск.

Листинг 3.3.4.1 Фрагмент start\_order.php

|  |
| --- |
| $stmt = $db->prepare("UPDATE Orders SET orderStatus = 'В работе' WHERE orderID = ?");  $stmt->execute([$orderID]); |

Функция startProduction(orderID) на клиенте:

* предварительно спрашивает подтверждение у пользователя (confirm);
* отправляет запрос через fetch;
* при успехе скрывает блок деталей и обновляет список заказов.

Таким образом, реализована простая, но функциональная система управления жизненным циклом заказа. Статус «В работе» используется в других модулях — например, только такие заказы доступны для добавления маршрутов на соответствующей вкладке. Это делает бизнес-логику системы согласованной и устойчивой к логическим ошибкам.

### ****3.4 Вкладка «Маршрут»****

Вкладка «Маршрут» предназначена для фиксации этапов прохождения заказа в процессе производства, от запуска в работу до выполнения всех производственных операций. Каждый маршрутный шаг привязан к конкретному заказу и содержит информацию о дате, номере этапа, изделии и, при необходимости, серийном номере.

Функциональность вкладки реализует трассировку выполнения, что позволяет отслеживать ход исполнения заказов в разрезе продукции. Это особенно важно в контексте промышленного производства, где требуется строгий учёт исполнения поэтапно.

****3.4.1 Структура интерфейса и форма ввода****

Интерфейс вкладки включает:

* заголовок «Маршруты» и кнопку «Добавить маршрут»;
* форму ввода с полями:
* заказ (выбирается из выпадающего списка — только заказы со статусом "В работе");
* ID изделия;
* этап маршрута (текстовое поле);
* дата этапа;
* серийный номер изделия (необязательно);
* примечание;
* таблицу со всеми существующими маршрутами, сортированными по дате (сначала последние).

Форма скрыта по умолчанию и отображается при необходимости. Выбор заказа осуществляется из сформированного списка заказов, полученных из обработчика get\_orders.php, отфильтрованных по статусу "В работе".

****3.4.2 Добавление маршрута: клиентская часть****

При отправке формы вызывается функция addRoute(event), которая:

1. формирует объект FormData;
2. отправляет POST-запрос на add\_route.php;
3. при успехе:

* очищает форму;
* вызывает функцию обновления таблицы маршрутов.

Листинг 3.4.2.1 Функция addRoute

|  |
| --- |
| function addRoute(event) {  const form = event.target;  const formData = new FormData(form);  fetch('add\_route.php', {  method: 'POST',  body: formData  })  .then(res => res.json())  .then(data => {  if (data.success) {  alert('Маршрут успешно добавлен');  form.reset();  } else {  alert('Ошибка: ' + data.error);  }  }); |

**М**аршруты можно добавлять **только для заказов, находящихся в статусе «В работе»**, что реализовано логикой фильтрации в функции loadWorkingOrders().

****3.4.3 Загрузка маршрутов****

Для отображения существующих маршрутов используется функция loadRoutes(orderID), которая:

* загружает все маршруты через get\_routes.php;
* фильтрует их по конкретному заказу, если указан orderID;
* отображает в таблице с перечислением ключевых данных: этап, дата, ID изделия, серийный номер, примечание.

Функциональность реализована с возможностью **переключения между заказами** из выпадающего списка и просмотра маршрутов только по выбранному заказу.

****3.4.4 Серверная часть: обработка маршрутов****

Обработчик add\_route.php получает данные с клиента и сохраняет их в таблицу Routes. Используется подготовленный SQL-запрос:

Листинг 3.4.4.1 Фрагмент add\_route.php

|  |
| --- |
| $stmt = $db->prepare("INSERT INTO Routes (productID, routeN, routeDate, orderID, productSerial, routeNote)  VALUES (?, ?, ?, ?, ?, ?)");  $stmt->execute([  $\_POST['productID'],  $\_POST['routeN'],  $\_POST['routeDate'],  $\_POST['orderID'],  $\_POST['productSerial'],  $\_POST['routeNote']  ]); |

Отладчик get\_routes.php выполняет запрос к таблице Routes и возвращает маршруты в формате JSON, отсортированные по дате:

Листинг 3.4.4.2 Фрагмент get\_routes.php

|  |
| --- |
| $stmt = $db->query("  SELECT routeN, orderID, productID, routeDate, productSerial, routeNote  FROM Routes  ORDER BY routeDate DESC  "); |

На основе полученных данных формируется таблица маршрутов на клиенте.

## ЗАКЛЮЧЕНИЕ

## СПИСОК ЛИТЕРАТУРЫ

1. Системы управления базами данных и проектирование баз данных : : Учеб. пособие для студентов спец. "Системы автоматизир. проектирования" / В. Л. Торхов ; Моск. горн. ин-т. - М. : МГИ, 1987-. - 20 см.
2. Польченко Максим Александрович БЕЗОПАСНОСТЬ БАЗ ДАННЫХ // E-Scio. 2023. №3 (78). URL: https://cyberleninka.ru/article/n/bezopasnost-baz-dannyh (дата обращения: 18.05.2025).
3. Баранова Елизавета Михайловна, Баранов Андрей Николаевич, Борзенкова Светлана Юрьевна, Васин Кирилл Игоревич, Перезябов Владислав Сергеевич СОВРЕМЕННЫЕ ТЕНДЕНЦИИ ЗАЩИТЫ БАЗ ДАННЫХ ВЕБ-ПРИЛОЖЕНИЙ ОТ SQL ИНЪЕКЦИЙ // Известия ТулГУ. Технические науки. 2023. №12. URL: https://cyberleninka.ru/article/n/sovremennye-tendentsii-zaschity-baz-dannyh-veb-prilozheniy-ot-sql-inektsiy (дата обращения: 18.05.2025).
4. Базы данных. Системы управления базами данных [Текст] : учебное пособие : для студентов вузов / М. В. Баканов, В. В. Романова, Т. П. Крюкова ; Федеральное агентство по образованию, Кемеровский технологический ин-т пищевой пром-сти. - Кемерово : Кемеровский технологический ин-т пищевой пром-сти, 2010. - 164 с. : цв. ил.; 20 см.; ISBN 978-5-89289-615-3