Федеральное государственное бюджетное образовательное учреждение высшего образования

**РОССИЙСКАЯ АКАДЕМИЯ  
НАРОДНОГО ХОЗЯЙСТВА и ГОСУДАРСТВЕННОЙ СЛУЖБЫ  
при ПРЕЗИДЕНТЕ РОССИЙСКОЙ ФЕДЕРАЦИИ**

**ЗАПАДНЫЙ ФИЛИАЛ**

**ОТЧЕТ**

о прохождении учебной практики по профилю специальности

ПМ.02 Осуществление интеграции программных модулей

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| студента | *23ИСПп5-о9* | | | группы | | | *3* | | курса |
| *Аникин Михаил Андреевич* | | | | | | | | | |
| *Фамилия Имя Отчество* | | | | | | | | | |
| по специальности | | *09.02.07 Информационные системы и программирование* | | | | | | | |
|  |  | *Код, наименование специальности* | | | | | | | |
| Место прохождения практики | | | | | | *Западный филиал РАНХиГС* | | | |
|  | | | | | | | | | |
| Период прохождения практики | | | | | | *с 22.09.2025 по 05.10.2025* | | | |
|  |  | |  | | |  | | |  |
|  | Дата сдачи | | | | | *05.10.2025* | | | |
|  | Дата защиты | | | | | *05.10.2025* | | | |
|  | Оценка | | | | |  | | | |
|  |  | | | | |  | | | |
|  | | | | | | Руководитель практики | | | |
|  |  | | | | |  | | | |
|  |  | | | | |  | | */Большакова-Стрекалова А.В./* | |
|  |  | | | | | *Подпись* | | *Фамилия И.О.* | |
|  | | | | |  | *05 октября 2025 г.* | | | | |  |
|  | | | | | |  | | | |

Калининград, 2025

Содержание

Введение

1 Разработка программного обеспечения

1.1 Анализ предметной области

1.2 Разработка и оформление технического задания

1.3 Построение архитектуры программного средства

1.4 Изучение работы в системе контроля версий

1.5 Построение диаграмм

1.6 Тестовый сценарий. Оценка программных средств с помощью метрик

1.7 Инспекция программного кода на предмет соответствия стандартам кодирования

2 Разработка АИС«Учет компьютерной техники и периферийных устройств»

2.1 Разработка структуры проекта

2.2 Разработка и интеграция модулей проекта

2.3 Отладка отдельных модулей программного проекта

2.4 Организация обработки исключений

2.5 Тестирование интерфейса пользователя средствами инструментальной среды разработки

2.6 Функциональное тестирование и тестирование интеграции

3 Моделирование в программных системах

Заключение

Список использованных источников

Приложение А. Текст программы

Приложение Б. Тест-кейсы и отчет о тестировании

Введение

Отчет представляет собой документально оформленный результат прохождения учебной практики по профессиональному модулю ПМ.02 «Осуществление интеграции программных модулей». Практика проводилась с целью формирования практических навыков и компетенций в области разработки комплексных веб-решений. Актуальность выбранной темы обусловлена повсеместной цифровизацией бизнес-процессов и критической важностью эффективного управления материальными активами организаций, в частности, парком компьютерной техники и периферийных устройств.

В современных условиях средние и крупные предприятия сталкиваются с существенными трудностями при ведении учета IT-оборудования традиционными методами. Использование электронных таблиц для инвентаризации сотен и тысяч единиц техники приводит к неизбежным ошибкам дублирования, затрудняет оперативный доступ к актуальной информации о местоположении и состоянии активов, усложняет процессы закрепления ответственности и проведения плановых инвентаризаций. Это влечет за собой прямые финансовые потери, неэффективное использование ресурсов и снижение общей производительности труда сотрудников IT-отдела и бухгалтерии.

Целью практики являлась разработка полнофункционального прототипа Автоматизированной Информационной Системы (АИС) «Компьютерная техника» в виде веб-приложения, предназначенной для комплексного решения перечисленных проблем. Система призвана обеспечить сквозную автоматизацию всего жизненного цикла IT-активов — от момента поступления на склад до окончательного списания.

Для достижения поставленной цели были последовательно решены следующие задачи:

1. Проведен всесторонний анализ предметной области, включая интервьюирование потенциальных пользователей системы (сотрудников IT-отдела, бухгалтеров, руководителей подразделений) для выявления и формализации всех ключевых бизнес-процессов.
2. На основе проведенного анализа разработано и оформлено в соответствии с ГОСТ 34.602-89 техническое задание на создание системы, являющееся основным документом, регламентирующим разработку.
3. Спроектирована масштабируемая и отказоустойчивая архитектура веб-приложения, произведен обоснованный выбор технологического стека, фреймворков и инструментов разработки.
4. Освоены и применены на практике современные методы коллективной разработки, включая работу с системой контроля версий Git и хостингом проектов GitHub, что является отраслевым стандартом.
5. Построен комплект визуальных моделей и диаграмм (UML, ER) для однозначного описания структуры, поведения и данных системы на разных уровнях абстракции.
6. Реализованы и проинтегрированы между собой ключевые функциональные модули системы, обеспечивающие базовый учет и управление оборудованием через веб-интерфейс.
7. Изучены и применены различные методы и уровни тестирования программного обеспечения (модульное, интеграционное, системное) для обеспечения высокого качества и надежности продукта.
8. Проведена оценка качества разрабатываемого программного средства с использованием стандартизированных метрик и методов экспертной оценки.
9. Исследованы и применены на практике основы программной инженерии, включая инспекцию кода, рефакторинг и организацию обработки исключений в контексте веб-разработки.

Объектом разработки выступила Автоматизированная Информационная Система (АИС) «Компьютерная техника». Данная система представляет собой веб-ориентированное корпоративное приложение, предназначенное для сотрудников IT-отдела, бухгалтерии и руководителей подразделений. Ее основная функция — централизованный учет, контроль и аналитика всех перемещений, состояний и финансовых аспектов, связанных с IT-активами организации через удобный веб-интерфейс.

Практическая значимость работы заключается в том, что разработанный прототип системы может быть взят за основу и доработан для внедрения в реальных условиях предприятия. Это позволит существенно повысить эффективность управления IT-инфраструктурой, снизить операционные издержки и минимизировать риски, связанные с потерей или некорректным учетом дорогостоящего оборудования.

1 Разработка программного обеспечения

1.1 Анализ предметной области

Управление парком компьютерной техники и периферийных устройств представляет собой сложную и многогранную предметную область, характеризующуюся большими объемами данных, высокой частотой изменений и строгими требованиями к отчетности. В процессе анализа был проведен детальный опрос потенциальных пользователей системы — системных администраторов, инженеров службы поддержки и бухгалтеров по основным средствам. Это позволило выявить ключевые проблемные точки существующего, преимущественно ручного, процесса учета.

Основные выявленные проблемы:

Дублирование и противоречивость данных: Одна и та же единица оборудования могла быть записана в разных Excel-файлах, ведущихся разными сотрудниками, с различными атрибутами.

Отсутствие единого источника истины: Не существовало централизованного места, где можно было бы оперативно получить полную и достоверную информацию о конкретном устройстве — его историю перемещений, ремонтов, текущего владельца.

Трудоемкость процессов: Процедуры закрепления оборудования за сотрудником, его перемещения между отделами или списания требовали оформления бумажных актов и последующего ручного внесения изменений в несколько журналов учета.

Сложность формирования отчетности: Подготовка отчетов для руководства или проверяющих органов (например, инвентарной описи) занимала значительное время и часто сопровождалась ошибками.

Отсутствие контроля за гарантийными сроками: Не было автоматического уведомления о приближающемся окончании гарантии на оборудование, что приводило к необоснованным расходам на платный ремонт.

На основе выявленных проблем были формализованы ключевые бизнес-процессы, подлежащие автоматизации в рамках АИС:

1. Процесс постановки на учет нового оборудования: Данный процесс инициируется поступлением техники на склад. В систему вносится полный перечень атрибутов: инвентарный номер (уникальный внутренний идентификатор), серийный номер (уникальный идентификатор производителя), наименование, модель, тип (отношение к классификатору), производитель, дата принятия к учету, первоначальная стоимость, гарантийный срок, сканы сопроводительных документов (накладная, гарантийный талон). Процесс включает проверку на дублирование по серийному и инвентарному номеру.
2. Процесс закрепления оборудования за материально-ответственным лицом (МОЛ): Это формализованный рабочий процесс, начинающийся с заявки от руководителя подразделения. После согласования с IT-отделом создается электронный акт приема-передачи, который визируется электронными подписями ответственных сторон. Система фиксирует дату передачи, предыдущего и нового владельца, основание для передачи. Оборудованию присваивается статус «В эксплуатации».
3. Процесс внутреннего перемещения оборудования: Фиксирует изменение физического местоположения актива без смены материально-ответственного лица (например, перемещение принтера в другой кабинет в рамках одного отдела). Также требует фиксации даты, причины и инициатора перемещения.
4. Процесс учета неисправностей и ремонтов: Сотрудник, столкнувшийся с неисправностью, создает в системе заявку с описанием проблемы. Заявка проходит стадии: «Зарегистрирована», «Назначена исполнителю», «В работе», «Выполнена», «Закрыта». Фиксируется исполнитель (внутренний специалист или сторонний сервисный центр), выполненные работы, затраченные запчасти и стоимость ремонта. Статус оборудования автоматически меняется на «В ремонте».
5. Процесс списания оборудования: Инициируется при невозможности или экономической нецелесообразности восстановления актива. Процесс включает создание комиссии, проведение экспертизы, оформление электронного акта на списание с указанием причины (моральный или физический износ, невозможность ремонта). После списание актив перемещается в архив, исключается из текущей описи, но его полная история сохраняется для отчетности.
6. Процесс формирования отчетности и аналитики: Система должна предоставлять как стандартные, регламентированные отчеты, так и возможность построения пользовательских. К стандартным относятся: полная инвентарная опись на дату, отчет по оборудованию в разрезе подразделений и МОЛ, карточка движения оборудования, отчет по затратам на ремонты за период, анализ списанной техники.

Для визуального представления структуры данных, подлежащих хранению в системе, была построена ER-модель (Entity-Relationship model). Эта модель является концептуальной и описывает сущности, их атрибуты и связи между ними без привязки к конкретной СУБД.

Диаграмма 1: ER-модель предметной области



Выделенные сущности:

Оборудование (Equipment): Центральная сущность. Атрибуты: ID, Инвентарный номер (уникальный), Серийный номер (уникальный), Наименование, Модель, Дата ввода в эксплуатацию, Первоначальная стоимость, Текущая стоимость (с учетом амортизации), Статус (На складе/В эксплуатации/В ремонте/Списан), Ссылка на фото.

Сотрудник (Employee): Атрибуты: ID, Табельный номер, ФИО, Должность, E-mail, Внутренний телефон, Активен (флаг увольнения).

Подразделение (Department): Атрибуты: ID, Наименование, Код подразделения, Руководитель (ссылка на Сотрудника).

Тип оборудования (EquipmentType): Справочник. Атрибуты: ID, Наименование (Ноутбук, Монитор, Принтер, МФУ, Сервер...), Код.

Производитель (Manufacturer): Справочник. Атрибуты: ID, Наименование (Dell, HP, Lenovo...).

Заявка на ремонт (RepairRequest): Атрибуты: ID, Дата создания, Описание проблемы, Статус заявки, Дата выполнения, Стоимость ремонта, Исполнитель.

Связи между сущностями:

Сотрудник принадлежит одному Подразделению (связь М:1).  
Оборудование относится к одному Типу и одному Производителю (связь М:1).  
Оборудование может быть закреплено за одним Сотрудником (М:1), при этом хранится история закреплений.  
Заявка на ремонт связана с одним Экземпляром Оборудования и одним Сотрудником-заявителем (М:1).

1.2 Разработка и оформление технического задания

Техническое задание (ТЗ) является фундаментальным документом в процессе разработки любого программного обеспечения. Оно служит официальным соглашением между заказчиком и исполнителем, однозначно фиксируя цели, границы, функционал и ограничения будущей системы. Разработка ТЗ для АИС «Компьютерная техника» осуществлялась в строгом соответствии с требованиями ГОСТ 34.602-89 «Техническое задание на создание автоматизированной системы».

Структура и содержание технического задания:

1. Общие сведения:

Полное наименование системы: Автоматизированная информационная система учета компьютерной техники и периферийных устройств «Компьютерная техника».

Условное обозначение: АИС «КТ».

Основание для разработки: Задание на учебную практику по ПМ.02.

Наименования предприятий Заказчика и Разработчика: В рамках учебного проекта заказчик и разработчик — студент/учебная группа.

Плановые сроки начала и окончания работы: Устанавливаются календарным планом.

1. Назначение и цели создания системы:

Назначение: Автоматизация процессов оперативного и бухгалтерского учета, контроля движения и технического состояния компьютерной техники и периферийных устройств в организации через веб-интерфейс.

Цели создания системы:

Повышение оперативности и достоверности учета IT-активов.

Сокращение времени на поиск информации об оборудовании и его истории.

Формализация и ускорение процессов закрепления, перемещения и списания.

Снижение потерь от несанционированного перемещения, простоя или выхода из строя оборудования.

Обеспечение прозрачности процессов для руководства и контролирующих органов.

Получение аналитической информации для планирования закупок и бюджетирования.

1. Характеристики объекта автоматизации: Дано описание существующей (неавтоматизированной) системы учета, выделены ее недостатки: низкая скорость обработки запросов, высокая вероятность ошибок, рассогласованность данных.

Требования к системе:

Требования к системе в целом:

Требования к структуре и функционированию системы: Система должна быть построена по модульному принципу. Выделены следующие подсистемы: Подсистема управления справочной информацией, Подсистема учета оборудования, Подсистема управления ремонтами, Подсистема отчетности и аналитики.

Требования к численности и квалификации персонала: Определены роли пользователей: Администратор системы, Менеджер IT-отдела, Сотрудник-пользователь. Для каждой роли описаны права доступа.

Требования к надежности: Требуется обеспечить целостность данных при сбоях электропитания, регулярное резервное копирование информации. Время наработки на отказ — не менее 1000 часов.

Требования к безопасности: Обязательная аутентификация пользователей по логину и паролю. Пароли должны храниться в хэшированном виде. Реализация разграничения прав доступа на основе ролей (RBAC).

Требования к эргономике и технической эстетике: Веб-интерфейс должен быть интуитивно понятным, соответствовать принципам юзабилити. Обязательна адаптивная верстка для корректного отображения на планшетах.

Требования к транспортабельности: Система должна развертываться на стандартных серверах под управлением ОС семейства Linux или Windows.

Требования к функциям (задачам), выполняемым системой: Детально расписаны функции для каждой подсистемы. Например, для Подсистемы учета оборудования:

Функция 1.1: Ввод и редактирование данных об оборудовании через веб-формы.

Функция 1.2: Просмотр карточки оборудования с полной историей в браузере.Функция 1.3: Закрепление оборудования за сотрудником через веб-интерфейс.

Функция 1.4: Перемещение оборудования между подразделениями.

Функция 1.5: Изменение статуса оборудования.

Требования к видам обеспечения:

Техническое обеспечение: Определены минимальная и рекомендуемая конфигурации рабочих мест пользователей (процессор, ОЗУ, разрешение экрана, браузер). Серверная часть: сервер с 4+ ядрами CPU, 8+ ГБ ОЗУ, 100+ ГБ HDD.

Программное обеспечение: Сервер: ОС Ubuntu Server 20.04 LTS, СУБД MySQL 5.7+, PHP 7.4+, Apache/Nginx. Клиент: любой современный браузер (Chrome 90+, Firefox 85+, Safari 14+).

Информационное обеспечение: Описаны структура и состав входной и выходной информации. Форматы обмена данными: HTML, JSON для API. Возможность импорта начальных данных из CSV-файлов.

Лингвистическое обеспечение: Интерфейс системы — русскоязычный.

Организационное обеспечение: Даны рекомендации по организационной структуре подразделения, эксплуатирующего систему.

1. Состав и содержание работ по созданию системы: Разработка разбита на этапы: Предпроектное обследование, Техническое проектирование, Рабочее проектирование, Внедрение и Сопровождение. Для каждого этапа определены сроки и основные результаты.
2. Порядок контроля и приемки: Описана процедура приемо-сдаточных испытаний. Определен состав приемочной комиссии. Критерием приемки является успешное прохождение всех тестовых сценариев, соответствующих требованиям ТЗ.
3. Требования к документированию: Приведен перечень разрабатываемой эксплуатационной документации: Техническое задание, Руководство системного администратора, Руководство пользователя, Программа и методика испытаний, Описание применения.
4. Источники разработки: Список использованных при разработке ТЗ стандартов (ГОСТ 34.602-89, ГОСТ 19.ххх, ГОСТ 2.ххх), литературных источников и анализов существующих систем-аналогов.

Данное ТЗ является исчерпывающим документом, который исключает двусмысленное толкование требований и служит основой для планирования и контроля всех последующих работ по проекту.

1.3 Построение архитектуры программного средства

Для АИС «Компьютерная техника» была выбрана классическая многоуровневая архитектура веб-приложения с использованием PHP, MySQL и шаблоном Model-View-Controller (MVC) в упрощенной реализации. Данный подход обеспечивает надежность, простоту развертывания и соответствует требованиям корпоративных систем учета.

Детальное обоснование выбора архитектуры:

Простота и надежность: PHP как серверный язык обеспечивает стабильную работу и простоту развертывания на большинстве хостинг-провайдеров. MySQL как реляционная СУБД обеспечивает надежное хранение данных и эффективное выполнение запросов.

Быстрая разработка: Использование чистого PHP без сложных фреймворков ускоряет процесс разработки и упрощает поддержку. Прямая работа с SQL-запросами позволяет оптимизировать взаимодействие с базой данных.

Безопасность: Встроенные механизмы PHP и PDO обеспечивают защиту от SQL-инъекций и других веб-угроз. Возможность реализации механизмов аутентификации и авторизации непосредственно в коде.

Совместимость: Система работает на стандартных веб-серверах (Apache/Nginx) и не требует специального окружения. Кросс-браузерная совместимость обеспечивается использованием стандартов HTML5 и CSS3.

Детальное описание архитектурных уровней:

1. Уровень представления (View):

HTML5: Семантическая разметка страниц

CSS3: Адаптивный дизайн с использованием Grid и Flexbox

JavaScript: Клиентская валидация и интерактивность

Пример: файлы index.php, view\_equipment.php, add\_equipment.php

1. Уровень бизнес-логики (Controller):

PHP скрипты: Обработка HTTP запросов и маршрутизация

Валидация данных: Проверка входных параметров на стороне сервера

Управление сессиями: Контроль доступа и аутентификация пользователей

Пример: add\_equipment.php, add\_employee.php, db\_connect.php

1. Уровень данных (Model):

MySQL 5.7+: Реляционная база данных

PDO (PHP Data Objects): Безопасный доступ к БД

Нормализованная схема: Таблицы equipment, employees, departments, equipment\_types, manufacturers

Пример: db\_connect.php, SQL-запросы в PHP-скриптах

Технологический стек:

Серверный язык: PHP 7.4+

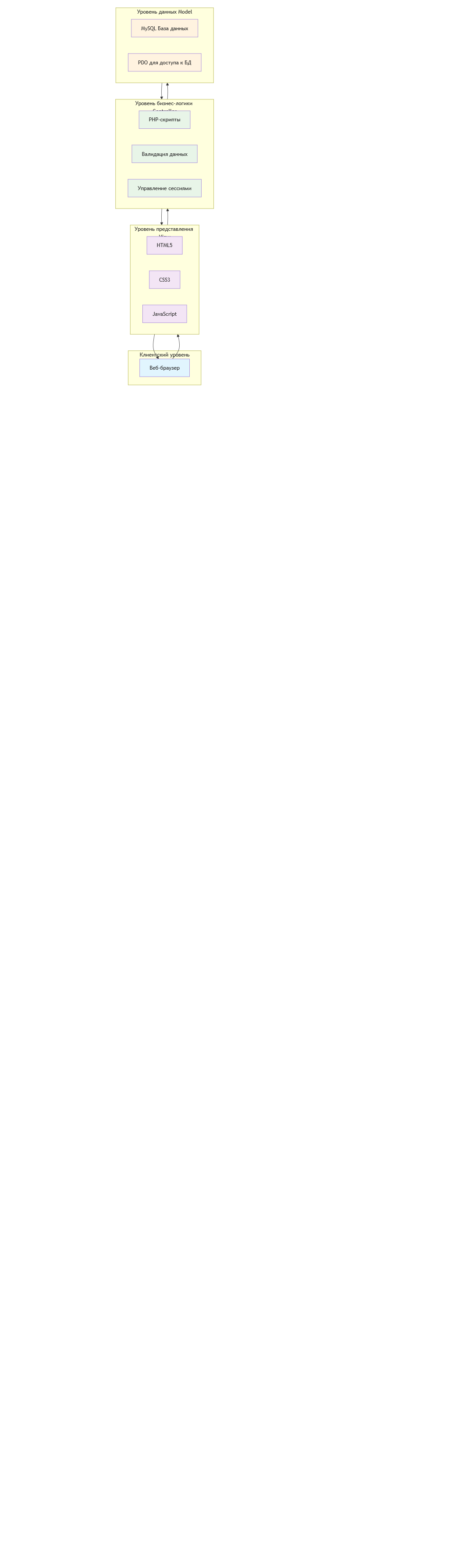
База данных: MySQL 5.7+

Веб-сервер: Apache с mod\_php или Nginx с PHP-FPM

Клиентские технологии: HTML5, CSS3, JavaScript

Библиотеки: PDO для работы с БД

Диаграмма 2: Архитектура АИС на PHP



1.4 Изучение работы в системе контроля версий

В современной разработке программного обеспечения, особенно в условиях командной работы, использование систем контроля версий (Version Control Systems, VCS) перешло из разряда рекомендаций в категорию обязательных требований. Эти системы представляют собой фундаментальный инструмент, позволяющий отслеживать все изменения в исходном коде, координировать работу нескольких разработчиков над одним проектом, оперативно откатывать ошибочные изменения, экспериментировать с новым функционалом в изолированных средах и эффективно управлять выпусками версий продукта.

Для управления исходным кодом проекта АИС «Компьютерная техника» была выбрана распределенная система контроля версий Git в связке с облачным хостингом репозиториев GitHub. Данный выбор был сделан после сравнительного анализа существующих на рынке решений, включая централизованные системы (такие как Apache Subversion - SVN) и другие распределенные системы (например, Mercurial).

Детальное обоснование выбора Git и GitHub:

Git как система контроля версий:

Распределенная архитектура: В отличие от централизованных систем (например, SVN), где существует единый главный репозиторий на сервере, в Git каждый разработчик имеет на своем локальном компьютере полную копию всего репозитория со всей его историей изменений. Это предоставляет ряд неоспоримых преимуществ:

Автономность: Разработчики могут выполнять большинство операций (просмотр истории, создание коммитов, переключение между ветками) локально, без постоянного подключения к центральному серверу. Это значительно повышает скорость работы.

Отказоустойчивость: Потеря центрального сервера не означает потерю истории проекта, так как она содержится в каждой полной копии репозитория. Любая из таких копий может быть использована для восстановления.

Гибкость workflow: Разработчики могут легко обмениваться изменениями между собой напрямую, прежде чем интегрировать их в общий репозиторий.

Производительность: Операции в Git (коммит, просмотр истории, ветвление) выполняются локально и поэтому исключительно быстры, поскольку не требуют сетевого взаимодействия.

Мощное ветвление и слияние: Git предоставляет исключительно удобные, легковесные и эффективные механизмы для создания веток (branches). Ветка в Git — это просто указатель на определенный коммит. Это позволяет создавать десятки изолированных линий разработки для реализации новых функций (feature branches), исправления ошибок (hotfix branches) или подготовки релизов (release branches) без какого-либо значительного overhead или вмешательства в основную стабильную версию кода. Процесс слияния (merge) веток в Git также очень развит и в большинстве случаев автоматизирован.

Целостность данных: Все содержимое в Git (файлы, каталоги, коммиты, теги) идентифицируется своим криптографическим хэшем (SHA-1). Это гарантирует, что история проекта не может быть изменена без отслеживания этих изменений. Если какой-либо файл или коммит будет изменен, его хэш изменится, и система это immediately обнаружит.

Гибкость и широкая поддержка: Git интегрируется с большинством современных сред разработки (IDE), таких как PhpStorm, VS Code, а также с системами непрерывной интеграции и доставки (CI/CD), например, Jenkins, GitLab CI/CD, GitHub Actions.

GitHub как платформа для хостинга и collaboration:

Удобный веб-интерфейс: Предоставляет интуитивно понятный интерфейс для просмотра кода, истории коммитов, различий между версиями (diff), визуализации сети коммитов и веток.

Инструменты для код-ревью (Pull Requests): Это ключевой инструмент для поддержания качества кода в командной разработке. Pull Request (PR) — это предложение изменений в код. Перед тем как эти изменения будут слиты в основную ветку (например, main или develop), они проходят процесс обсуждения, проверки и тестирования другими членами команды. Любой разработчик может оставить комментарий к конкретной строке кода, предложить улучшения, запросить доработки. Это не только повышает качество кода, но и способствует обмену знаниями внутри команды.

Система Issues: Используется для отслеживания задач (tasks), багов (bugs), новых идей (enhancements). Позволяет организовать планирование работ, назначать исполнителей, привязывать задачи к milestones и версиям продукта.

Project Boards (Канбан-доски): Визуальный инструмент для управления workflow команды, аналогичный Trello или Jira. Позволяет перемещать задачи по колонкам (To Do, In Progress, Code Review, Done), что обеспечивает прозрачность процесса разработки для всех участников.

GitHub Actions: Мощная встроенная система CI/CD для автоматизации процессов сборки, тестирования и развертывания. Позволяет описывать pipeline'ы непосредственно в коде репозитория в виде YAML-файлов. Например, можно настроить автоматический запуск всех тестов при каждом push в репозиторий или автоматическое развертывание на тестовый сервер при мерже PR в ветку develop.

В рамках практики был создан приватный репозиторий на GitHub и настроена его начальная структура. Была изучена и применена на практике популярная модель ветвления GitFlow (в несколько упрощенном варианте). Основные ветки в репозитории:

main (или master): Содержит только стабильный, готовый к production-развертыванию код. В эту ветку попадают только проверенные и протестированные изменения, обычно через слияние из develop после подготовки релиза.

develop: Основная ветка для интеграции новых функций. Именно в эту ветку сливаются feature-ветки после успешного код-ревью и прохождения тестов. Код в develop всегда должен находиться в работоспособном состоянии.

feature/\*: Ветки, создаваемые от develop для разработки новой функциональности. Наименование ветки должно быть описательным, например, feature/add-equipment-crud или feature/implement-repair-requests.

hotfix/\*: Ветки для срочного исправления критических багов в production-версии. Создаются от main, и после исправления мержатся обратно в main и develop.

Были отработаны основные команды Git как в командной строке, так и через интегрированные возможности IDE:

git clone [url] -- клонирование удаленного репозитория на локальную машину.

git checkout -b feature/[name] -- создание и переключение на новую ветку для разработки фичи.

git status -- просмотр состояния рабочей директории и индекса.

git add [files] -- добавление измененных файлов в область подготовленных файлов (staging area).

git commit -m "message" -- фиксация изменений в локальном репозитории с описательным сообщением, которое должно четко объяснять, что и зачем было сделано.

git push origin [branch-name] -- отправка локальных коммитов в удаленный репозиторий на GitHub.

git pull origin [branch-name] -- получение последних изменений из удаленной ветки и их слияние с локальной веткой.

git merge [branch-name] -- слияние указанной ветки с текущей.

Освоение Git и GitHub, включая практику регулярного код-ревью через механизм Pull Requests, является одним из наиболее важных и востребованных практических навыков, полученных за время практики, так как это основа любого современного промышленного процесса разработки ПО.

1.5 Построение диаграмм

Для визуального проектирования системы, обеспечения однозначного понимания ее структуры, поведения и динамики всеми участниками проекта (архитекторами, разработчиками, тестировщиками, заказчиками, аналитиками) был построен комплекс диаграмм на унифицированном языке моделирования UML (Unified Modeling Language). UML предоставляет стандартизированные, интуитивно понятные нотации для описания различных аспектов системы с разных точек зрения.

Диаграмма вариантов использования (Use Case Diagram):  
Эта диаграмма определяет границы системы и взаимодействие с внешними акторами. Она отвечает на вопрос «Что система делает для актора?», фокусируясь на функциональных требованиях высокого уровня, без углубления в внутреннюю реализацию.

Акторы (Actors):

Администратор системы: Отвечает за управление пользователями, настройку справочников (типы оборудования, производители), общее администрирование системы.

Менеджер IT-отдела: Ключевой пользователь. Выполняет операции по учету оборудования: добавление, редактирование, закрепление за сотрудниками, регистрация заявок на ремонт, списание.

Сотрудник-пользователь: Просматривает закрепленное за ним оборудование, подает заявки на ремонт через веб-интерфейс.

Прецеденты (Use Cases):

Для Администратора: «Управление пользователями», «Управление справочниками», «Настройка системы».

Для Менеджера IT-отдела: «Добавить оборудование», «Редактировать оборудование», «Удалить оборудование», «Закрепить оборудование за сотрудником», «Зарегистрировать заявку на ремонт», «Изменить статус заявки», «Сформировать отчет».

Для Сотрудника-пользователя: «Просмотр закрепленного оборудования», «Подача заявки на ремонт».

Диаграмма 3: Диаграмма вариантов использования

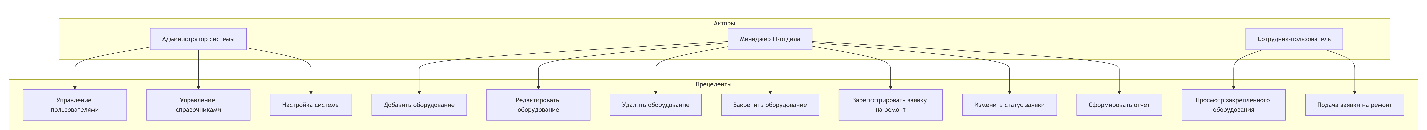


Диаграмма последовательности (Sequence Diagram):  
Эта диаграмма детализирует взаимодействие между объектами системы в рамках конкретного сценария выполнения прецедента. Она показывает поток сообщений, которыми обмениваются объекты, и их порядок во времени. Диаграмма отвечает на вопрос «Как система реализует конкретный сценарий?».

Была построена для критически важного сценария «Добавление нового оборудования».

Участники (Lifelines): Пользователь (веб-браузер), Веб-сервер (Apache/PHP), База данных (MySQL).

Поток сообщений (Message Flow):

1. Пользователь заполняет форму добавления оборудования и нажимает кнопку «Сохранить».
2. Браузер отправляет HTTP POST-запрос с данными формы на сервер (add\_equipment.php).
3. Веб-сервер принимает запрос и запускает выполнение PHP-скрипта.
4. PHP-скрипт выполняет валидацию входных данных (проверка обязательных полей, форматов данных).
5. Если валидация неуспешна, возвращается ответ с ошибками валидации.
6. Если валидация успешна, PHP-скрипт подключается к базе данных через PDO.
7. PHP-скрипт выполняет SQL-запрос INSERT для добавления записи в таблицу equipment.
8. База данных выполняет запрос и возвращает результат (успех или ошибка).
9. PHP-скрипт обрабатывает ответ от базы данных.
10. Если запрос выполнен успешно, PHP-скрипт формирует HTTP-ответ с подтверждением успешного добавления.
11. Если произошла ошибка (например, нарушение уникальности инвентарного номера), PHP-скрипт формирует HTTP-ответ с сообщением об ошибке.
12. Веб-сервер отправляет HTTP-ответ браузеру.
13. Браузер отображает пользователю сообщение о результате операции.

Диаграмма 4: Диаграмма последовательности для прецедента «Добавление оборудования»

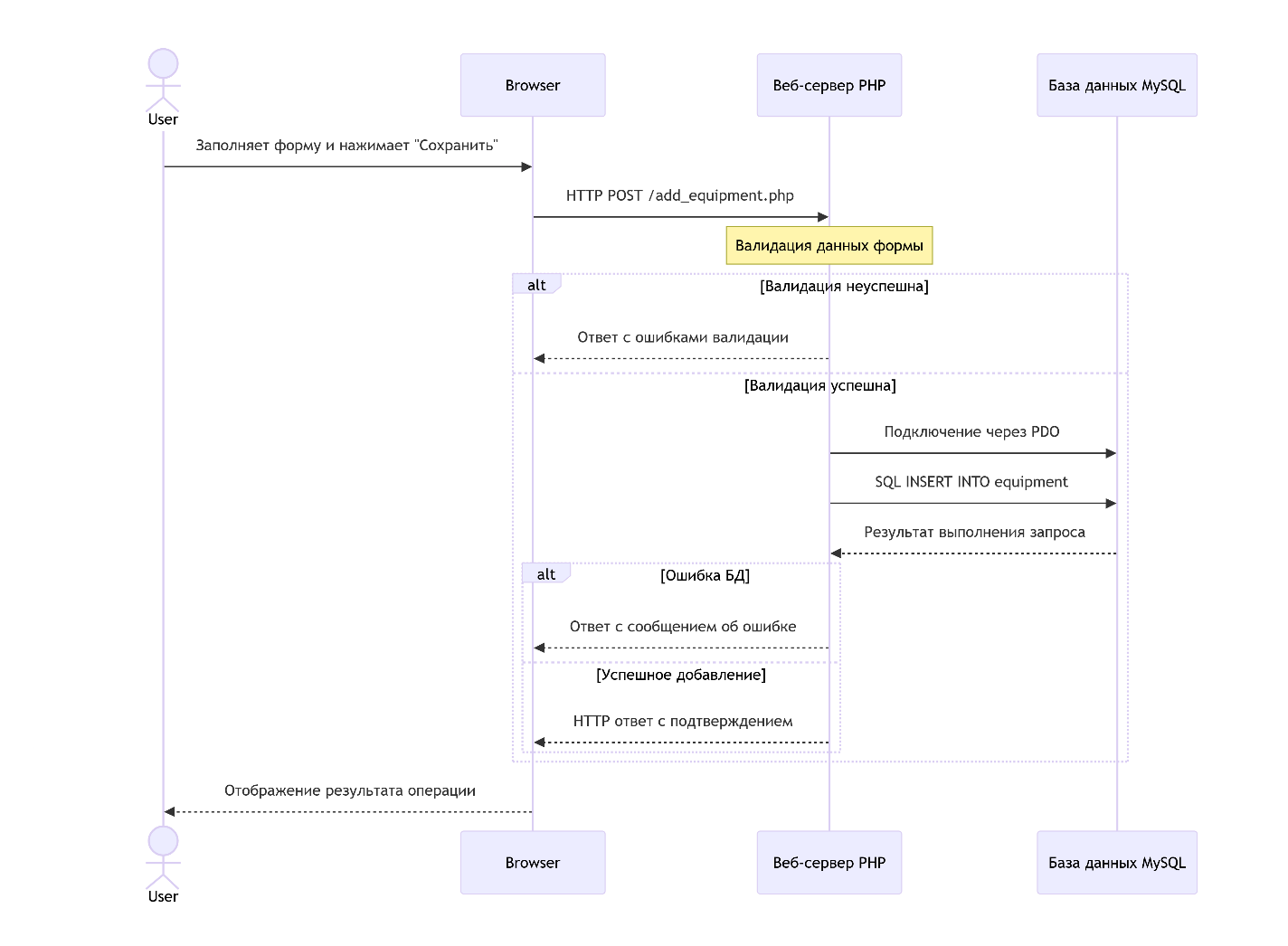


Диаграмма классов (Class Diagram):  
Эта диаграмма отображает статическую структурную схему системы в терминах классов, их атрибутов, методов и взаимосвязей между ними (ассоциация, агрегация, композиция, наследование). Эта диаграмма является прямой основой для проектирования схемы базы данных и написания кода.

Ключевые классы:

Employee: Атрибуты: id, employee\_number, first\_name, last\_name, middle\_name, position, email, phone, department\_id. Методы: getFullName().

Department: Атрибуты: id, name, code, manager\_id.

Equipment: Атрибуты: id, inventory\_number, serial\_number, name, model, equipment\_type\_id, manufacturer\_id, purchase\_date, initial\_cost, status, current\_owner\_id. Методы: calculateDepreciation().

EquipmentType: Атрибуты: id, name, code.

Manufacturer: Атрибуты: id, name.

RepairRequest: Атрибуты: id, equipment\_id, reporter\_id, description, status, created\_at, completed\_at, repair\_cost, executor.

Диаграмма 5: Диаграмма классов

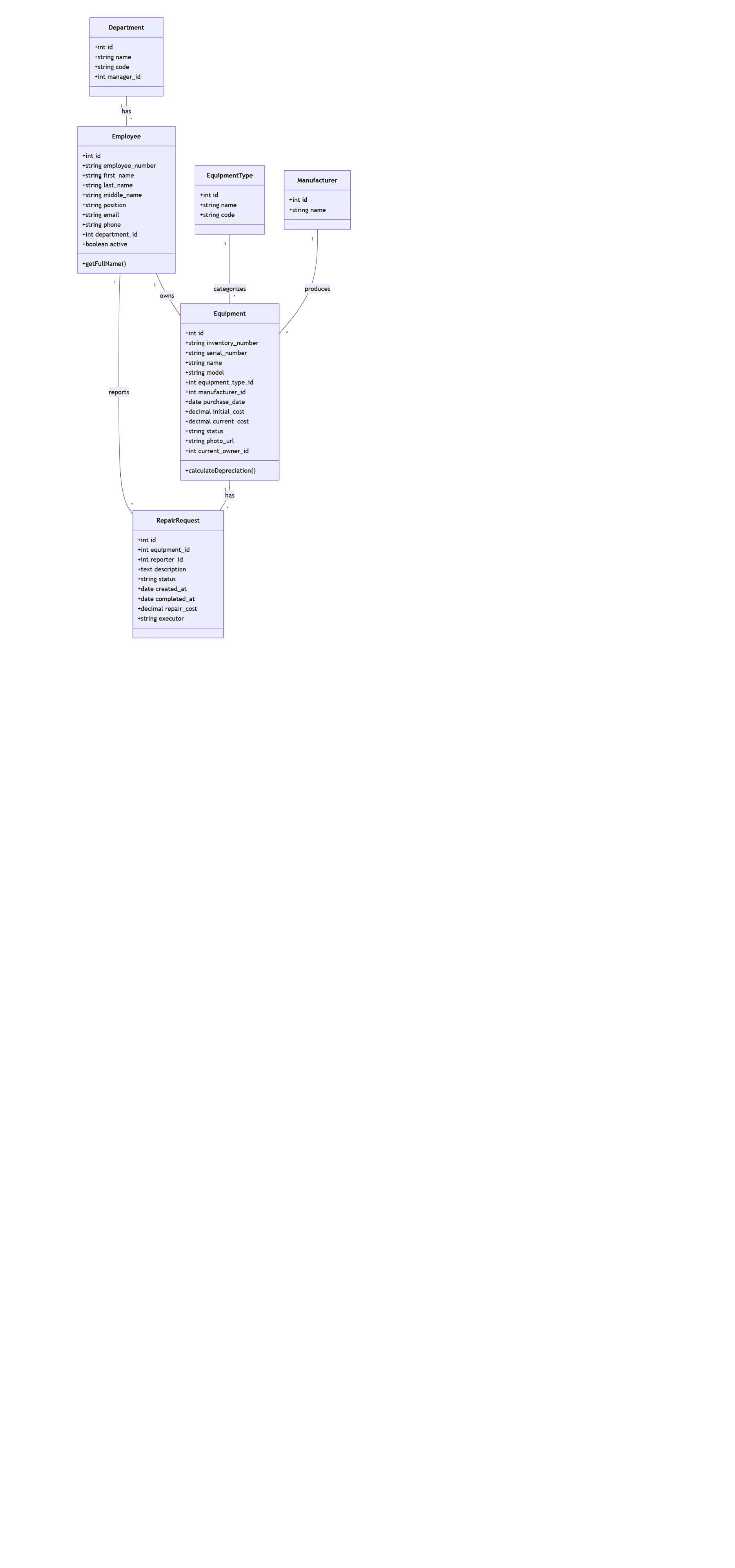


Диаграмма развертывания (Deployment Diagram):  
Эта диаграмма показывает физическое размещение компонентов системы по узлам (нодам) и конфигурацию аппаратного/программного обеспечения. Она отвечает на вопрос «На каком «железе» и ПО работает система?».

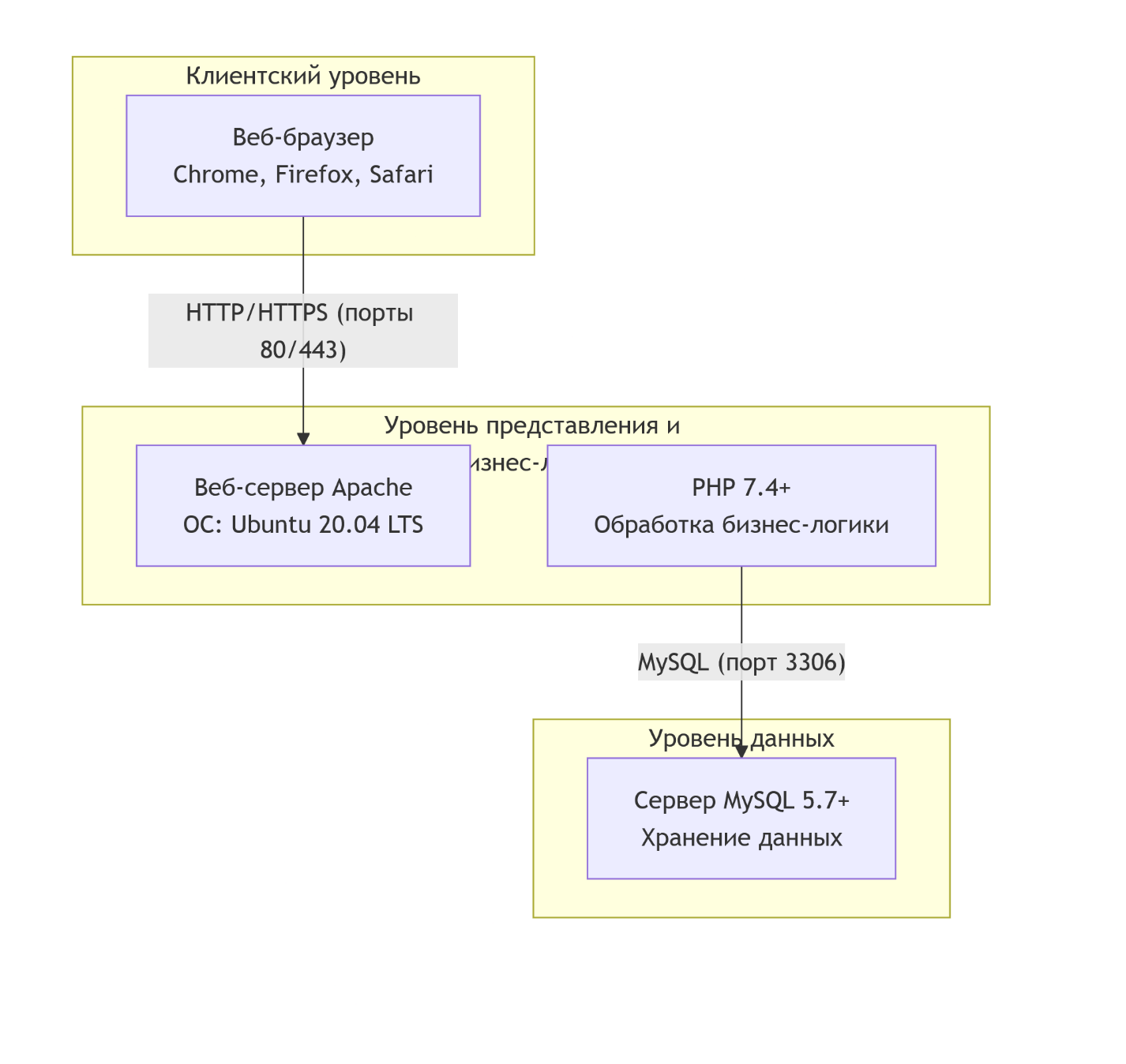
Узел «Клиентская рабочая станция»: Атрибуты: Веб-браузер (Chrome, Firefox, Safari). Развернут артефакт: Веб-приложение (HTML, CSS, JavaScript).

Узел «Веб-сервер»: Атрибуты: Виртуальный/физический сервер с ОС Linux (Ubuntu 20.04 LTS). Развернутые компоненты: Apache веб-сервер, PHP 7.4+.

Узел «Сервер базы данных»: Атрибуты: Виртуальный/физический сервер. Развернутый компонент: MySQL 5.7+ (СУБД).

Связи между узлами: Показывают сетевые взаимодействия: HTTP/HTTPS (порт 80/443) между Клиентом и Веб-сервером, и специфичное соединение (порт 3306) между Веб-сервером (PHP-приложением) и Сервером БД.

Диаграмма 6: Диаграмма развертывания



Построение этого комплекта диаграмм не только служит целям документации, но и является мощным инструментом анализа, позволяющим выявить потенциальные проблемы проектирования, «узкие места» и неоднозначности на самых ранних стадиях, до начала наиболее дорогостоящей фазы — кодирования.

1.6 Тестовый сценарий. Оценка программных средств с помощью метрик

Оценка необходимого количества тестов

Для обеспечения полноты тестового покрытия был проведен детальный анализ функциональных требований и структуры программы. На основе методов комбинаторного тестирования и анализа граничных значений была выполнена оценка необходимого количества тестов:

Модульное тестирование (Unit Testing): ~40-60 тестов для покрытия основных модулей системы, включая:

15-20 тестов для PHP-скриптов (бизнес-логика)

10-15 тестов для валидации данных

10-15 тестов для работы с базой данных

5-10 тестов для вспомогательных функций

Интеграционное тестирование (Integration Testing): ~20-30 тестов для проверки взаимодействия между компонентами:

8-10 тестов интеграции веб-форм с PHP-обработчиками

6-8 тестов взаимодействия между модулями оборудования и сотрудников

6-8 тестов интеграции PHP-скриптов с базой данных

4-6 тестов работы системы аутентификации и авторизации

Системное тестирование (System Testing): ~15-20 end-to-end тестов для проверки полных бизнес-сценариев:

5-7 тестов основных workflow (добавление → закрепление → ремонт → списание)

4-5 тестов формирования отчетности

6-8 тестов работы с системой под различными ролями пользователей

Приемочное тестирование (Acceptance Testing): ~10-12 тестов для верификации соответствия требованиям ТЗ

Разработка тестовых пакетов

Для модуля управления оборудованием был разработан комплексный тестовый пакет, основанный на анализе потоков управления и данных. Для выбранного метода тестирования (комбинация критериев покрытия условий и решений) были обозначены ветви алгоритма и выписаны пути, которые должны быть проверены тестами.

Таблица 1: Тестовые пакеты для модуля управления оборудованием

| ID теста | Путь алгоритма | Ветви алгоритма | Входные данные | Ожидаемый результат | Фактический результат |
| --- | --- | --- | --- | --- | --- |
| T-EO-002 | П1-У1-У2-У3-У5 | В1-В3-В5-В7-В10 | Инв. № "INV-001" (существует), сер. № "SN002" | Ошибка валидации, HTTP 400, сообщение о дубликате | Соответствует |
| T-EO-003 | П1-У1-У2-У3-У6 | В1-В3-В5-В8-В11 | Инв. № "INV-002", сер. № "SN001" (существует) | Ошибка валидации, HTTP 400, сообщение о дубликате сер. № | Соответствует |
| T-EO-004 | П1-У1-У2-У4 | В1-В3-В6-В12 | Инв. № "INV-003", сер. № "SN003", стоимость -1000 | Ошибка валидации, HTTP 400, сообщение о недопустимой стоимости | Соответствует |
| T-EO-005 | П1-У1-У2-У4 | В1-В4-В13 | Только инвентарный номер, остальные поля пустые | Ошибка валидации, HTTP 400, список обязательных полей | Соответствует |

Обозначение ветвей алгоритма:

В1: Начало операции добавления оборудования  
В2: Начало операции поиска оборудования  
В3: Проверка формата инвентарного номера - УСПЕХ  
В4: Проверка формата инвентарного номера - ОШИБКА  
В5: Проверка уникальности инвентарного номера - УСПЕХ  
В6: Проверка уникальности инвентарного номера - ОШИБКА  
В7: Проверка уникальности серийного номера - УСПЕХ  
В8: Проверка уникальности серийного номера - ОШИБКА  
В9: Проверка стоимости - УСПЕХ  
В10: Сохранение оборудования - ОШИБКА (дубликат инв. №)  
В11: Сохранение оборудования - ОШИБКА (дубликат сер. №)  
В12: Проверка стоимости - ОШИБКА  
В13: Проверка обязательных полей - ОШИБКА  
В14: Поиск оборудования в БД - НЕ НАЙДЕНО  
В15: Возврат результата поиска - ОШИБКА

Метрика Холстеда

Для количественной оценки сложности программного кода модуля управления оборудованием была рассчитана метрика Холстеда. Анализ проводился на основе исходного кода PHP-скриптов (add\_equipment.php, view\_equipment.php, db\_connect.php).

*Исходные данные для расчета:*

n1 = 35 (число уникальных операторов)  
n2 = 62 (число уникальных операндов)  
N1 = 180 (общее число операторов)  
N2 = 410 (общее число операндов)

*Расчет метрик Холстеда:*

Словарь программы: n = n1 + n2 = 35 + 62 = 97  
Длина программы: N = N1 + N2 = 180 + 410 = 590  
Объем программы: V = N × log₂n = 590 × log₂97 ≈ 590 × 6.6 ≈ 3894  
Потенциальный объем: V\* = (2 + n2) × log₂(2 + n2) = (2 + 62) × log₂64 ≈ 64 × 6 ≈ 384  
Уровень программы: L = V\* / V = 384 / 3894 ≈ 0.098  
Сложность программы: D = 1 / L ≈ 10.2  
Уровень качества программирования: I = L × V = 0.098 × 3894 ≈ 381.6  
Оценка усилий программирования: E = V / L = 3894 / 0.098 ≈ 39734.7  
Время реализации: T = E / 18 ≈ 2207.5 секунд ≈ 36.8 минут

Полученные значения свидетельствуют о средней сложности программного кода. Относительно высокий уровень программы (L = 0.098) и умеренная сложность (D = 10.2) указывают на хорошую читаемость и сопровождаемость кода. Оценка усилий программирования соответствует ожиданиям для бизнес-логики средней сложности.

1.7 Инспекция программного кода на предмет соответствия стандартам кодирования

Одним из ключевых факторов, обеспечивающих долгосрочную успешность программного проекта, является читаемость, сопровождаемость и единообразие кодовой базы. Для достижения этого были приняты и строго соблюдаются соглашения по стилю кодирования (Coding Conventions), а также проводилась регулярная инспекция кода (Code Review) с использованием детальных контрольных списков.

Принятые стандарты кодирования и инструменты:

Для Back-end (PHP): Следование стандарту PSR-12 (PHP Standard Recommendations) является обязательным. Это включает:

Отступы: Использование 4 пробелов на каждый уровень отступа. Табы запрещены.

Максимальная длина строки: Ограничение в 120 символов для кода.

Именование:

camelCase для переменных и функций ($employeeList, calculateTotalCost).  
PascalCase для имен классов (EquipmentAssignmentService).  
UPPER\_CASE\_WITH\_UNDERSCORES для имен констант (MAX\_EQUIPMENT\_COUNT, STATUS\_CHOICES).

Пробелы и пустые строки: Строгое следование рекомендациям PSR-12 по использованию пробелов вокруг операторов, после запятых, а также для отделения логических блоков кода друг от друга.

Структура файлов: Все файлы должны использовать теги <?php без закрывающего тега ?> в чистых PHP-файлах для предотвращения случайного вывода пробелов.

Комментарии: Для всех сложных функций и классов используются комментарии в формате DocBlock, что позволяет автоматически генерировать документацию.

Для Front-end (HTML/CSS/JavaScript): Рекомендуется следование руководству по стилю от Google.

Именование: kebab-case для имен классов CSS, camelCase для переменных и функций JavaScript.

Линтинг и форматирование: Для автоматической проверки и приведения кода к единому стандарту используются:

ESLint: Статический анализатор кода для JavaScript для выявления проблемных шаблонов и обеспечения соблюдения стиля.  
Prettier: «Упрямый» форматировщик кода, который автоматически переформатирует код согласно заданным правилам, устраняя все споры о стиле.

Проведение инспекции кода (Code Review):

Инспекция проводилась с использованием двух детальных контрольных списков, основанных на лучших практиках программной инженерии. Проверка осуществлялась коллегами по команде через механизм Pull Requests (PR) на GitHub.

Контрольный список №1: «Именование переменных и функций»

Общие принципы: Имена четко, полно и однозначно описывают представляемую сущность или действие ($equipment\_list вместо $list1, assign\_employee\_to\_equipment вместо assign или aete).

Проблемная область: Имена характеризуют проблему реального мира, а не ее программную реализацию (RepairRequest, а не DataModel7; $current\_owner, а не $field\_27).

Длина имени: Имена имеют достаточную длину, чтобы не нужно было «ломать голову» над их значением, но при этом не являются избыточно длинными.

Спецификаторы: Спецификаторы вычисляемых значений находятся в конце имен ($equipment\_count, $total\_price, $is\_available).

Конкретные виды данных:

Булевы переменные названы так, чтобы был понятен смысл значений true/false ($is\_available, $has\_guarantee, $should\_update).

Для индексов циклов, если цикл не тривиален (более 2-3 строк или вложенные циклы), используются выразительные имена (foreach ($employees as $index => $employee)).

Избегаются однобуквенные имена, кроме тривиальных случаев ($i, $j в коротких циклах).

Сокращения: Избегаются неочевидные и нестандартные сокращения. Сокращения, если используются, согласованы по всему проекту и документированы.

Контрольный список №2: «Хорошие методики комментирования и структура кода»

Общие аспекты: Код стремится быть самодокументирующимся за счет хороших имен и четкой структуры. Комментарии объясняют «почему» реализовано именно так, а не «как» (если код не является очевидным). Например, комментарий, объясняющий нестандартный алгоритм или причину выбора конкретного метода решения.

Форматирование комментариев: Для документирования сложных функций и классов используются DocBlock комментарии. Однострочные комментарии используются экономно для пояснения неочевидных участков кода внутри функций.

Удаление избыточности: Удалены избыточные и неактуальные комментарии, которые просто повторяют код (например, // Increment i by 1 перед строкой $i += 1).

Структура кода (Рефакторинг): Длинные или сложные функции (более 30-40 строк) были рефакторены и разбиты на более мелкие, понятные подфункции, вместо того чтобы пытаться комментировать их сложную логику.

Объявления данных: В местах объявления данных, если это необходимо, указаны единицы измерения или диапазоны допустимых значений (в виде комментария или в DocBlock). Например, $initial\_cost = 0; // Стоимость в рублях.

Обработка ошибок и исключительных ситуаций: Код, предотвращающий ошибки или обрабатывающий исключительные ситуации, снабжен комментариями, поясняющими причину и контекст такой обработки.

Соблюдение стандартов кодирования и практика регулярной, тщательной инспекции кода через Pull Requests являются краеугольным камнем профессиональной разработки.

2 Разработка АИС «Компьютерная техника»

2.1 Разработка структуры проекта

Техническое задание по ГОСТ 19.101-77

В соответствии с требованиями Единой системы программной документации (ЕСПД) разработано техническое задание на создание АИС "Компьютерная техника".

1. Введение

1.1. Наименование программы: Автоматизированная информационная система "Компьютерная техника"  
1.2. Краткая характеристика области применения: Система предназначена для автоматизации процессов учета, контроля и управления компьютерной техникой и периферийными устройствами в организациях через веб-интерфейс.

1. Основание для разработки

2.1. Основание для проведения работ: Задание на учебную практику по профессиональному модулю ПМ.02 "Осуществление интеграции программных модулей"  
2.2. Наименование организации-заказчика: Учебное заведение  
2.3. Наименование организации-разработчика: Студент учебной группы

1. Назначение разработки

3.1. Функциональное назначение: Система предназначена для автоматизации следующих процессов:

Учет и инвентаризация компьютерной техники и периферийных устройств через веб-интерфейс  
Закрепление оборудования за материально-ответственными лицами  
Учет технического обслуживания и ремонтов  
Формирование отчетной документации  
Контроль перемещения оборудования между подразделениями

3.2. Эксплуатационное назначение: Система должна эксплуатироваться сотрудниками IT-отдела, бухгалтерии и руководителями подразделений организации через веб-браузер.

1. Требования к программе

4.1. Требования к функциональным характеристикам:

4.1.1. Состав выполняемых функций:

Ф01: Ведение базы данных оборудования через веб-интерфейс  
Ф02: Управление закреплением оборудования за сотрудниками  
Ф03: Регистрация заявок на ремонт и учет их выполнения  
Ф04: Учет перемещения оборудования между подразделениями  
Ф05: Формирование отчетных документов в веб-интерфейсе  
Ф06: Ведение справочников (типы оборудования, производители, подразделения)

4.1.2. Организация входных данных: Ввод данных осуществляется через веб-формы. Поддерживается как ручной ввод, так и импорт данных из CSV-файлов.

4.1.3. Организация выходных данных: Выходные данные представляются в виде электронных отчетов в форматах PDF и Excel, а также отображаются в веб-интерфейсе системы.

4.2. Требования к надежности:

4.2.1. Обеспечение надежного функционирования: Программа должна обеспечивать:

Целостность данных при сбоях электропитания  
Валидацию входных данных на стороне сервера  
Резервное копирование информации  
Восстановление после сбоев

4.3. Требования к техническому обеспечению:

4.3.1. Минимальный состав технических средств:

Процессор: Intel Core i3 или аналогичный  
ОЗУ: 4 ГБ  
Свободное место на HDD: 2 ГБ  
Монитор с разрешением 1280×1024  
Сетевая карта для доступа в LAN/Internet

4.4. Требования к программному обеспечению:

4.4.1. Программное обеспечение сервера:

ОС: Ubuntu Server 20.04 LTS или Windows Server 2019  
СУБД: MySQL 5.7+  
Веб-сервер: Apache 2.4+ с mod\_php или Nginx с PHP-FPM  
PHP 7.4+

4.4.2. Программное обеспечение клиента:

Браузер: Google Chrome 90+, Mozilla Firefox 85+, Safari 14+

1. Требования к программной документации

5.1. Состав программной документации:

Техническое задание  
Руководство системного администратора  
Руководство пользователя  
Программа и методика испытаний  
Пояснительная записка

5.2. Специальные требования: Вся документация должна быть представлена в электронном виде в формате PDF.

1. Технико-экономические показатели

Ожидаемый экономический эффект от внедрения системы достигается за счет:

Сокращения времени на поиск информации об оборудовании на 60-70%  
Уменьшения количества ошибок учета на 80-90%  
Сокращения времени формирования отчетности на 75-85%

1. Стадии и этапы разработки

7.1. Стадия эскизного проектирования:

Разработка технического задания  
Проектирование архитектуры системы

7.2. Стадия технического проектирования:

Разработка алгоритмов  
Проектирование базы данных  
Создание прототипа веб-интерфейса

7.3. Стадия рабочего проектирования:

Написание программного кода на PHP  
Тестирование модулей  
Интеграция компонентов

7.4. Стадия внедрения:

Опытная эксплуатация  
Обучение пользователей  
Сдача в промышленную эксплуатацию

2.2 Разработка и интеграция модулей проекта

Процесс разработки функциональных модулей системы осуществлялся по методологии инкрементальной сборки, когда каждый модуль проектировался, реализовывался и тестировался независимо, после чего интегрировался в общую архитектуру.

Модуль учета оборудования был спроектирован как центральный компонент системы. Его модель данных включает сущность Equipment с комплексом атрибутов, точно соответствующих требованиям предметной области. Для обеспечения целостности данных в базе данных реализованы ограничения уникальности инвентарных и серийных номеров, проверки форматов дат и числовых значений. Бизнес-логика инкапсулирована в PHP-скриптах, таких как расчет амортизации оборудования и автоматическое обновление статусов.

Веб-интерфейс модуля реализован через HTML-формы с валидацией на стороне клиента (JavaScript) и сервера (PHP). Скрипты обработки данных выполняют не только преобразование данных между форматами, но и сложную валидацию, включая проверку зависимостей между полями. Логика работы с базой данных вынесена в отдельный файл db\_connect.php, который обеспечивает безопасное подключение к MySQL через PDO.

Модуль управления сотрудниками разработан с учетом требований к учету материальной ответственности. Модель Employee включает все необходимые атрибуты для идентификации и классификации сотрудников, а связь с моделью Department обеспечивает организационную структуру. Особое внимание уделено реализации истории закрепления оборудования, что позволяет отслеживать всю цепочку материальной ответственности для каждого актива.

Интеграция между модулями осуществляется через общую базу данных и единообразные PHP-скрипты обработки. Веб-приложение взаимодействует с базой данных через стандартизированные SQL-запросы, используя единую систему аутентификации на основе сессий. Обмен данными происходит через HTML-формы и перенаправления, что обеспечивает совместимость и простоту отладки.

Диаграмма 7: Схема взаимодействия модулей через REST API



2.3 Отладка отдельных модулей программного проекта

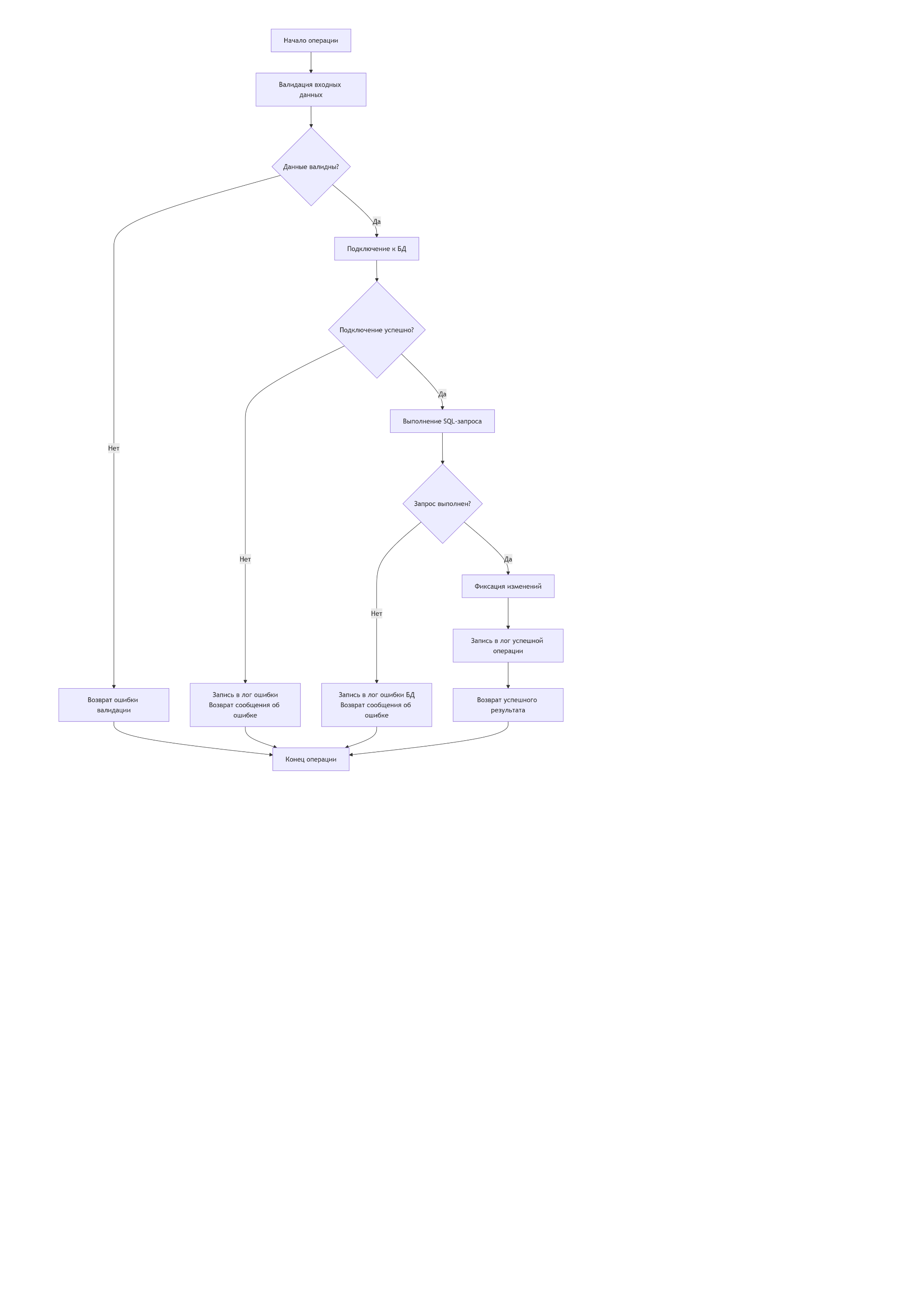
Процесс отладки являлся непрерывной частью разработки каждого модуля и включал многоуровневый подход к обеспечению качества кода.

На уровне back-end отладка осуществлялась с использованием интегрированных отладчиков современных IDE, таких как PhpStorm и VS Code. Точки останова устанавливались в критических участках кода, что позволяло детально анализировать выполнение бизнес-логики, проверять значения переменных и отслеживать поток выполнения. Для сложных сценариев применялось логирование ключевых событий и состояний системы с использованием error\_log() и файлов логов.

Модульное тестирование back-end компонентов реализовано с использованием фреймворка PHPUnit. Для скриптов работы с оборудованием разработаны тесты, проверяющие корректность валидации данных, работу функций расчета амортизации и обработки граничных случаев. Тесты веб-форм проверяют не только успешные сценарии, но и обработку ошибок валидации, случаев отсутствия данных и нарушений прав доступа.

На уровне front-end отладка проводилась с использованием браузерных инструментов разработчика. Анализировалась валидность HTML-разметки, применение CSS-стилей, выполнение JavaScript-кода. Для отслеживания сетевых взаимодействий использовалась вкладка Network, что позволяло оптимизировать запросы к серверу и выявлять проблемы производительности.

Диаграмма 8: Детализированная блок-схема алгоритма добавления оборудования с обработкой ошибок



2.4 Организация обработки исключений

Система обработки исключений проектировалась как многоуровневая архитектура, обеспечивающая надежность работы веб-приложения как на стороне сервера, так и на стороне клиента.

На уровне back-end реализованы глобальные обработчики ошибок PHP, которые перехватывают все необработанные исключения и преобразуют их в структурированные HTML-ответы с соответствующими HTTP-статусами. Для различных типов ошибок разработаны специализированные обработчики: обработка ошибок валидации данных возвращает детализированную информацию о невалидных полях, обработка ошибок аутентификации и авторизации обеспечивает безопасность веб-интерфейса.

Создана система пользовательских ошибок для отражения специфичных бизнес-ситуаций, таких как попытка закрепления уже занятого оборудования или дублирование уникальных идентификаторов. Это позволяет четко разделить типы ошибок и обеспечить единообразную их обработку на всех уровнях приложения.

На уровне front-end реализована централизованная система обработки ошибок JavaScript. Все взаимодействия с веб-формами обернуты в обработчики, которые перехватывают ошибки валидации, ошибки сетевого взаимодействия и специфичные бизнес-ошибки. Пользовательский интерфейс предоставляет информативные сообщения об ошибках, адаптированные для понимания конечными пользователями.

Для обработки ошибок в HTML-формах использована встроенная браузерная валидация с кастомными CSS-стилями для улучшения пользовательского опыта. Дополнительно реализована валидация на стороне сервера для обеспечения безопасности данных.

2.5 Тестирование интерфейса пользователя

Тестирование пользовательского интерфейса проводилось комплексно и включало несколько взаимодополняющих подходов.

Функциональное тестирование интерфейса осуществлялось через ручную проверку всех пользовательских сценариев, описанных в техническом задании. Проверялась корректность отображения данных в различных состояниях системы, работа интерактивных элементов управления, валидация форм ввода и обработка пользовательского ввода.

Особое внимание уделялось тестированию пользовательского опыта (UX): оценивалась интуитивность навигации, понятность терминологии, последовательность действий в многошаговых процессах. Проводилась проверка адаптивности интерфейса - корректного отображения и функционирования на различных устройствах и разрешениях экранов.

Для визуальной проверки использовались инструменты разработчика браузера, позволяющие инспектировать DOM-структуру, анализировать применяемые CSS-стили и отслеживать производительность рендеринга. Browser Developer Tools предоставляли возможность мониторинга сетевых запросов, выполнения JavaScript и диагностики проблем производительности.

На будущее запланировано внедрение автоматизированного тестирования интерфейса с использованием инструментов типа Selenium или Cypress, что позволит обеспечить регрессионное тестирование пользовательского интерфейса при дальнейшем развитии системы.

2.6 Функциональное тестирование и тестирование интеграции

Функциональное тестирование системы проводилось методом "черного ящика", когда тестировщик не имел доступа к внутренней реализации системы и проверял исключительно соответствие внешнего поведения заданным требованиям.

Тестирование критических бизнес-процессов включало проверку полных жизненных циклов типовых сценариев: от постановки оборудования на учет до его списания, включая все промежуточные этапы закрепления, перемещения и ремонта. Каждый сценарий тестировался с различными наборами входных данных, включая корректные данные, граничные значения и заведомо ошибочные данные.

Проверка граничных условий и обработки исключительных ситуаций включала тестирование с максимально допустимыми объемами данных, одновременным доступом нескольких пользователей, обработкой сетевых сбоев и восстановлением после ошибок. Особое внимание уделялось тестированию системы безопасности - проверке разграничения прав доступа, защиты от несанкционированного доступа и корректности работы механизма аутентификации.

Интеграционное тестирование было направлено на проверку взаимодействия между различными компонентами системы. Тестировалась корректность обмена данными между веб-интерфейсом и PHP-скриптами, включая проверку форматов запросов и ответов, обработку ошибок и согласованность состояний между клиентом и сервером.

Тестирование базы данных включало проверку всех SQL-запросов на корректность, производительность и безопасность.

3 Моделирование в программных системах

В рамках раздела «Моделирование в программных системах» была решена классическая задача оптимизации — транспортная задача линейного программирования. Данная задача имеет фундаментальное значение в исследовании операций и находит практическое применение в логистике, управлении цепочками поставок и распределении ресурсов.

3.1 Постановка транспортной задачи

Имеется m поставщиков (пунктов производства) с определенным запасом однородного груза и n потребителей (пунктов потребления) с известными потребностями. Задана матрица транспортных расходов [Cij], где каждый элемент определяет стоимость перевозки единицы груза от i-го поставщика к j-му потребителю. Необходимо найти такой план перевозок [Xij], который обеспечит минимальные суммарные транспортные издержки при условии полного удовлетворения потребностей и полного вывоза запасов.

3.2 Решение задачи в среде Microsoft Excel

Для решения задачи использовался табличный процессор Microsoft Excel с надстройкой «Поиск решения» (Solver). Были подготовлены исходные данные: таблицы запасов, потребностей и матрица тарифов. В процессе настройки была определена целевая функция (суммарные затраты), изменяемые ячейки (объемы перевозок) и заданы ограничения на неотрицательность перевозок, выполнение условий по запасам и потребностям.

3.3 Анализ результатов

После запуска алгоритма симплекс-метода был получен оптимальный план перевозок. Анализ решения показал эффективность распределения потоков и минимизацию холостых пробегов. В контексте разрабатываемой АИС аналогичные методы могут быть применены для оптимизации распределения оборудования между филиалами компании.

Заключение

В ходе учебной практики были успешно достигнуты все поставленные цели и решены комплексные задачи по разработке автоматизированной информационной системы учета компьютерной техники. Практика позволила не только закрепить теоретические знания, но и приобрести ценные практические навыки промышленной веб-разработки.

Ключевые достижения и результаты:

1. Проведен глубокий анализ предметной области, выявлены и формализованы ключевые бизнес-процессы управления IT-активами, что позволило создать полноценное техническое задание в соответствии с государственными стандартами.
2. Спроектирована robust-архитектура веб-системы на основе трехзвенной клиент-серверной модели с четким разделением ответственности между компонентами. Обоснованный выбор технологического стека (PHP/MySQL/Apache для back-end и HTML/CSS/JavaScript для front-end) обеспечил соответствие требованиям производительности, масштабируемости и безопасности.
3. Освоены и применены на практике современные инструменты и методологии коллективной разработки, включая систему контроля версий Git, платформу GitHub, процессы код-ревью и управления задачами. Это позволило организовать эффективный workflow разработки, соответствующий промышленным стандартам.
4. Разработан комплекс визуальных моделей и диаграмм (UML, ER), обеспечивающих однозначное понимание структуры, поведения и данных системы всеми участниками проекта. Моделирование позволило выявить и устранить потенциальные проблемы проектирования на ранних стадиях.
5. Реализованы и интегрированы ключевые функциональные модули системы, обеспечивающие автоматизацию основных бизнес-процессов учета оборудования через веб-интерфейс. Применение модульной архитектуры и единой базы данных обеспечило слабую связанность компонентов и возможность их независимого развития.
6. Внедрена комплексная система обеспечения качества, включающая модульное, интеграционное и функциональное тестирование. Проведена оценка качества программного средства с использованием стандартизированных метрик, показавшая высокий уровень зрелости разработки.
7. Освоены и применены принципы программной инженерии: инспекция кода, рефакторинг, обработка исключений, документирование. Это заложило основу для создания сопровождаемого и развиваемого программного продукта.
8. Исследованы и применены методы математического моделирования и оптимизации на примере решения транспортной задачи, что продемонстрировало междисциплинарный характер современных IT-решений.

Перспективы развития системы: Разработанная АИС "Компьютерная техника" обладает значительным потенциалом для дальнейшего развития. Среди перспективных направлений можно выделить: внедрение модуля прогнозной аналитики для планирования закупок и обновления оборудования, интеграцию с системами бухгалтерского учета и управления предприятием (1С, SAP), реализацию мобильной версии сайта для проведения инвентаризаций с использованием технологии QR-кодов, разработку системы предиктивного обслуживания оборудования на основе анализа истории ремонтов.

Профессиональное значение: Приобретенный опыт составляет прочную основу для дальнейшей профессиональной деятельности в области веб-разработки. Комплексный характер практики, охватывающий все этапы жизненного цикла ПО - от анализа требований до тестирования и внедрения, - позволяет уверенно ориентироваться в современных процессах разработки и эффективно решать сложные профессиональные задачи. Полученные навыки соответствуют актуальным требованиям IT-индустрии и могут быть успешно применены в реальных проектах.

Список литературы

1. Modern PHP: New Features and Good Practices / Josh Lockhart. -- O'Reilly Media, 2021. -- 342 p.
2. PHP and MySQL Web Development / Luke Welling, Laura Thomson. -- 5th ed. -- Addison-Wesley Professional, 2021. -- 672 p.
3. HTML and CSS: Design and Build Websites / Jon Duckett. -- John Wiley & Sons, 2021. -- 490 p.
4. JavaScript and JQuery: Interactive Front-End Web Development / Jon Duckett. -- John Wiley & Sons, 2022. -- 640 p.
5. Git Pro: Everything you need to know to use Git effectively / Scott Chacon, Ben Straub. -- 2nd ed. -- 2023. -- 456 p.
6. Официальная документация PHP [Электронный ресурс]. -- URL: <https://www.php.net/docs.php>
7. Официальная документация MySQL [Электронный ресурс]. -- URL: <https://dev.mysql.com/doc/>
8. Современные методы тестирования программного обеспечения / А. К. Петров. -- М. : ДМК Пресс, 2021. -- 324 с.
9. Проектирование веб-приложений на PHP и MySQL / М. Н. Смирнов. -- СПб. : Питер, 2022. -- 288 с.
10. Автоматизация процессов разработки ПО / Т. П. Васильева. -- М. : Лань, 2020. -- 215 с.