МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное бюджетное образовательное учреждение высшего образования
«Тольяттинский государственный университет»

Кафедра «Прикладная математика и информатика»
(наименование)

09.03.03 Прикладная информатика
(код и наименование направления подготовки)

Разработка программного обеспечения
(направленность (профиль))

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА (БАКАЛАВРСКАЯ РАБОТА)

на тему «Разработка программного обеспечения для учета товарно-материальных ценностей аналитических лабораторий»

Обучающийся И.П. Гринишак
(Инициалы Фамилия) (личная подпись)

Руководитель Н.Н. Казаченок
(ученая степень (при наличии), ученое звание (при наличии), Инициалы Фамилия)

Тольятти 2024

Аннотация

Бакалаврская работа посвящена актуальной проблеме учета товарно-материальных ценностей (ТМЦ) в аналитических лабораториях. В рамках работы осуществлена разработка проекта системы, целью которой является оптимизация процессов учета и управления ТМЦ с применением современных технологий. Проведен анализ существующих методов учета товарно-материальных ценностей, выявлены их недостатки, а также сформулированы требования к разрабатываемой системе.

Определены цели и задачи проектирования. В работе раскрываются логическая и физическая модели системы. Сущность бизнес-процессов раскрывается в нотациях IDEF0 и UML [1], [7, 8], [10, 11, 12], [18]. Выбраны и обоснованы методы разработки программного обеспечения, включая архитектуру системы, обоснован выбор технологий – стек ASP.NET Core : Svelte JS : SQLite [4].

Были проведены тестирование и отладка системы для обеспечения ее надежности и стабильности в работе [6]. Эффективность предложенного решения выражается в повышении надежности учета ТМЦ. Назначение партии идентификатора позволяет значительно ускорить поиск ТМЦ в месте фактического хранения. Аутентификация и минимизация привилегий позволяют снизить риск реализации негативных последствий, связанных с несанкционированным доступом и непреднамеренным внесением изменений в базу данных.

Система автоматизации учета ТМЦ представляет аналитическим лабораториям современный инструмент, отвечающий требованиям оптимизации процессов, связанных с движением товарно-материальных ценностей, в том числе реактивов и лабораторного оборудования.

Оглавление

Введение

Автоматизация учёта товарно-материальных ценностей (ТМЦ) эволюционировала от ручных методов к современным высокотехнологичным решениям. Изначально учёт велся вручную, что было сопряжено с большим количеством ошибок. С появлением компьютерных систем в середине XX века началась первая волна автоматизации, упростившая учёт и повысившая его точность. В 1970 – 1980 гг. появились специализированные программы для управления запасами, такие как MRP-системы, которые позже переросли в более сложные ERP-системы, интегрирующие все аспекты управления бизнесом. В 1990 – 2000 гг. ERP-системы стали основным инструментом для управления ТМЦ.

Инновации, такие как штрих-коды и RFID-метки, значительно улучшили точность и эффективность учёта. Современные облачные решения и мобильные приложения предоставляют доступ к данным из любой точки мира, повышая гибкость и надёжность процессов. Искусственный интеллект и аналитика больших данных открывают новые возможности для оптимизации управления ТМЦ.

Актуальность работы обусловлена тем, что современные аналитические лаборатории сталкиваются с необходимостью эффективного учета и управления ТМЦ. Ведение точных записей о фактическом наличии ТМЦ, их перемещении и использовании является критически важным для обеспечения бесперебойной работы лабораторий, минимизации потерь и предотвращения ошибок в аналитических исследованиях. Это особенно актуально для лабораторий, где точность и надежность данных имеют первостепенное значение.

Новизна работы заключается в разработке современного инструментария для учета и управления ТМЦ в аналитических лабораториях.

Теоретическая значимость работы – обобщение процессов управления ТМЦ позволит разработать новые методологические подходы, которые могут

быть применены для повышения эффективности учета ТМЦ в аналитических лабораториях.

В качестве объекта исследования рассматриваются процессы учета и управления ТМЦ в аналитической лаборатории.

Предметом исследования является автоматизация учета и управления товарно-материальными ценностями в аналитической лаборатории.

Целью работы является разработка эффективной и надежной системы учета ТМЦ для аналитических лабораторий с использованием современных технологий. В соответствии с поставленной целью в работе определены следующие задачи:

- анализ требований и особенностей учета ТМЦ в аналитических лабораториях;
- выбор и обоснование стека технологий для разработки системы;
- проектирование архитектуры системы;
- разработка серверной части;
- создание «фронтенд» – интерфейса;
- интеграция базы данных;
- тестирование и отладка системы.

Основные решения, выносимые на защиту:

- архитектура программного обеспечения для учета ТМЦ в лабораториях;
- функциональные и технические характеристики разработанной системы.

В работе используется структурный подход к анализу и проектированию, реализация которого опирается на анализ требований, проектирование и документирование.

Методы исследования. Моделирование на этапе проектирования включает использование технологии UML. Применение принципов ООП осуществлено при разработке серверной и клиентской частей приложения.

Разработка программного обеспечения, рассмотренного в данной работе, предоставит аналитическим лабораториям возможность автоматизировать процессы учета ТМЦ. Это способствует повышению надежности исследований и улучшению общей эффективности работы испытательных лабораторий в целом.

Таким образом, данная работа представляет собой важный шаг к созданию современных и эффективных инструментов для управления ТМЦ в аналитических лабораториях.

Бакалаврская работа состоит из введения, трех глав, заключения и списка литературы.

Во введении обосновывается выбор темы и подчёркивается значимость исследования, обусловленная его актуальностью. Формулируется цель работы и ставятся конкретные задачи, необходимые для её достижения.

В первой главе проводится анализ потребностей аналитических лабораторий в системе учета ТМЦ. Рассматриваются текущие методы учета и их недостатки, а также проводится обзор существующих программных решений, их преимущества и недостатки. Формулируются требования к разрабатываемой системе.

Вторая глава включает определение функциональных требований к системе и описание ее общей структуры. Рассматривается проектирование и разработка серверной части с использованием ASP.NET Core, а также клиентской части с использованием фреймворка, основанного на JavaScript – технологии. Описывается взаимодействие между компонентами системы и структура реляционной базы данных.

Третья глава посвящена тестированию и отладке системы для обеспечения ее стабильной работы.

Выпускная квалификационная работа состоит из 60 страниц и содержит 37 рисунков, 21 источник.

Глава 1 Постановка задачи на разработку программного обеспечения для учёта товарно-материальных ценностей аналитических лабораторий

1.1 Анализ и описание технологий учёта товарно-материальных ценностей аналитических лабораторий

Процесс управления товарно-материальными ценностями (ТМЦ) на любом предприятии требует особого внимания, поскольку он является ключевым элементом затрат, без которых не обходится ни одно производство. В него входят различные виды запасов, которые составляют основу для создания продукции. Если не организовать систематический учет перемещений ТМЦ, это может нарушить нормальный ход работы компании, повысить риск ненужных расходов и возникновения ошибок, которые впоследствии могут вылиться в серьезные финансовые затраты. Чтобы избежать проблем в конце отчетного периода, важно вовремя проводить и документировать все операции.

Основные процессы, сопровождающие движение товарно-материальных средств, включают в себя прием, хранение, выдачу и учет. За выполнение этих задач должно отвечать материально ответственное лицо, которое должно быть уполномочено на выполнение этих функций. Уже по этой причине сложно организовать на надлежащем уровне прослеживаемость движения ТМЦ, при использовании в качестве первичных документов, незащищенных электронных рабочих книг, особенно при отсутствии налаженного электронного документооборота.

Каждая операция с ТМЦ должна быть задокументирована с использованием первичных учетных документов, как того требует статья 9 Федерального закона № 402-ФЗ «О бухгалтерском учете». Эти документы подтверждают юридическую достоверность всех действий с материальными ценностями и содержат подписи ответственного лица и бухгалтера.

Примерами таких документов являются товарные накладные, акты приема-передачи или списания, приходные и расходные ордера.

Хотя строгого формата для первичных документов не существует, есть определенные обязательные реквизиты, которые должны быть указаны: название документа и дата его составления; наименование организации, составившего документ; описание операции; единицы измерения (штуки, литры, килограммы, рубли и т.д.); подпись ответственного лица.

Эти документы составляют основу для бухгалтерских записей, которые затем вносятся в специальные учетные регистры. Регистры используются для систематизации информации из первичных документов и служат базой для составления бухгалтерской отчетности. Хранение первичных документов и учетных регистров должно быть обеспечено в течение минимум пяти лет после окончания отчетного периода.

Регистры позволяют оперативно контролировать движение ТМЦ на складе. В качестве регистров могут использоваться журналы, карточки товаров, складские книги и другие документы, предназначенные для учета материальных ценностей.

Документы можно оформлять как в бумажном, так и в электронном виде, что особенно актуально в современных условиях.

Ведение учета вручную сопряжено с рядом сложностей. Оно требует синхронизации с бухгалтерскими данными и усиленного контроля для предотвращения хищений. Кроме того, ручной учет увеличивает риск ошибок в документации, что может привести к проблемам с налоговыми органами. Поэтому для точного и эффективного учета остатков на складе рекомендуется использование автоматизированных систем.

Учет ТМЦ – многоплановый процесс, затрагивающий различные подразделения организации, начиная бухгалтерией и заканчивая складским хозяйством. В зависимости от характера учета, различают количественно-суммовой и сальдовый метод. В случае применения сальдового метода, на складе ведется учет только по количеству, а в бухгалтерии – по стоимости.

В данной работе рассматривается складской учет постоянно расходуемых товарно-материальных ценностей, находящихся на балансе испытательной лаборатории, с акцентом на применение программного продукта.

Существует два основных подхода к организации учета товаров на складах: партионный и сортовой.

Партионный метод предполагает хранение каждой партии ТМЦ отдельно. Данный метод положен в основу программного продукта, рассматриваемого в настоящей работе. При этом подходе, на этапе регистрации, вновь принятой партии материала присваивается индивидуальный регистрационный номер. Очевидны плюсы такого подхода: полная прослеживаемость движения материалов, без необходимости проведения периодической инвентаризации; интенсификация контроля за оборотом материалов; снижение финансовых потерь организации за счет оперативного планирования закупок, на основании оперативных отчетов о состоянии ТМЦ.

Сортовой метод ведения учета предполагает классификацию материалов по наименованиям и сортам, независимо от даты поступления или их стоимости.

Определить единый универсальный способ учета практически невозможно, так как выбор метода зависит от объема номенклатуры и ассортимента товаров, характера их получения и объекта возникновения трат, аспектов ведения документации, отражающей движение матсредств.

1.2 Обзор и анализ аналогов программного обеспечения для учёта товарно-материальных ценностей аналитических лабораторий

Рынок программного обеспечения предоставляет ряд решений для складского учета. Наиболее известные и широко распространенные из их числа рассмотрены.

«Контур» – предлагает одно из ведущих решений для складского учета. Обладает рядом недостатков: пользователю необходимо обладать хотя бы минимальными бухгалтерскими знаниями; интерфейс меню несколько перегружен бухгалтерским функционалом.

«1С: Торговля и склад» – славится своей гибкостью и простотой в настройке, позволяя выполнять весь спектр учетных функций – от управления справочниками и создания первичных документов до составления различных отчетов и ведомостей. Обладает недостатками: высокая стоимость; сложность обучения; длительное внедрение и настройка.

«Sigma» – решение для управления складским учетом и автоматизации процессов продаж. Обладает недостатками: недоступность функций складского учета в минимальном тарифном плане.

«LiteBox» – облачный сервис, предназначенный для управления торговлей и финансовым учетом, специально разработанный для автоматизации процессов малого бизнеса. Его функционал позволяет эффективно организовывать торговлю, закупки и складской учет, а также формировать и наглядно представлять аналитические отчеты. Обладает недостатками: перегруженный GUI-интерфейс; невозможность доработки интерфейса под конкретного пользователя.

Резюмируя, среди наиболее часто встречающихся проблем, с которыми сталкивается конечный пользователь, стоит отметить: взаимное наложение функционала; перегруженность интерфейса; высокая стоимость; сложность в обучении и внедрении; ограничения функционала и сложности в его кастомизации.

1.3 Разработка требований к программному обеспечению для учёта ТМЦ

1.3.1 Разработка и анализ модели «как есть»

Объектом исследования и автоматизации выступает пробирно-аналитическая лаборатория месторождения «Кумроч», АО «Быстринская горная компания». На момент написания проекта, лаборатория проходит этапы проектирования и строительства. Необходимость учета разных типов товарно-материальных ценностей, отнесенных к различным подразделениям, позволяет унифицировать задачу до использования формального «склада», на котором числятся все подлежащие учету ТМЦ. Действительно, Такой подход позволяет отслеживать и управлять ТМЦ в единой системе. Это снижает вероятность дублирования данных, ошибок учета и потерь при перемещении материалов между подразделениями (централизованный контроль).

Вся информация по ТМЦ доступна в одном месте, что позволяет легко контролировать запасы, видеть, какие материалы находятся на «складе» и их текущее состояние. Это также упрощает проведение инвентаризации (прослеживаемость).

Централизованный учет позволяет более точно планировать закупки и использование ТМЦ, что уменьшает избыточные запасы и снижает расходы на их хранение (снижение издержек).

Такой подход легче адаптировать к изменениям в структуре организации или изменению количества и типов ТМЦ. Лаборатория может начать с малого количества позиций и масштабировать учет по мере необходимости (гибкость и масштабируемость).

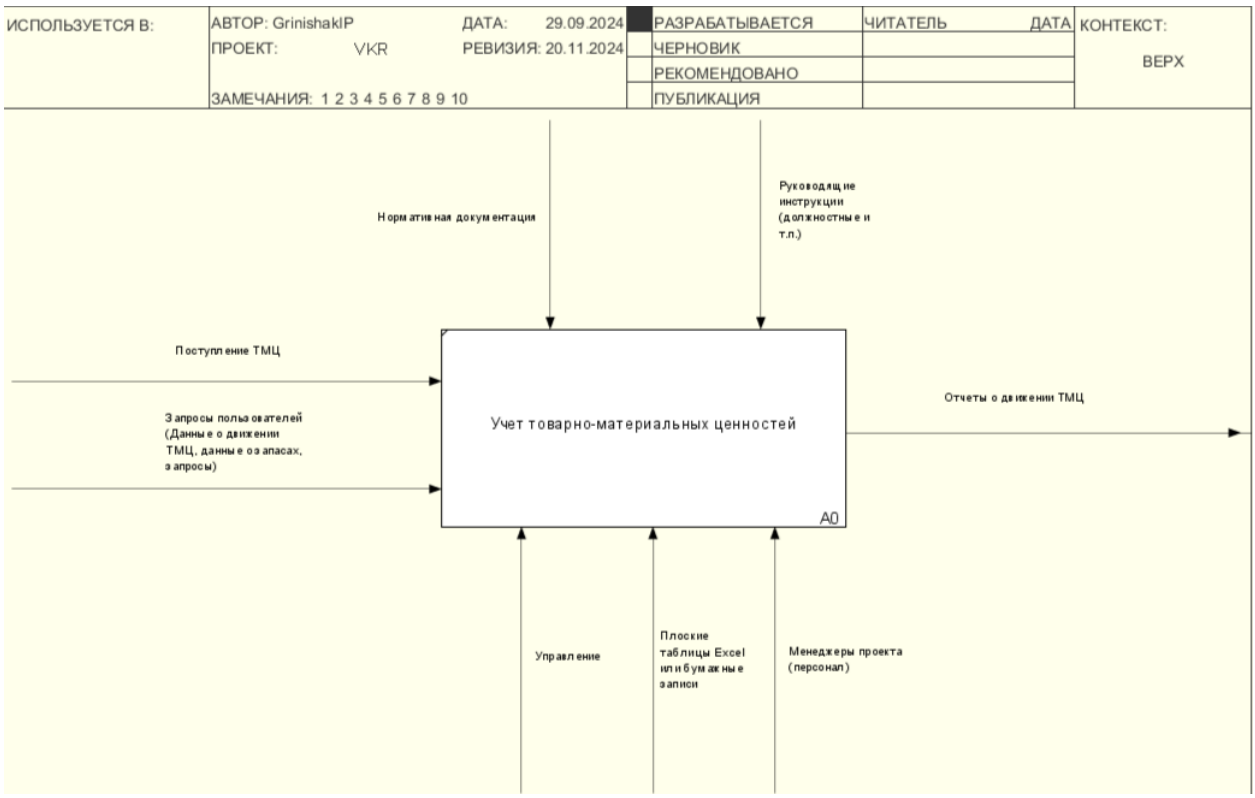


Рисунок 1 – Представление бизнес-процессов учёта ТМЦ в нотации IDEF0, уровень декомпозиции A0, «AS-IS»

Входы включают данные о движении товарно–материальных ценностей.

Выходы представляют собой отчеты о движении ТМЦ (актуальная информация о наличии товаров), документы, отражающие учет товарно-материальных средств.

Механизмы включают управление, персонал, плоские таблицы Excel (платформа для учета ТМЦ).

Управление определяется нормативной документацией и руководящими документами – инструкциями и т.п.

Определение и выявление требований, необходимых для автоматизации

Исходя из вышеперечисленных недостатков системы было решено определить требования к будущей системе.

На основании анализа диаграммы IDEF0, описывающей процессы учета товарно-материальных ценностей, стало очевидным, что для эффективного

управления движением товаров и учета запасов необходимо разработать специализированное программное обеспечение (рисунок 1).

Благодаря анализу диаграммы IDEF0, выявлены ключевые функции, такие как отслеживание входящих и исходящих движений ТМЦ, учет запасов по группам, управление многоуровневым доступом пользователей и необходимость создания интуитивно понятного интерфейса. Также были обозначены требования к масштабируемости системы для адаптации к росту производственных мощностей, с сопутствующим развитием бизнес-системы.

При разработке программного обеспечения для учета товарно-материальных ценностей (ТМЦ) важно учитывать как функциональные, так и нефункциональные требования, для эффективного выполнения системой задач и наиболее полного соответствия потребностям пользователей. Рассмотрим требования внутри каждого из блоков по отдельности.

а) блок функциональных требований:

1) система должна отслеживать входящие и исходящие движения ТМЦ, фиксируя каждую операцию и позволяя пользователям получать актуальные данные о наличии товаров;
2) система должна обеспечивать возможность учета запасов по различным категориям и группам, включая информацию о производителе, сроках годности и партиях;
3) программное обеспечение должно поддерживать многоуровневую систему управления доступом, позволяя настраивать права для различных категорий пользователей.

б) блок нефункциональных требований:

1) интерфейс программного обеспечения должен быть интуитивно понятным и удобным для пользователей, что способствует снижению времени на обучение и повышению эффективности системы;
2) важно учитывать возможность параллельной работы нескольких пользователей;

3) система должна быть готова к масштабированию, позволяя добавлять новые функции.

Постановка задачи на разработку приложения

Чтобы поставить задачу разработки веб-приложения для автоматизации процесса учета товарно-материальных ценностей (ТМЦ), необходимо учесть недостатки существующих систем, выявленные в предыдущем анализе. Это позволит сформулировать требования и цели для нового приложения. Особенное внимание необходимо уделить архитектуре приложения [2], [13].

Задача включает в себя:

– уменьшение времени, затрачиваемого на учет движения ТМЦ,
– автоматизацию процессов учета товаров, что повысит скорость обработки информации;
– упрощение учета запасов с возможностью внесения данных о дате производства и сроках годности, на этапе верификации продукции;
– переход к электронному ведению записей, если это не противоречит требованиям действующего законодательства РФ или иных ведомственных или отраслевых документов (например, учет драгметаллов или прекурсоров);
– повышение прозрачности и доступности информации для пользователей с разными уровнями доступа;
– генерацию отчетов (вывод результатов на экран монитора) о движении ТМЦ и состоянии запасов в реальном времени.

Требования к функциональности приложения

На основании системного анализа выдвигаются следующие требования к функциональности приложения:

– наличие базы данных для хранения информации о движении ТМЦ, что позволяет быстро получать актуальные данные о наличии товаров;
– наличие интерфейса для ввода и редактирования данных о поступлении;
– наличие системы управления доступом, обеспечивающей

разграничение прав для различных категорий пользователей;

– генерация отчетов о движении ТМЦ.

1.3.2 Требования к архитектуре и реализации приложения

На основании системного анализа, выдвигаются следующие требования архитектуре приложения:

– масштабируемость и гибкость. Архитектура должна поддерживать гибкость и масштабируемость для обеспечения возможности роста и адаптации приложения под новые требования, изменяющиеся с течением времени. Это важно для учета большого объема данных ТМЦ и их возможного расширения в лаборатории;

– модульность. Приложение должно быть построено на модульной архитектуре, где каждая часть (серверная, клиентская, база данных) может независимо развиваться и обновляться, обеспечивая лёгкую замену или улучшение модулей без затрагивания других компонентов системы;

– производительность. Веб-приложение должно эффективно обрабатывать запросы пользователей, обеспечивая быструю реакцию системы и минимальные задержки при взаимодействии с базой данных и интерфейсом;

– удобство разработки и поддержки. Приложение должно быть построено с использованием современных технологий, упрощающих разработку, тестирование, и последующую поддержку системы.

1.3.3 Обоснование выбора стека используемых в работе технологий

Использование ASP.NET Core обеспечивает высокую производительность, масштабируемость и легкость интеграции с различными внешними сервисами и базами данных [14, 15, 16]. Благодаря поддержке кроссплатформенности и встроенным средствам для работы с аутентификацией и авторизацией, идеально подходит для разработки серверной части приложения.

На стороне клиента, (JS, Svelte) позволяет создавать интерактивный, отзывчивый пользовательский интерфейс, который предоставляет удобные инструменты для работы с ТМЦ в реальном времени, делая взаимодействие с системой простым и эффективным. [17], [19, 20].

СУБД SQLite – это отличный выбор для программного обеспечения учета движения материальных средств, в условиях высоких нагрузок и отсутствия сложных параллельных вычислений [3], [5], [9], [21]. Простота в использовании, надежность, легковесность и минимальные затраты на обслуживание делают SQLite предпочтительным решением для небольших и средних программных систем учета, не требующих сложной серверной архитектуры.

Таким образом, выбранный стек технологий не только удовлетворяет функциональным и техническим требованиям, но и обеспечивает гибкость, надежность и удобство разработки и эксплуатации системы.

Минимальные требования к параметрам оборудования и сопутствующего программного обеспечения определяются нагруженностью системы.

При низкой нагруженности системы (исходя из системного анализа требований испытательных лабораторий, к клиент-серверной архитектуре), клиентская и серверная части могут быть сопоставимы по производительности.

CPU: Intel Core i3 2.0 GHz, RAM: 4 GB, GPU: встроенная (Intel HD Graphics 4000), сетевое подключение: 10 Mbps, ОС: Windows 10/11 с поддержкой ASP.NET Core. Браузер: Chrome v.121.0.0.0, Yandex v.24.7.0.0

С учетом вышеизложенного, мы можем перейти к разработке модели «как должно быть», что позволит более четко определить архитектуру и функциональность системы, отвечающую всем требованиям бизнеса.

1.3.4 Разработка и анализ модели бизнес-процесса «как должно быть»

С учетом анализа процессов в IDEF0 – нотации предложена модель «TO-BE» (рисунок 2).

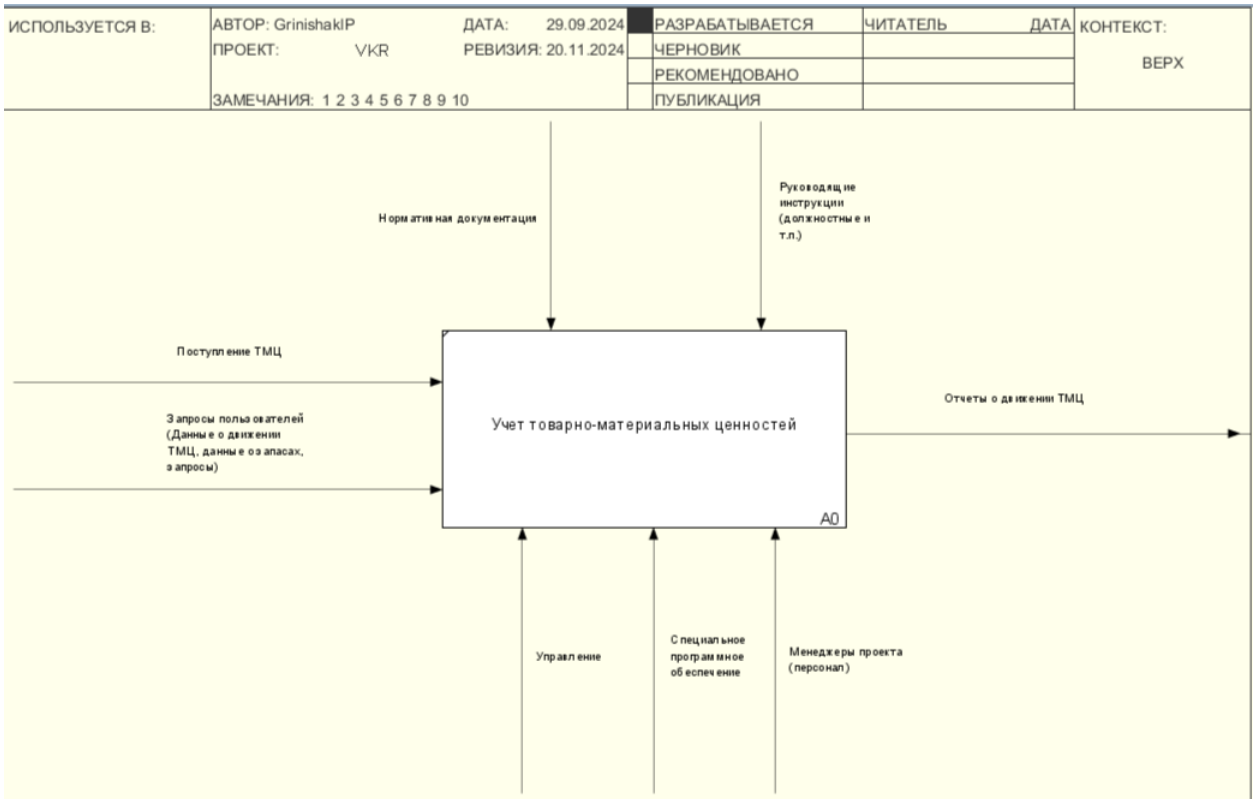


Рисунок 2 – Представление бизнес-процессов движения ТМЦ в нотации IDEF0, уровень декомпозиции A0, «TO-BE»

Входы, выходы, механизмы, управление идентичны приведенной ранее схеме «как есть» за исключением изменения в механизме управления данными – взамен плоских электронных таблиц используется самостоятельное программное обеспечение.

Выводы по главе 1

Первая глава работы включает анализ бизнес-процесса учета товарно-материальных ценностей. Рассмотрены основные подходы к учету ТМЦ. Осуществлен анализ представленных на рынке программных продуктов,

которые могут быть использованы при автоматизации задач интересующей предметной области. На основании декомпозиции бизнес-процесса, сформулированы требования к программному обеспечению. Дано обоснование выбора стека используемых в разработке технологий.

Глава 2 Проектирование программного обеспечения для учёта товарно-материальных ценностей аналитических лабораторий

2.1 Выбор методологии проектирования программного обеспечения для учёта товарно-материальных ценностей аналитических лабораторий

Выбор объектно-ориентированного проектирования (ООП) в сочетании с водопадной моделью разработки – это сильное и логичное решение для создания систем, особенно в контексте разработки программного обеспечения для учета товарно-материальных ценностей (ТМЦ).

Водопадная модель – линейный и пошаговый подход к разработке программного обеспечения, где каждый этап строго завершает предыдущий, и возвращение назад возможно только на этапе поддержки. Для ООП это подходит следующим образом:

Четкое планирование и проектирование. На этапе проектирования создаются классы, их взаимосвязи и структура системы. Поскольку водопадная модель требует четко зафиксированных требований на старте, весь объектно-ориентированный проект будет хорошо продуман до начала разработки.

Последовательное развитие. В ООП каждый класс и модуль может быть разработан последовательно, а затем протестирован и интегрирован с другими частями системы. Водопадная модель поддерживает такой линейный процесс.

Стабильность проекта. ООП, благодаря своей модульности, позволяет реализовывать сложные системы, и водопадная модель обеспечивает стабильность, особенно при четко заданных требованиях к учету ТМЦ.

Преимущества сочетания ООП и водопадной модели заключаются в следующем.

Во-первых, это планируемость: водопадная модель идеально подходит для тех случаев, когда требования к системе стабильны и хорошо документированы, как это часто бывает в системах учета.

Во-вторых, четкая структура разработки: благодаря объектно-ориентированному подходу система будет разделена на логически завершенные классы и объекты, что облегчит планирование и реализацию каждого компонента.

В-третьих, простота в сопровождении: после завершения разработки и внедрения системы ООП обеспечивает легкость поддержки и обновлений, а водопадная модель гарантирует, что все функции протестированы и работают должным образом. Таким образом, сочетание ООП и водопадной модели обеспечивает структурированную, четкую и надежную разработку систем учета ТМЦ.

2.2 Логическое моделирование программного обеспечения для учёта товарно-материальных ценностей аналитических лабораторий

Выбор метода логического моделирования – это важный шаг в процессе разработки программного обеспечения и системного анализа.

Унифицированный язык моделирования (UML, Unified Modeling Language) представляет собой стандартизированный набор нотаций, который широко используется для моделирования объектов и их взаимодействий в объектно-ориентированных системах. UML предоставляет гибкий инструмент для визуализации, спецификации, разработки и документирования различных аспектов программного обеспечения.

В основе UML лежат три ключевых элемента: диаграммы, сущности, связи.

Сущности представляют собой абстракции реальных или концептуальных объектов, которые составляют модель. Они образуют основу, которая определяет структуры и взаимодействия в системе. Сущности могут

быть сгруппированы в диаграммы – графические представления взаимосвязанных объектов, что облегчает понимание общей картины. Связи, в свою очередь, описывают отношения между сущностями.

UML включает множество видов диаграмм, однако наиболее популярными являются: диаграмма классов, диаграмма активности, диаграмма прецедентов.

Диаграмма прецедентов отражает взаимодействие акторов с системой. В рассматриваемой системе имеется два типа акторов – пользователь и администратор (рисунок 3)

Рисунок 3 – Диаграмма прецедентов (упрощена для удобства восприятия)

Конкретные функции и сценарии, реализуемые при взаимодействии акторов с системой, отражены на рисунках 3 и 4.

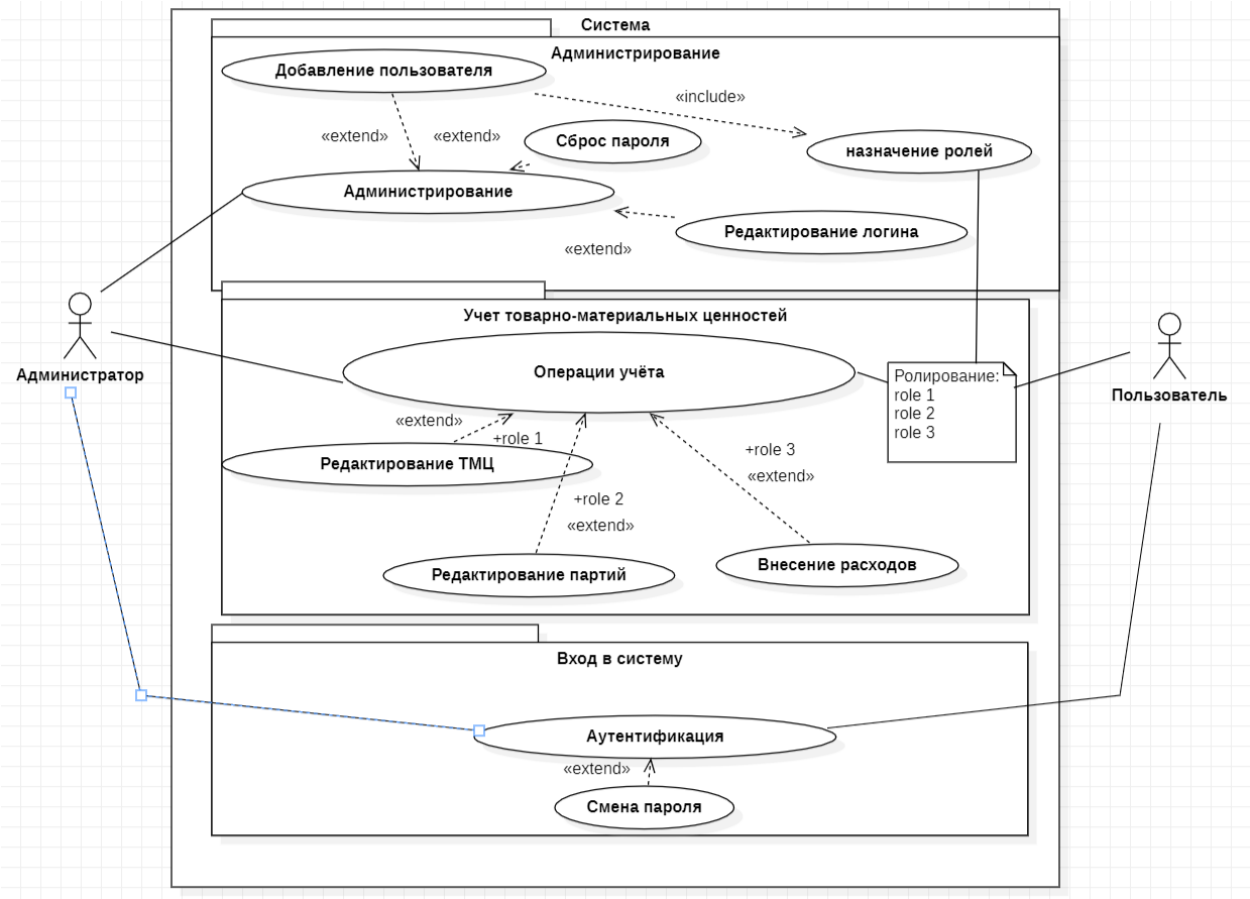


Рисунок 4 – Диаграмма прецедентов (полная схема)

Администратор представляет собой суперпользователя, наделенного функциями всех ролей. На рисунке 4 этот аспект раскрыт наиболее полно.

Визуализация последовательности действий, условий, а также параллельных процессов наглядно представлена на рисунке 5.

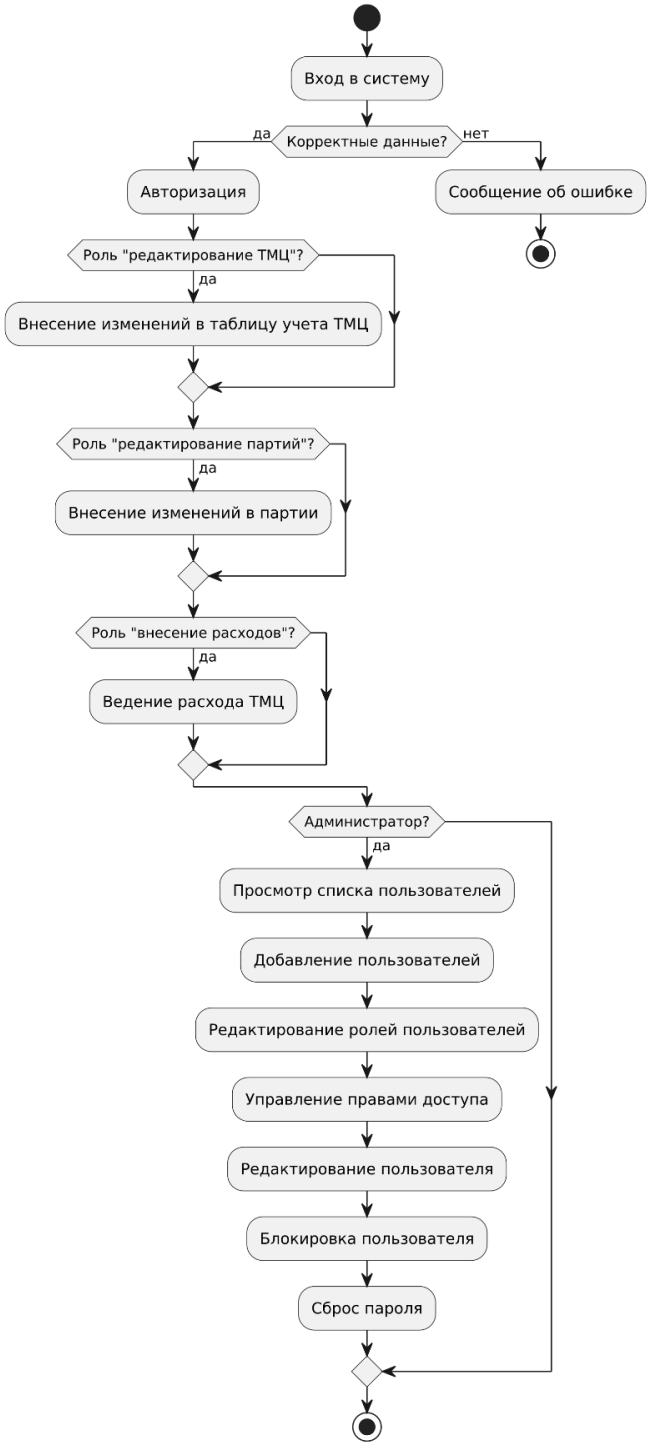


Рисунок 5 – Диаграмма деятельности

Администратору доступен функционал приложения в соответствии с функциями по всем трем ролям, наделение которыми обеспечивает пользователей возможностями редактирования и внесения данных о товарно-

материальных ценностях в систему. Помимо прочего, администратор обеспечивает пользовательскую политику.

Диаграмма компонентов (Component Diagram) — это один из типов диаграмм в UML (Unified Modeling Language), предназначенный для моделирования структуры программных систем на уровне компонентов. Она отображает, как различные компоненты системы взаимодействуют друг с другом и как они организованы внутри системы. Основными компонентами разрабатываемой системы выступают компоненты авторизации, ролирования, функционала пользователя, функционала администратора, базы данных. Взаимодействие компонентов между собой приведено на рисунке 6.

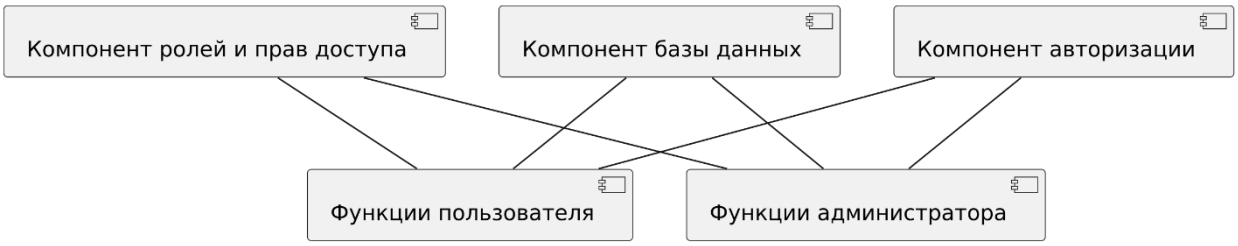


Рисунок 6 – Диаграмма компонентов (упрощена для удобства восприятия)

Взаимодействие контроллеров сервера и функционала клиентской части рассматриваются на диаграмме компонентов, рисунок 7.

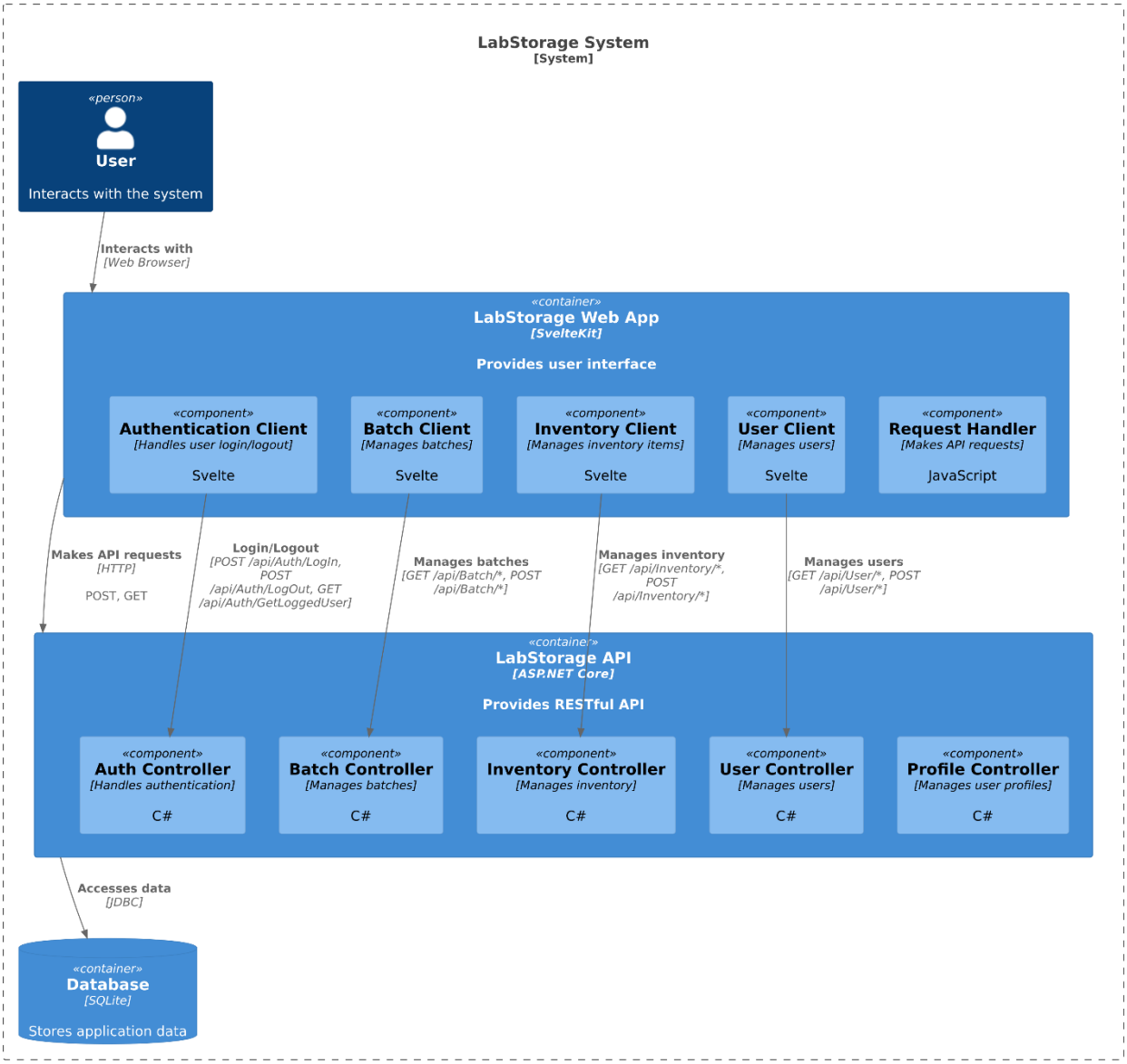


Рисунок 7 – Диаграмма компонентов (полная схема)

Основные компоненты, которые иллюстрирует диаграмма компонентов, включают:

LabStorage Web App – часть приложения, которая предоставляет пользовательский интерфейс и содержит различные клиентские компоненты, такие как «Authentication Client», «Batch Client», «Inventory Client», «User Client» и «Request Handler».

LabStorage API - это интерфейс, который предоставляет доступ к основным функциям системы, таким как аутентификация, управление партиями, управление инвентарем и управление пользователями.

Контроллеры – компоненты, которые реализуют логику для работы с API, такие как «AuthController», «BatchController», «InventoryController», «UserController» и «ProfileController».

Database – компонент, который хранит данные приложения, такие как учетные данные пользователей, информация о движении и остатке товарно-материальных ценностей.

Диаграмма развертывания (Deployment Diagram) — это один из типов диаграмм в UML (Unified Modeling Language), который используется для визуализации архитектуры системы в развернутом состоянии. Она показывает, как программное обеспечение и его компоненты размещаются на аппаратных устройствах и как они взаимодействуют друг с другом. Диаграмма развертывания показана на рисунке 8.

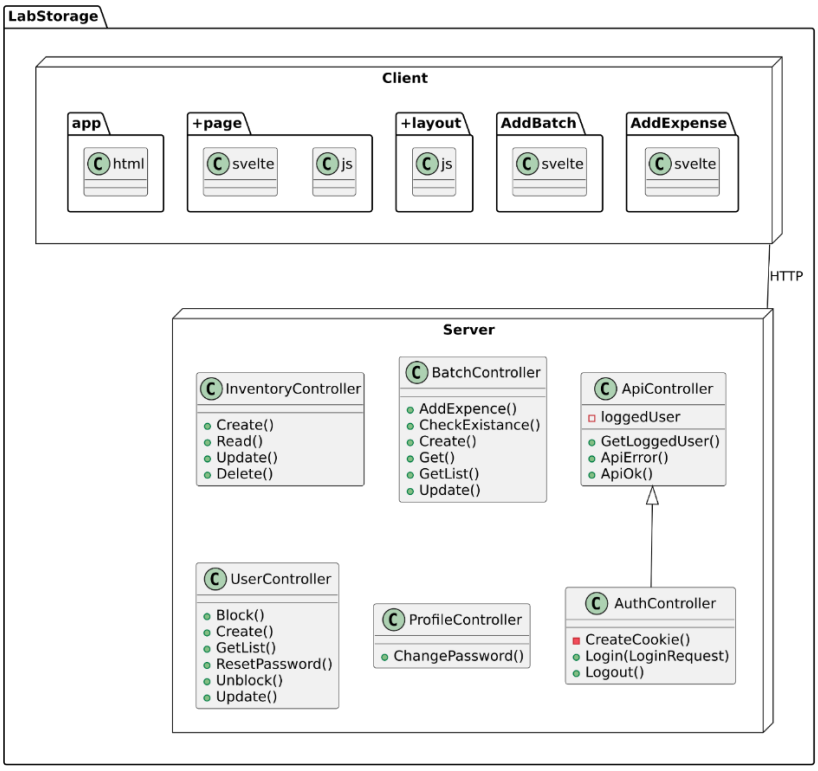


Рисунок 8 – Диаграмма развертывания

Эта диаграмма развертывания показывает архитектуру и компоненты LabStorage системы, в которой клиентская часть взаимодействует с серверной частью через HTTP – протокол.

На стороне клиента показаны компоненты, такие как «app», «page», «layout», «AddBatch» и «AddExpense», которые реализованы с использованием HTML, Svelte и JavaScript.

На стороне сервера мы показаны несколько контроллеров, реализованных на C#:

InventoryController - отвечает за управление инвентарем (Create, Read, Update, Delete).

BatchController - отвечает за управление партиями (AddExpence, CheckExistance, Create, Get, GetList, Update).

UserController - отвечает за управление пользователями (Block, Create, GetList, ResetPassword, Unblock, Update).

AuthController - отвечает за аутентификацию (CreateCookie, Login, Logout).

ProfileController - отвечает за управление профилем пользователя (ChangePassword).

Также на стороне сервера есть ApiController, который обрабатывает API-запросы от клиентской части.

Диаграмма развертывания дает общее представление об архитектуре и компонентах LabStorage системы, показывая, как клиентская и серверная части взаимодействуют друг с другом.

Диаграмма классов предназначена для представления структуры системы, включая типы классов и их статические связи. Она отображает классы, их атрибуты, методы, а также ограничения, которые определяют отношения между ними. На этот тип диаграмм существенно влияет уровень абстракции: диаграммы могут быть использованы для анализа предметной области или для проектирования конкретных элементов системы. В первом

случае фокус смещается на сущности, отражающие реальный мир, тогда как в проектировании внимание уделяется реализации классов в программном коде.

При разработке программного обеспечения было принято решение сделать упор на представление структуры системы в формате диаграммы классов (рисунок 9).

Рисунок 9 – Диаграмма классов проекта «LabStorage»

Подробное описание рассматриваемых классов приведено в разделе 3.1.

2.3 Моделирование данных системы для учёта товарно-материальных ценностей аналитических лабораторий

В данной базе данных, состоящей из пяти таблиц, осуществляется учет товарно-материальных ценностей, что позволяет эффективно управлять запасами, контролировать расходы и обеспечивать безопасность пользователей системы.

База данных осуществляет учет товарно-материальных ценностей, что позволяет эффективно управлять материальными ценностями – контролировать расход, обеспечивать доступ в систему. База данных состоит из пяти таблиц: Inventory – «Список», Batches – «партии», Expences – «расходы», Users – «Пользователи», UserRole – «Ролирование» (рисунок 10):

Таблица Inventory является основой всей системы, поскольку в ней хранится информация о всех товарно-материальных ценностях. Она позволяет отслеживать, какие материалы имеются в наличии, а также управлять запасами.

Структура таблицы Inventory:

– Name (строковый тип данных): Название инвентаря (например, «хлорид калия»). Это поле позволяет легко идентифицировать каждый товар в системе;
– Unit (строковый тип данных): Единица измерения, в которой ведется учет (например, «кг», «шт»). Это поле помогает пользователям понимать, в каких единицах они работают с запасами;
– Id (целочисленный тип данных): Уникальный идентификатор для каждой записи. Это поле служит ключом для связи с другими таблицами.

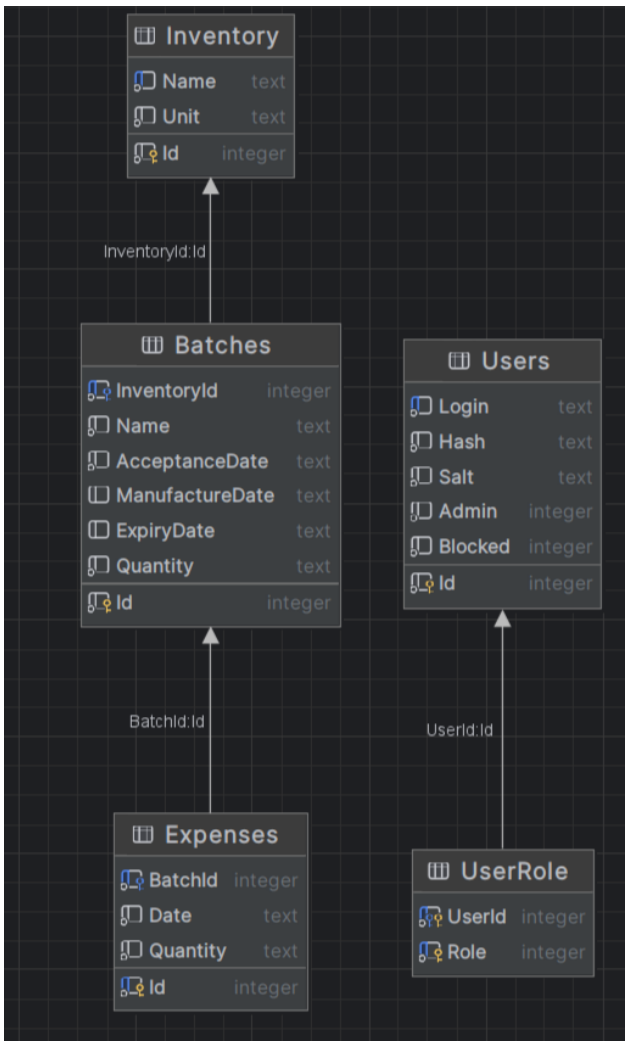


Рисунок 10 – Структура базы данных

Таблица Batches хранит информацию о партиях товаров, что позволяет отслеживать их движение, контролировать запасы и управлять сроками годности. Связь с таблицей Inventory позволяет быстро находить информацию о партиях для конкретного товара.

Структура таблицы Batches:

– InventoryId (целочисленный тип данных): Ключевое поле, связывающее данную запись с конкретным товаром из таблицы Inventory. Это позволяет идентифицировать, к какому товару относится партия;
– Name (строковый тип данных): Название партии или дополнительные

характеристики, помогающие идентифицировать ее;

- AcceptanceDate (строковый тип данных): Дата принятия партии на склад. Это поле помогает отслеживать, когда партия была добавлена в список;
- ManufactureDate (строковый тип данных): Дата производства партии. Позволяет пользователям следить за свежестью товаров;
- ExpiryDate (строковый тип данных): Дата окончания срока годности. Это важное поле для управления качеством материалов и предотвращения потерь;
- Quantity (строковый тип данных): Количество товара в партии. Указывает, сколько единиц товара находится в данной партии;
- Id (целочисленный тип данных): Уникальный идентификатор для каждой записи партии.

Таблица Expences позволяет вести учет всех расходов, связанных с партиями товаров. Это важно для анализа затрат и управления запасами, что способствует более эффективному использованию ресурсов.

Структура таблицы Expences:

- BatchId (целочисленный тип данных): Ключевое поле, связывающее данную запись с конкретной партией из таблицы Batches. Это позволяет идентифицировать, какие расходы связаны с какой партией;
- Date (строковый тип данных): Дата расхода. Это поле помогает отслеживать, когда произошел расход;
- Quantity (строковый тип данных): Количество расходуемого товара. Позволяет контролировать, сколько единиц было потрачено.

Таблица Users отвечает за безопасность системы, храня информацию о пользователях, их учетных данных и ролях. Это важно для контроля доступа и защиты данных.

Структура таблицы Users:

- Login (строковый тип данных): Уникальное имя пользователя для входа в систему. Это поле необходимо для аутентификации

пользователей;

– Hash (строковый тип данных): Хэш пароля для обеспечения безопасности. Хранение паролей в зашифрованном виде предотвращает несанкционированный доступ;
– Salt (строковый тип данных): Соль, используемая для хэширования пароля. Добавляет дополнительный уровень безопасности при хранении паролей;
– Admin (целочисленный тип данных): Указывает, является ли пользователь администратором (например, 1 - да, 0 - нет). Это поле помогает управлять правами доступа;
– Blocked (целочисленный тип данных): Указывает, заблокирован ли пользователь (например, 1 - да, 0 - нет). Это поле позволяет временно приостанавливать доступ пользователей к системе;
– Id (целочисленный тип данных): Уникальный идентификатор для каждого пользователя.

Таблица UserRole позволяет управлять ролями пользователей, что помогает разграничивать доступ к различным функциям системы. Это критически важно для обеспечения безопасности и правильного функционирования приложения.

Структура таблицы UserRole:

– UserId (целочисленный тип данных): Ключевое поле, связывающее данную запись с конкретным пользователем из таблицы Users. Это поле позволяет определить, к какому пользователю относится данная роль;
– Role (строковый тип данных): Название роли, которую занимает пользователь (например, «администратор», «пользователь»). Это поле определяет, какие действия пользователь может выполнять в системе.

Взаимосвязи между таблицами:

– Inventory ↔ Batches: Связь между полем Id таблицы Inventory и полем InventoryId таблицы Batches позволяет системе знать, какие партии

относятся к какому товару. Это обеспечивает легкий доступ к информации о партиях для каждого типа товара;

– Batches ↔ Expences: Связь между полем Id таблицы Batches и полем BatchId таблицы Expences позволяет отслеживать расходы, связанные с конкретными партиями. Обеспечивает целостность данных и возможность анализа затрат;
– Users ↔ UserRole: Связь между полем Id таблицы Users и полем UserId таблицы UserRole позволяет определять роли пользователей в системе. Необходимо для управления доступом и правами пользователей.

Важным аспектом является возможность гибкого управления данными через правильную структуру связей между таблицами. Это позволяет избежать дублирования информации, минимизировать ошибки и улучшить оперативное принятие решений на основе данных. В результате система становится более устойчивой, безопасной и адаптируемой к изменениям, что важно для ее стабильной и эффективной работы в долгосрочной перспективе.

Выводы по главе 2

Глава посвящена проектированию программного обеспечения для учета товарно-материальных ценностей в аналитических лабораториях. В ходе разработки был выбран подход, сочетающий объектно-ориентированное проектирование (ООП) и водопадную модель. Такое решение объясняется линейным процессом разработки, при котором каждая фаза завершает предыдущую, что подходит для систем, где требования заранее хорошо известны и стабильно задокументированы, как в случае учета товарно-материальных средств. Использование ООП-парадигмы позволяет разделить систему на модули, каждый из которых можно разрабатывать и тестировать последовательно, что повышает удобство планирования и сопровождения системы. Для логического моделирования использована методология UML. Структура проекта продемонстрирована на диаграммах деятельности, прецедентов, классов, развертывания.

Глава 3 Реализация и тестирование программного обеспечения для учёта товарно-материальных ценностей аналитических лабораторий

3.1 Реализация программного обеспечения для автоматизации учёта товарно-материальных ценностей аналитических лабораторий

Для выделения приоритетных задач и ключевых компонентов, в целях организации процесса работы над системой, следует описать дерево функций.

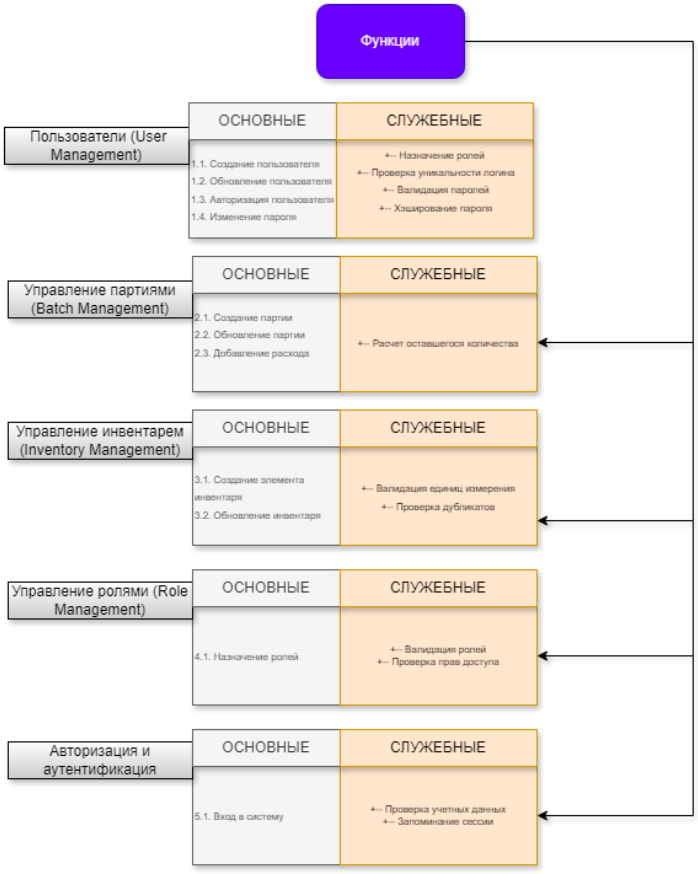


Рисунок 11 – Дерево функций проекта LabStorage

Дерево функций используется для структуризации и организации функциональных возможностей системы, разделяя их на основные и служебные (рисунок 11). Основные функции отвечают за выполнение

ключевых бизнес-процессов, таких как создание, управление данными и выполнение операций, в то время как служебные функции поддерживают эти процессы, обеспечивая корректность, безопасность и взаимодействие с системными компонентами. Такое разделение помогает лучше понять архитектуру приложения.

С учетом специфики поставленной задачи, целесообразно рассматривать не обособленные компоненты, а функционал в декомпозиции задач.

Этот формат демонстрирует иерархическую структуру основных и служебных функций, где каждая основная функция имеет соответствующие служебные функции, поддерживающие их выполнение.

3.1.1 Описание спецификации к API

Описание документации к API произведено на основании спецификации OpenAPI (OAS3) для сервера «LabStorage.Server». API предоставляет доступ к разделам с различными операциями, относящимися к аутентификации, управлению партиями, инвентаризацией и управлению пользователями.

Раздел Auth (Аутентификация) содержит несколько операций, связанных с аутентификацией пользователя (рисунок 12):

POST /api/Auth/LogIn – позволяет пользователю выполнить вход в систему, отправив данные для аутентификации (логин и пароль).

POST /api/Auth/LogOut – завершает сеанс пользователя, выполняя операцию выхода из системы.

GET /api/Auth/GetLoggedUser – возвращает информацию о текущем авторизованном пользователе, если сеанс активен.

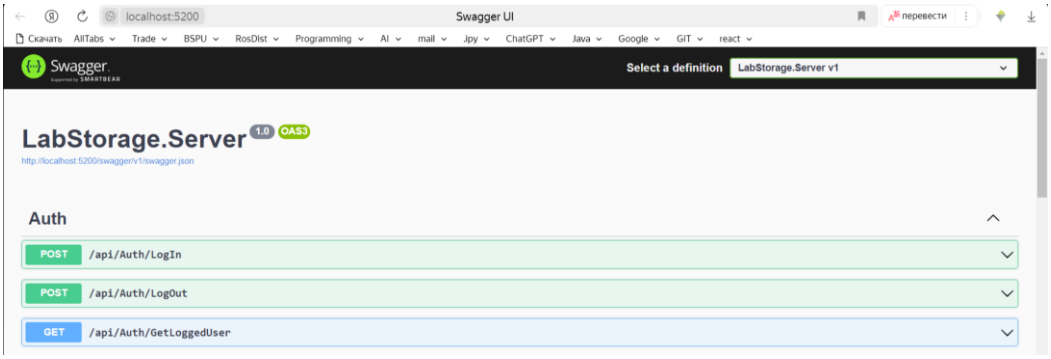


Рисунок 12 – Описание серверной части «Раздел Auth (Аутентификация)»

Раздел Batch (Партии) содержит операции, связанные с управлением партиями (Batch) (рисунок 13):

– GET /api/Batch/CheckExistence – проверяет наличие указанной партии в системе;
– GET /api/Batch/Get – получает данные о конкретной партии;
– GET /api/Batch/GetList – возвращает список всех партий, доступных в системе;
– POST /api/Batch/Create – создает новую партию в системе;
– POST /api/Batch/Update – обновляет информацию о существующей партии;
– POST /api/Batch/AddExpense – добавляет расходы к партии.

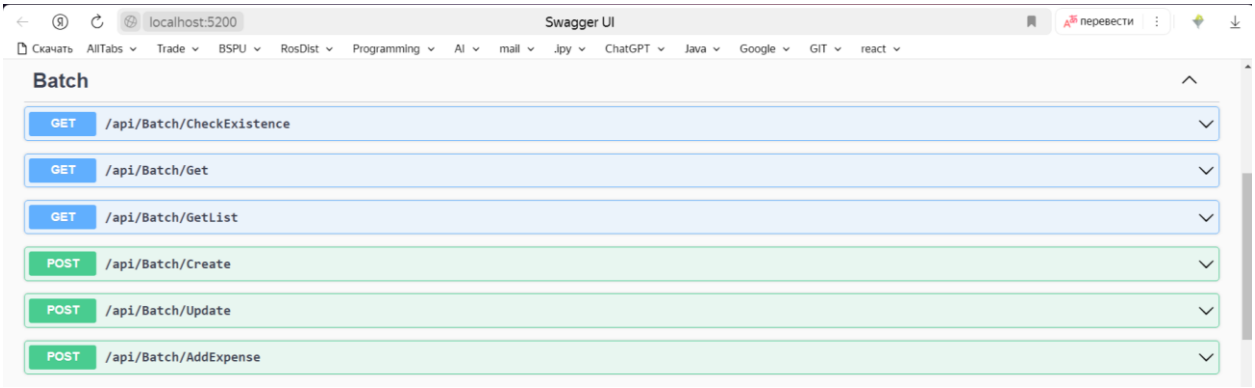


Рисунок 13 – описание серверной части. Раздел «Batch» (Партии)

В разделе Inventory (Список, ТМЦ) находятся операции, касающиеся управления инвентаризацией (рисунок 14):

– GET /api/Inventory/Get – возвращает информацию об одном элементе списка;
– GET /api/Inventory/GetList – получает список всех предметов списка;
– POST /api/Inventory/Create – добавляет новый элемент в список;
– POST /api/Inventory/Update – обновляет данные существующего объекта;
– POST /api/Inventory/Delete – удаляет элемент из списка.

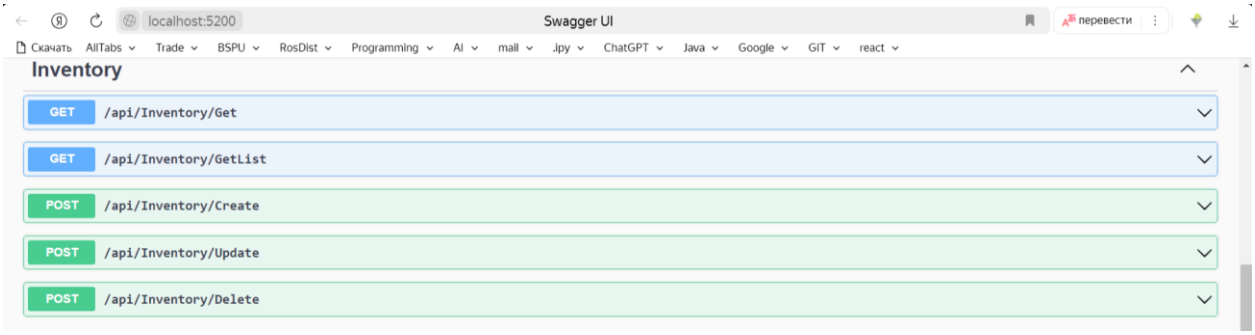


Рисунок 14 – Описание серверной части. Раздел «Inventory» (Список, ТМЦ)

Раздел Profile (Профиль). Метод POST /api/Profile/ChangePassword – позволяет пользователю изменить пароль в своём профиле (рисунок 15). Для этого пользователю нужно отправить старый пароль и новый.

Рисунок 15 – Описание серверной части. Раздел «Profile» (Профиль)

Раздел User (Пользователи) определяет операции для управления пользователями (рисунок 16):

– POST /api/User/Create – создаёт нового пользователя;
– GET /api/User/GetList – возвращает список всех пользователей;
– POST /api/User/Block – блокирует указанного пользователя (например, для ограничения доступа) ;
– POST /api/User/Unblock – разблокирует ранее заблокированного пользователя.

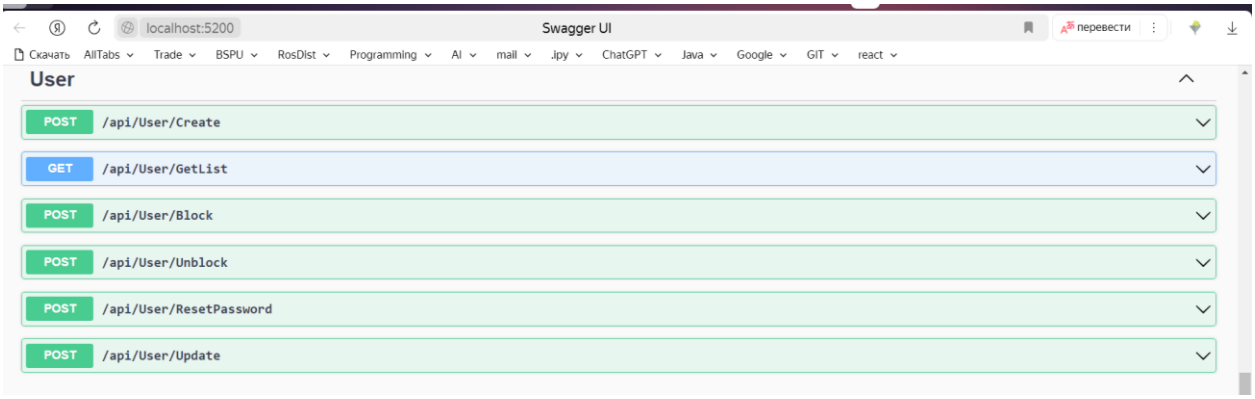


Рисунок 16 – Описание серверной части. Раздел «User» (Пользователи)

Каждый раздел API описывает разные аспекты системы. Разделы Auth и Profile связаны с аутентификацией и управлением профилем пользователя.

Разделы «Batch» и «Inventory» управляют партиями товаров и инвентаризацией соответственно. Раздел «User» предоставляет функции для управления пользователями системы: их создание, блокировка и получение списка.

Каждая операция поддерживает один из HTTP-методов, таких как GET (для получения данных), POST (для создания или изменения данных), и может быть доступна через указанный URL (например, /api/Batch/GetList для получения списка партий). Далее приведено подробное описание структур (JSON Schema Specification).

AddExpenseRequest:

– batchId (integer, формат $int32): Идентификатор партии, к которой

будет добавлен расход;

– quantity (number, формат $double): Количество расходов, которое добавляется к партии.

Эта структура (JSON Schema Specification) описывает добавление информации о расходах в определённую партию товаров или материалов.

BatchCreateRequest:

– inventoryId (integer, формат $int32): Идентификатор списка, к которому относится создаваемая партия;

– name (string, обязательное поле): Имя или название партии. Длина должна быть от 1 до 200 символов. minLength: Минимальная длина строки – 1 символ; maxLength: Максимальная длина строки – 200 символов;

– acceptanceDate (string, формат $date): Дата приёмки партии;

– manufactureDate (string, формат $date, nullable): Дата производства партии (может быть пустым);

– expiryDate (string, формат $date, nullable): Дата истечения срока годности партии (может быть пустым);

– quantity (number, формат $double): Количество единиц в партии.

Эта структура (JSON Schema Specification) используется для создания новой партии товаров или материалов в системе. Включает обязательные и необязательные даты, а также количество единиц товара.

BatchUpdateRequest:

– id (integer, формат $int32): Идентификатор существующей партии, которую необходимо обновить,

– name (string, обязательное поле): Обновлённое название партии. minLength: Минимальная длина строки – 1 символ; maxLength: Максимальная длина строки – 200 символов,

– acceptanceDate (string, формат $date): Обновлённая дата приёмки партии;

– manufactureDate (string, формат $date, nullable): Обновлённая дата

производства партии (может быть пустым);

– expiryDate (string, формат $date, nullable): Обновлённая дата истечения срока годности партии (может быть пустым);
– quantity (number, формат $double): Обновлённое количество единиц в партии.

Эта структура (JSON Schema Specification) используется для обновления данных о существующей партии, таких как название, количество, даты и идентификатор партии.

ChangePasswordRequest:

– userId (integer, формат $int32): Идентификатор пользователя, который меняет пароль;
– currentPassword (string, обязательное поле): Текущий пароль пользователя. minLength: Минимальная длина строки – 1 символ;
– newPassword (string, обязательное поле): Новый пароль пользователя. minLength: Минимальная длина строки – 1 символ;

Эта структура (JSON Schema Specification) используется для изменения пароля пользователя. Необходимо указать текущий и новый пароли.

CreateRequest:

– name (string, обязательное поле): Название элемента инвентаря. minLength: Минимальная длина строки – 1 символ;
– unit (string, обязательное поле): Единица измерения для элемента инвентаря. minLength: Минимальная длина строки – 1 символ.

Эта структура (JSON Schema Specification) используется для создания нового элемента инвентаря. Требует указания названия и единицы измерения.

LogInRequest:

– login (string, обязательное поле): Логин пользователя. minLength: Минимальная длина строки – 1 символ;
– password (string, обязательное поле): Пароль пользователя. minLength: Минимальная длина строки – 1 символ;
– rememberMe (boolean): Опциональный параметр, который указывает,

следует ли запомнить пользователя для последующих сеансов.

Эта структура (JSON Schema Specification) используется для входа пользователя в систему. Требуется указать логин и пароль, а также возможно установить флажок «Запомнить меня».

Role:

– integer (формат $int32) содержит перечисление возможных ролей пользователя в системе;
– Enum содержит список возможных значений ролей. Например, массив из 3 значений (например, администратор, пользователь и гость).

Эта структура (JSON Schema Specification) определяет роли пользователей, которые могут быть назначены в системе.

UpdateRequest:

– id (integer, формат $int32): Идентификатор элемента, который требуется обновить;
– name (string, nullable): Обновлённое название элемента (может быть пустым);
– unit (string, nullable): Обновлённая единица измерения элемента (может быть пустым).

Эта структура (JSON Schema Specification) используется для обновления данных существующего элемента инвентаря. Обновляются его название и единица измерения.

UserCreateRequest:

– login (string, обязательное поле): Логин нового пользователя. minLength: Минимальная длина строки – 1 символ;
– roles ([...]): Список ролей, назначаемых новому пользователю.

Эта структура (JSON Schema Specification) используется для создания нового пользователя в системе с указанием его логина и ролей.

UserUpdateRequest:

– id (integer, формат $int32): Идентификатор пользователя, которого

необходимо обновить;

– login (string, nullable): Обновлённый логин пользователя (может быть пустым);
– roles ([...]): Обновлённый список ролей пользователя.

Эта структура (JSON Schema Specification) используется для обновления информации о существующем пользователе, таких как его логин и роли.

Документация API, основанная на спецификации OpenAPI (OAS3), описывает сервер «LabStorage.Server» и его взаимодействие с различными разделами системы, включая аутентификацию, управление партиями, инвентаризацией и пользователями.

Раздел Auth содержит операции для входа, выхода и получения информации об авторизованном пользователе. Batch предоставляет API для управления партиями товаров, таких как создание, обновление и добавление расходов. Inventory охватывает операции для управления инвентаризацией, включая добавление, обновление и удаление элементов матценностей. Profile позволяет пользователям изменять свои пароли, а User управляет созданием, блокировкой и получением списка пользователей.

Для каждого раздела предусмотрены схемы запросов и ответов в формате JSON, которые описывают параметры, необходимые для выполнения различных операций, таких как создание партии, добавление расходов, изменение пароля или управление пользователями.

3.1.2 Описание серверной части

Класс ApiController – родительский абстрактный класс. Поле loggedUser используется для кеширования состояния. Метод GetLoggedUser возвращает поле loggedUser. Метод доступен дочерним классам. ApiError, ApiOk – вспомогательные методы, альтернатива Success TRUE/FALCE.

Класс AuthController – наследуется от ApiController. Метод CreateCookie – приватный метод, вспомогательный, для создания «куков». Метод GetLoggedUser возвращает поле loggedUser. Метод Login – логинизация,

метод Logout – разлогинивание. LoginRequest – структура, которую необходимо передать при взаимодействии с клиентом.

Класс InventoryController – содержит методы CRUD и включает структуру, которая передается при взаимодействии с клиентом (поля, необходимые для парсинга поступающей от клиента информации).

Класс BatchController – содержит методы для работы с расходом по позициям: AddExpence, CheckExistance, Create, Get, GetList, Update.

Класс UserController – для работы с состоянием юзера. Содержит методы блокировки/разблокировки, создания пользователя, уточнения состояния, сброса пароля: Block, Create, GetList, ResetPassword, Unblock, Update.

Класс ProfileController – содержит метод для изменения пароля: ChangePassword. Программный код некоторых компонентов серверной части приложения приведен на рисунках А.1 – А.2, Приложение А.

Классы контроллеров в системе разделены по функциям. ApiController выступает базовым классом для других контроллеров, предоставляя методы для работы с авторизацией и ответами API. AuthController отвечает за аутентификацию пользователей, InventoryController – за управление инвентарём, BatchController – за работу с партиями и расходами, UserController – за управление пользователями (блокировка, создание, сброс пароля), а ProfileController – за смену пароля пользователя.

Система включает набор классов, каждый из которых отвечает за выполнение определённых функций. Базовый класс предоставляет общие методы для работы с авторизацией и API-ответами. Другие классы управляют аутентификацией, инвентарём, партиями, пользователями и изменением паролей. Все контроллеры реализуют операции для создания, обновления, получения и удаления данных, а также обеспечивают поддержку управления пользователями и учётными записями.

3.1.3 Описание клиентской части

App.html – файл с базовой HTML – разметкой для приложения на Svelte. Используются плейсхолдеры sveltekit.head, sveltekit.body для динамического рендеринга содержимого при помощи фреймворка.

Клиентские точки навигации доступны по адресу LabStorage\LabStorage.Client\src\routes\

Пользовательские стили доступны по адресу LabStorage\LabStorage.Client\src\lib\css\site.css

Ввиду использования самостоятельного сервера на ASP.Net, SSR отключен. Ниже указывается путь, от каталога src, размещенного относительно корневого каталога: LabStorage\LabStorage.Client\src

Форма авторизации и реализация запроса доступны по адресу \routes\login\+page.svelte

При переходе по NavBar реализована проверка авторизации, в файле \routes\(authed)\+layout.js. Асинхронная функция load вызывается автоматически при рендеринге страницы или компонента. Далее реализуется GET – запрос на сервер для получения данных о текущем пользователе. В последующем проверяется успешность логинизации пользователя с решением о предоставлении доступа:

```
export async function load({ fetch }) {
    const res = await sendRequest(fetch, 'GET', '/api/auth/getLoggedUser');
    const data = await getSuccessDataOrThrow500(res);
    return { user: data };
}
```

Загрузка данных осуществляется посредством функции load, \routes\(authed)\inventory\list\+page.js:

```
import { sendRequest, getSuccessDataOrThrow500 } from '$lib/js/request.js';

export async function load({ fetch }) {
    const res = await sendRequest(fetch, 'GET', '/api/inventory/getList');
```

```
    const data = await getSuccessDataOrThrow500(res);
    return { item: data };
}
```

Формы для поиска партии, внесения ТМЦ в базу данных, внесения расхода материальных средств, доступны по соответствующим адресам:

- \routes\(authed)\batch\+page.svelte,
- \routes\(authed)\inventory\[id]\AddBatch.svelte,
- \routes\(authed)\batch\[id]\AddExpense.svelte.

Программный код некоторых компонентов клиентской части приложения приведен на рисунках А.3 – А.4, Приложение А.

Файл App.html содержит основную разметку для Svelte с динамическим рендерингом. Навигация и стили расположены в отдельных директориях. SSR отключен, так как используется сервер на ASP.Net. Реализованы формы для авторизации и работы с инвентарем с проверкой доступа. Данные загружаются асинхронными запросами на сервер для получения информации о пользователе и инвентаре.

3.1.4 Демонстрация работы приложения

Дефолтные настройки подразумевают наличие суперпользователя – администратора, с логином admin и одноименным паролем. Запуск серверной части осуществляется посредством выполнения команды «dotnet run» в PowerShell, запущенной из серверной папки проекта (рисунок 17).

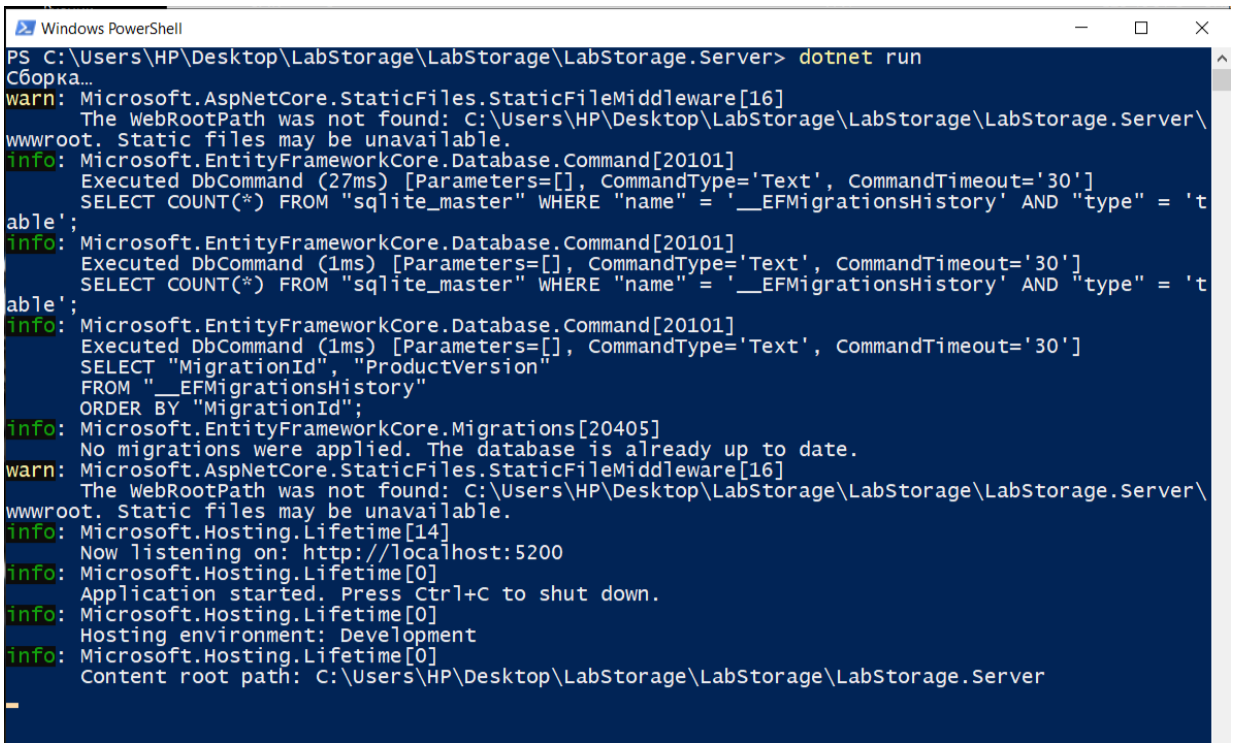


Рисунок 17 – Запуск серверной части приложения

Аналогично запуску сервера, из каталога с клиентской частью производится запуск клиента, посредством выполнения команды bun --bun run dev (рисунок 18).

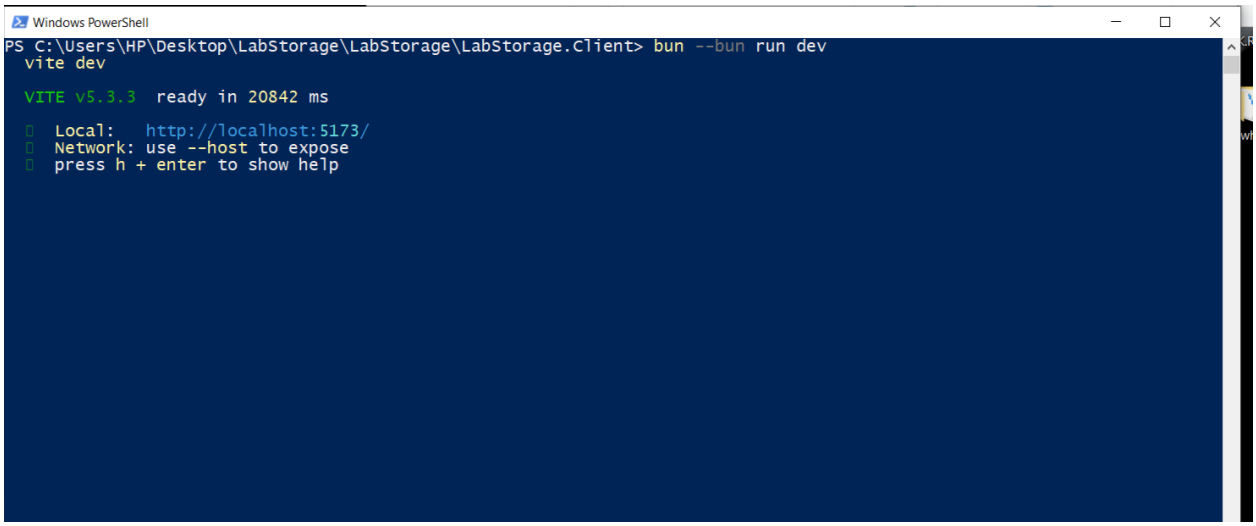


Рисунок 18 – Запуск клиентской части приложения

В открывшемся окне браузера, к заполнению доступна форма авторизации (рисунок 19).

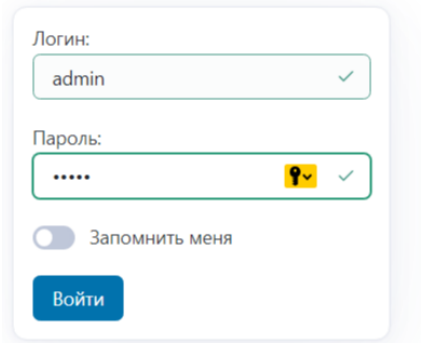


Рисунок 19 – Форма авторизации пользователя

В случае успешной авторизации, происходит перенаправление по адресу личного профиля пользователя (рисунок 20).

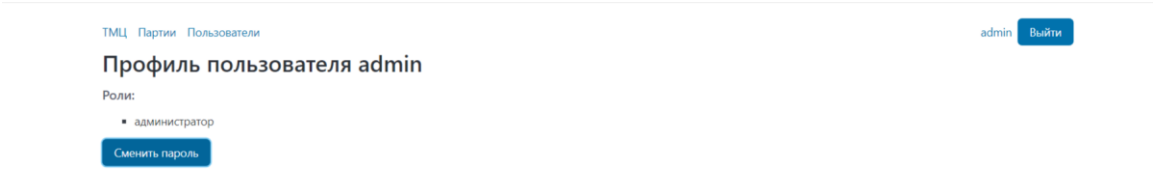


Рисунок 20 – Профиль с учетом распределения ролей

Среди доступного функционала, из профиля, следует выделить систему смены пароля (рисунок 21).

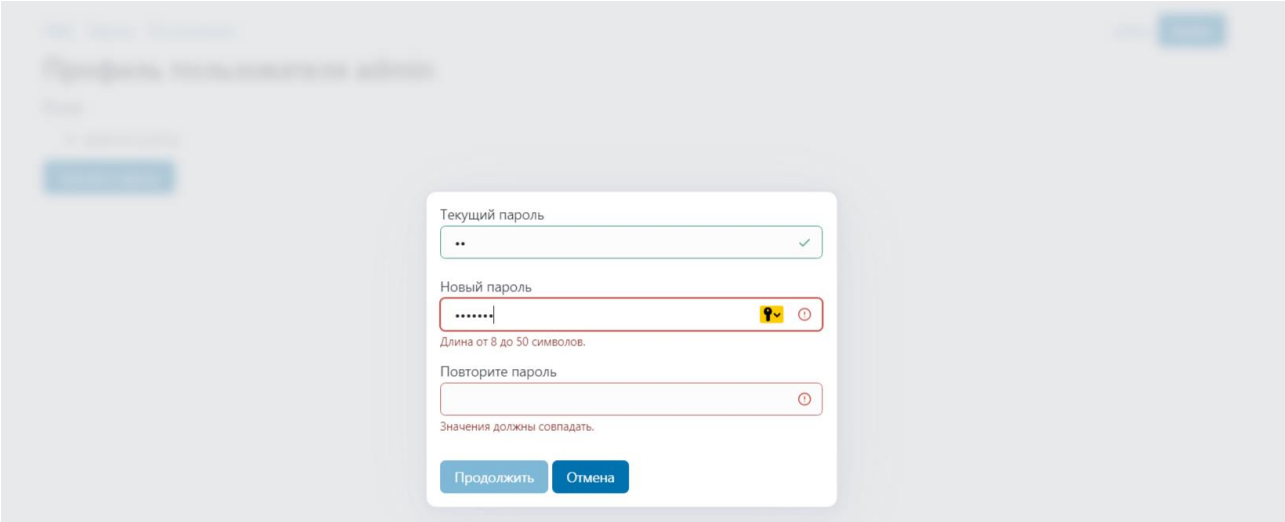


Рисунок 21 – Изменение пароля

Администратор может производить назначение и переназначение ролей в соответствии с заранее принятой политикой разграничения функционала пользователей (Рисунок 22).

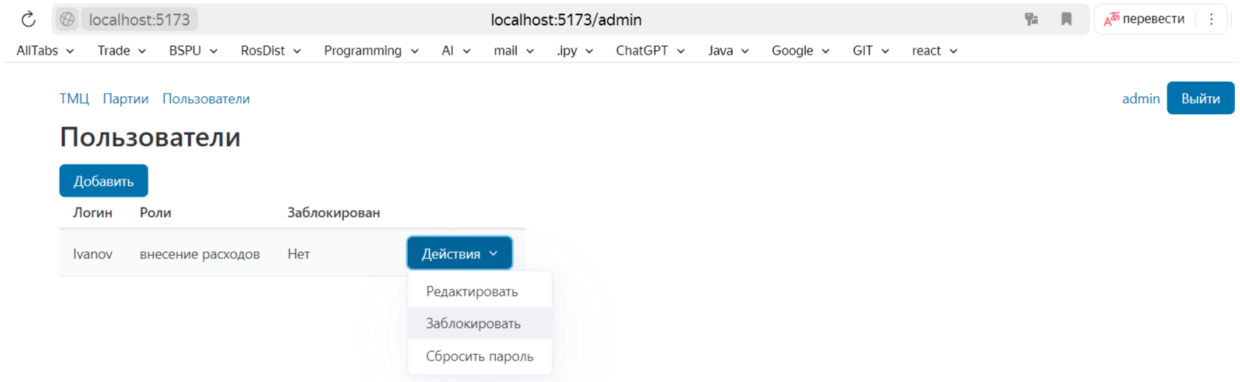


Рисунок 22 – Администрирование пользователей с учетом назначения ролей

Регистрация товарно-материальных ценностей представлена на рисунке 23. Доступно добавление наименования и единиц измерения по позициям.

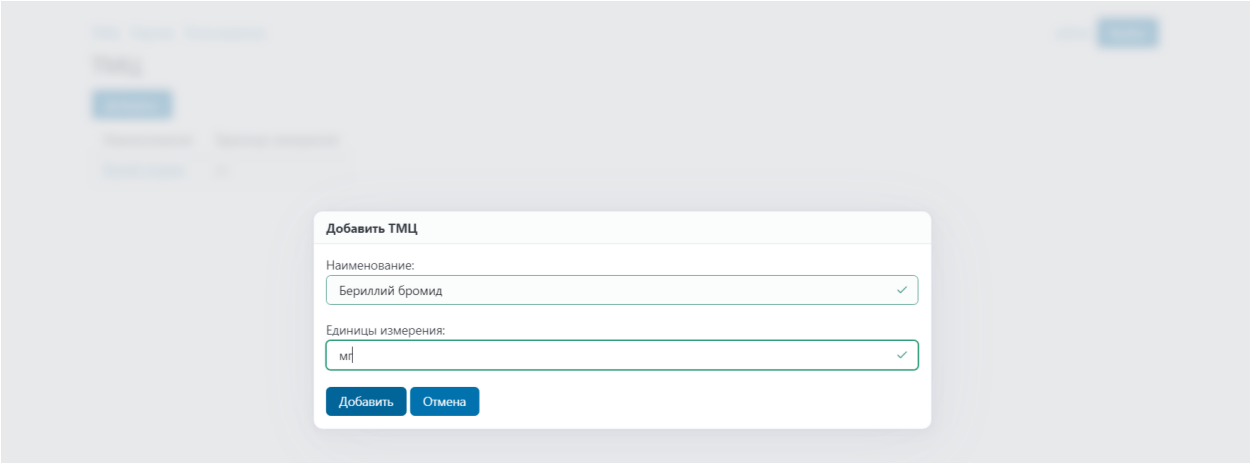


Рисунок 23 – Добавление ТМЦ

Дальнейшее редактирование партий товара представлено на рисунке 24.

Рисунок 24 – Редактирование партий

Карта партии представлена на рисунке 25. Доступны поля для заполнения: «наименование», «дата принятия», «дата изготовления», «годен до», «количество».

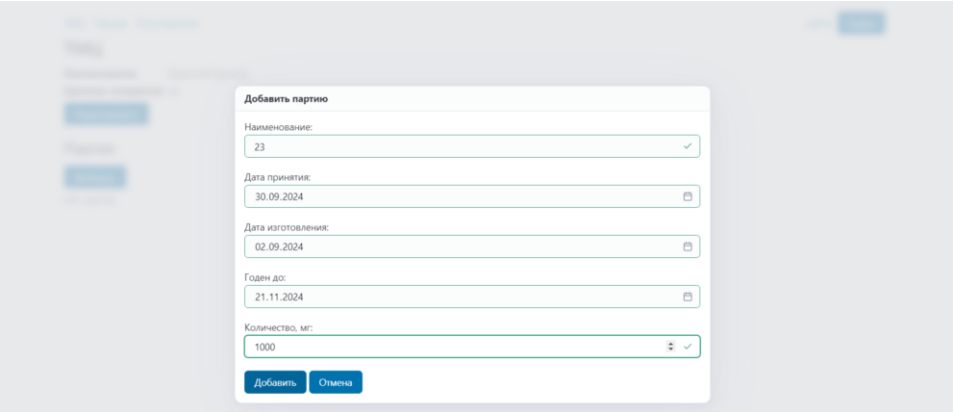


Рисунок 25 – Окно редактирования партий ТМЦ

Принятие изменений, при заполнении требуемых полей, после отправки данных в базу данных, переводит на страницу с общим списком товарно-материальных ценностей. Список приведен на рисунке 26.

Рисунок 26 – Рабочая таблица

Переход по ссылке «Партия» позволяет осуществить редактирование данных по позиции или же осуществить расход, как показано на рисунке 27.

Рисунок 27 – Контроль расходов

Администрирование подразумевает добавление пользователей в систему. На рисунке 28 показан интерфейс, наделяющий администратора данной функцией.

Рисунок 28 – Добавление пользователей в систему

Создание пользователя в системе, позволяет администратору на этапе создания в системе, наделить пользователя следующими ролями: редактирование ТМЦ, редактирование партий или внесение расходов.
Администратор по умолчанию представляет собой суперпользователя, в иерархической системе ролирования занимает ключевую роль, наделен всеми перечисленными ролями (рисунок 29).

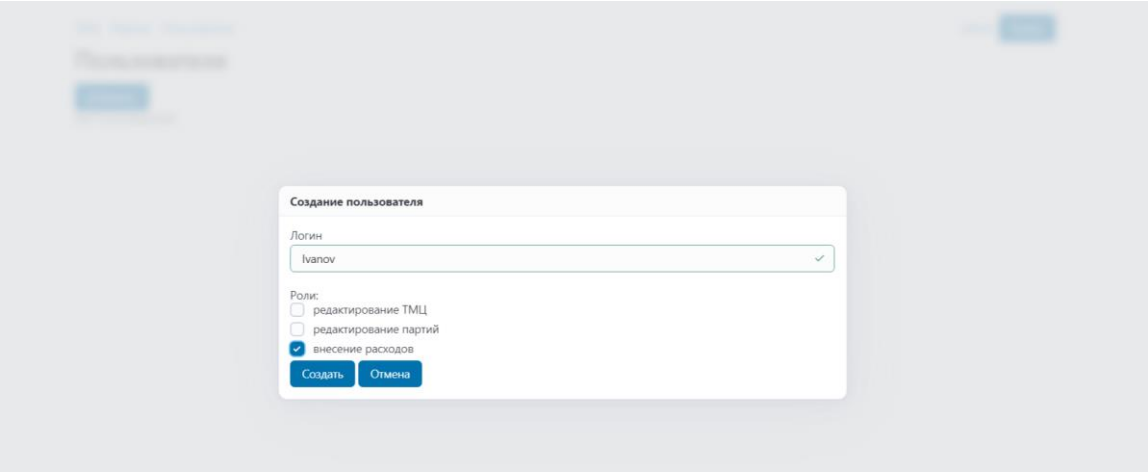


Рисунок 29 – Назначение ролей пользователей

При создании пользователя, автоматически генерируется пароль (рисунок 30).

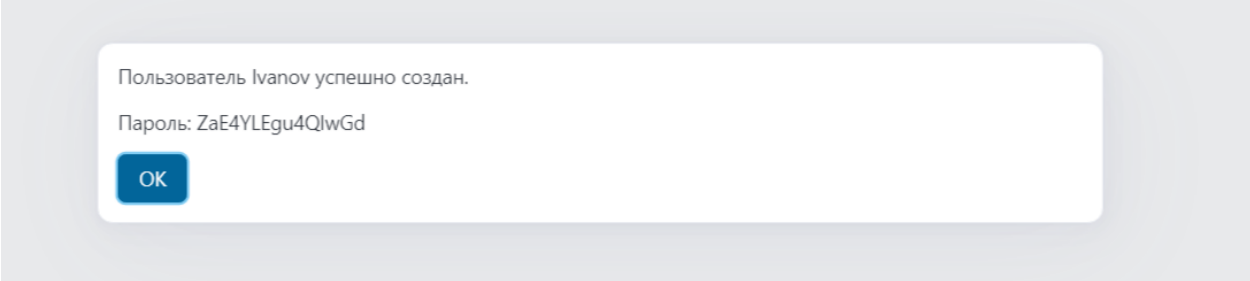


Рисунок 30 – Генерация временного пароля

В дальнейшем происходит безусловный переход на страницу профиля вновь созданного пользователя (рисунок 31).

Рисунок 31 – Профиль пользователя с учетом ролирования

Учитывая порядок назначения ролей администратором, можно пронаблюдать доступный функционал (рисунок 32).

Рисунок 32 – Ограничение функционала пользователей с учетом распределения ролей

Создание пользователя в системе позволяет администратору назначить ему определенные роли, такие как редактирование ТМЦ, редактирование партий или внесение расходов. Администратор, обладая полномочиями суперпользователя, имеет все роли по умолчанию. При создании пользователя система автоматически генерирует пароль, после чего происходит переход на страницу профиля нового пользователя, где можно пронаблюдать назначенные администратором роли и доступный функционал.

3.2 Тестирование программного обеспечения для автоматизации учёта товарно-материальных аналитических лабораторий

В проекте были использованы различные типы тестов для проверки стабильности и корректности работы приложения. Тестирование включает как юнит-тесты, так и интеграционные тесты, затрагивающие функциональные проверки и проверку бизнес-логики. Каждый из этих типов тестов выполняет свою роль в обеспечении качества кода.

В общей сложности реализовано 3 функциональных теста (7 методов, 17 сценариев).

В тестах используются реальные сервисы и контроллеры, что позволяет проверить их взаимодействие с базой данных. Каждый тест выполняется с изолированным состоянием, что позволяет избежать зависимостей между тестами. Эти подходы помогают тестировать систему с разных сторон: правильность работы с базой данных, логика бизнес-операций, обработка ошибок.

Юнит-тесты фокусируются на проверке отдельной логики приложения, изолируя каждый компонент, чтобы гарантировать его корректную работу без зависимости от других частей системы. В данном проекте юнит-тесты используются для проверки отдельных методов, классов и компонентов.

Тесты в классе PasswordGeneratorTests:

– ValidatePassword_ValidPassword_ReturnTru;
– ValidatePassword_InvalidPassword_ReturnFalse;
– ComputeHash_ShouldComputeCorrectly.

Первые два теста проверяют валидность паролей в классе PasswordGenerator. Эти тесты независимы от базы данных или других компонентов, так как проверяется только логика работы с паролем, и они могут быть выполнены без внешних зависимостей.

Третий тест проверяет метод хеширования пароля в классе PasswordGenerator. Здесь тестируются только вычисления и соответствие заранее определённым значениям хеш-суммы.

Эти тесты выполняются на уровне отдельных методов и компонентов, не взаимодействуя с внешними сервисами или базой данных. Их задача – удостовериться в правильности выполнения бизнес-логики.

Интеграционные тесты проверяют взаимодействие различных компонентов системы, например, контроллеров с базой данных или с внешними сервисами. Они тестируют, как компоненты работают вместе в реальных условиях работы системы.

Интеграционные тесты в классе ProfileControllerTests:

– ChangePassword_ValidCurrentPassword_ReturnSuccess;

– ChangePassword_InvalidCurrentPassword_ReturnError;
– ChangePassword_InvalidNewPassword_ReturnError.

Первый метод тестирует успешное изменение пароля, когда текущий пароль пользователя введён правильно. Ожидается, что метод вернёт статус «200 OK» и успешный результат.

Второй метод проверяет ситуацию, когда введён неправильный текущий пароль. В этом случае метод должен вернуть ошибку с кодом InvalidPassword.

Третий метод тестирует случай, когда новый пароль не соответствует требованиям безопасности (например, слишком короткий). В этом случае метод должен вернуть ошибку с кодом Validation.

Отдельно стоит отметить, что в тестах ProfileControllerTests проверяется не только успешность операции, но и корректность обработки ошибок, таких как неправильный текущий пароль или слишком слабый новый пароль. Это гарантирует, что приложение отвечает на запросы в соответствии с заданной логикой и бизнес-правилами.

Класс BatchControllerTests содержит один метод – AddExpense_ReturnSuccess. Проверке подлежит метод AddExpense контроллера BatchController, который добавляет расход для определённой партии ТМЦ. Этот тест является интеграционным, поскольку взаимодействует с базой данных через контекст DbContext и проверяет не только логику контроллера, но и корректность обновлений в базе данных.

Результат успешного тестирования наглядно демонстрирует окно инструмента «TestExplorer», интегрированной среды разработки Visual Studio (рисунок 33).

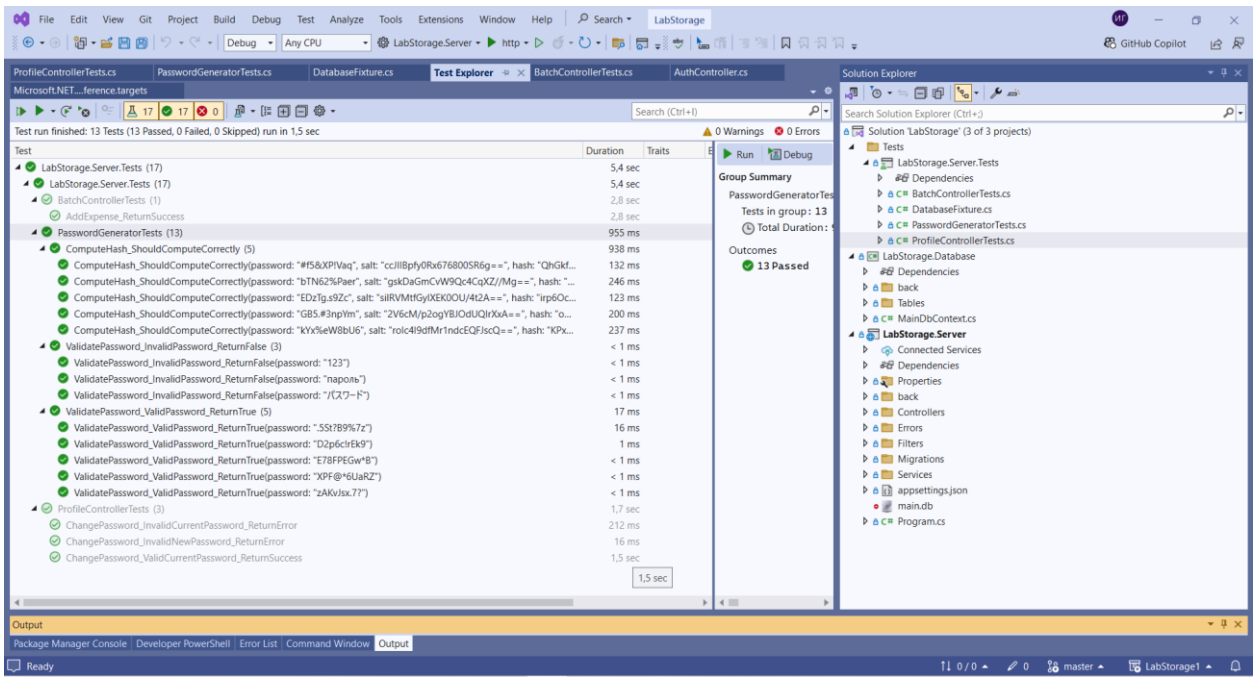


Рисунок 33 – Результат успешного каскадного тестирования

Выводы по главе 3

В главе рассмотрены функционал системы, спецификация API. Приведено описание серверной и клиентской частей приложения. Дано описание работы приложения. В процессе тестирования использовались методы юнит-тестирования для проверки отдельных функциональных блоков и интеграционные тесты для проверки взаимодействия компонентов системы. Также в проекте учтены аспекты, связанные с безопасностью, такими как хеширование паролей, и обработкой ошибок.

Заключение

В рамках бакалаврской работы была разработана система автоматизированного учета товарно-материальных ценностей для аналитических лабораторий, с использованием современных технологий.

Осуществлен анализ требований и особенностей учета ТМЦ в аналитической лаборатории.

Дано обоснование выбора стека технологий, используемых для разработки системы. Реализация стека технологий ASP.Net Core : Svelte JS : SQLite обеспечивает надежность и гибкость приложения, при должной производительности.

Сделан выбор в пользу клиент-серверной архитектуры.

Произведена разработка клиентской и серверной частей приложения, осуществлена интеграция базы данных.

В системе организована политика избирательного управления доступом, основанная на распределении ролей пользователей, что обеспечивает защиту данных и управление правами доступа. Это создает условия для безопасного и удобного взаимодействия с системой, позволяя пользователям выполнять необходимые операции в зависимости от их ролей и полномочий.

На основании данных отладки и тестирования, показано: разработанное программное обеспечение обладает необходимыми и достаточными техническими характеристиками и функционалом для обеспечения контроля движения товарно-материальных ценностей, начиная с этапа их регистрации, и предоставляет гибкое управление доступом.

Список используемой литературы и используемых источников

1. Буч Г., Рамбо Д., Джекобсон А. Язык UML Руководство пользователя – С-П.: Издательство «Питер», 2010. 432 с.

2. Грекул В. И., Денищенко Г. Н., Коровкина Н. Л. Проектирование информационных систем : учебное пособие. М. : Интернет-Университет Информационных Технологий (ИНТУИТ), Ай Пи Ар Медиа, 2020. 299 с.

3. Грофф, Джеймс Р., Вайнберг, Пол Н., Оппель, Эндрю Дж. SQL. Полное руководство [Текст] : [пер. с англ.] / Джеймс Р. Грофф, Пол Н. Вайнберг, Эндрю Дж. Оппель - 3-е изд. – М. : ООО «И.Д. Вильямс», 2015. 960 с. ил. – Перевод изд.: SQL, The Complete Reference / J. Groff, P. Weinberg, A. Oppel, The Mc-Grow-Hill Companies – 2000 экз. – ISBN 978-5-8459-1654-9 (в пер.).

4. Кантелон М.. Node.js в действии / М. Кантелон, М. Хартер, Т. Головайчук, Н. Райлих - СПб.: Питер, 2018. 548 с. ISBN 978-5-496-01079-5

5. Кузнецов С. Д. Основы баз данных. / С. Д. Кузнецов – М. : Бином. Лаборатория знаний, Интернет-университет информационных технологий, 2017. 488 с.

6. Котляров В. П. Основы тестирования программного обеспечения: учеб. пособие / В. П. Котляров. – М.: ИНТУИТ, 2016. 335 с.

7. Леоненков А. В. Объектно-ориентированный анализ и проектирование с использованием UML и IBM Rational Rose : учебное пособие. М. : Интернет-Университет Информационных Технологий (ИНТУИТ), Ай Пи Ар Медиа, 2020. 317 с.

8. Любимов Е. В. Управление бизнес-процессами: моделирование: учебное пособие / Е. В. Любимов. – Владивосток: Изд-во ВГУЭС, 2018. 68 с.

9. Пирогов В. Ю. Информационные системы и базы данных: Организация и проектирование. СПб. : БХВ-Петербург, 2020. 254 с.

10. Репин В. В. Бизнес-процессы. Моделирование, внедрение, управление. М. : Манн, Иванов и Фербер, 2018. 177 с.

11. Репин В. В., Елиферов В. Г. Процессный подход к управлению. Моделирование бизнес–процессов М. : Манн, Иванов и Фербер, 2021. 544 с.

12. Розенберг Д., Скотт К. Применение объектного моделирования с использованием UML и анализ прецедентов.: Пер. с англ. М.: ДМК Пресс, 2002.

13. Советов Б. Я. Архитектура информационных систем: учебник для студентов учреждений высшего профессионального образования. М. : Издательский центр Академия, 2012. 288 с.

14. Троелсен Эндрю. Язык программирования C# 5.0 и платформа .NET 4.5, 6-е изд. : Пер. с англ. – М. : ООО “И.Д. Вильямс”, 2013. 1312 с.

15. Фленов М.Е. Библия C#. – 3-е изд., перераб. и доп. – СПб.: БХВ Петербург, 2016. 560 с.

16. Хорев П.Б. Объектно-ориентированное программирование с примерами на C#. – Москва: Форум, Инфра-М, 2016. 200 с.

17. Bun is an all-in-one runtime for JavaScript and TypeScript apps [Электронный ресурс] - URL: https://bun.sh/ (дата обращения: 20.09.2024)

18. IDEF0. Знакомство с нотацией и пример использования. [Электронный ресурс] // URL: https://www.trinion.org/blog/idef0-znakomstvo-s-notaciey-i-primer-ispolzovaniya (дата обращения: 20.09.2024)

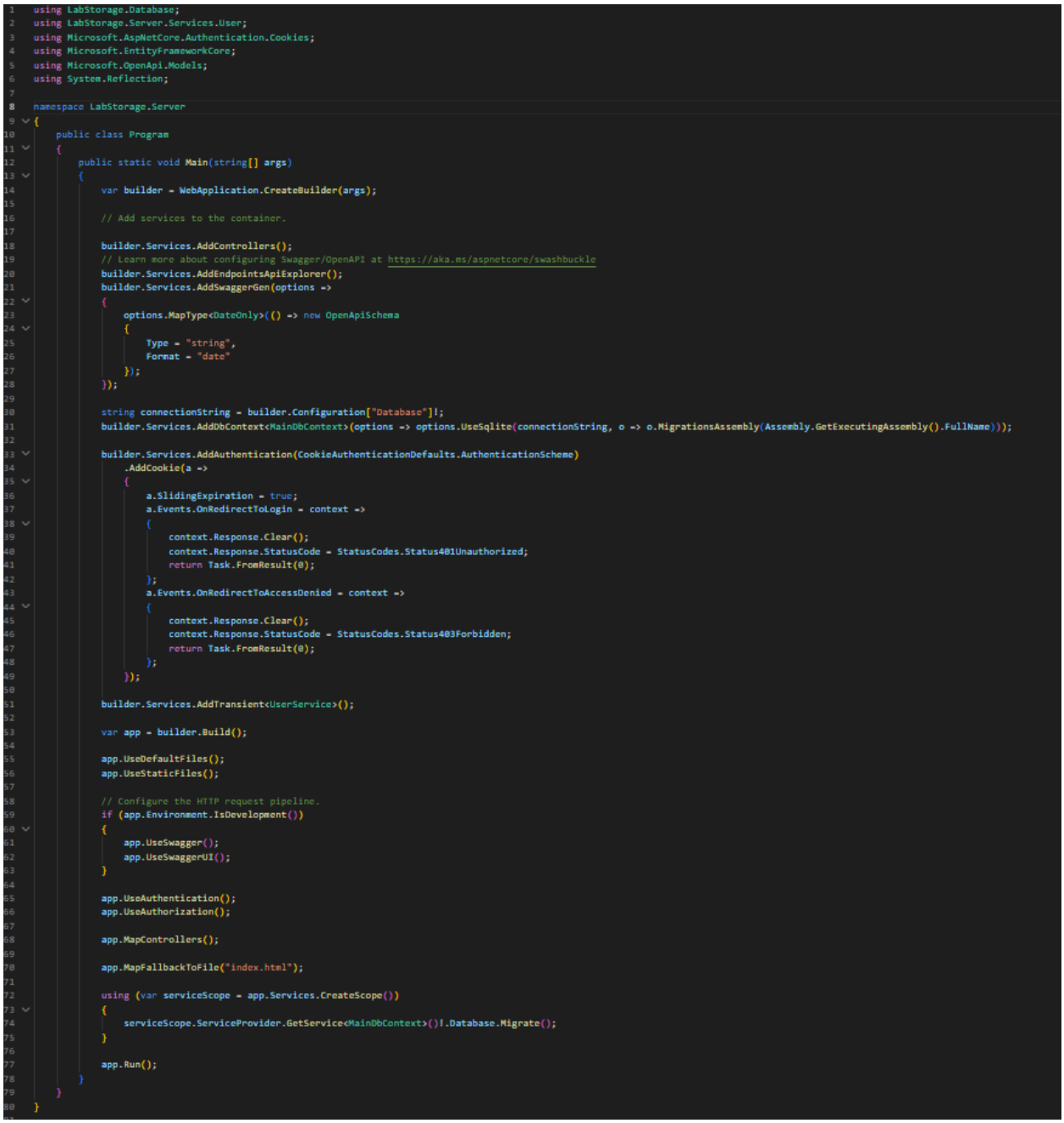
19. React The library for web and native user interfaces [Электронный ресурс] - URL: https://react.dev/ (дата обращения: 20.09.2024)

20. React – JavaScript-библиотека для создания пользовательских интерфейсов // Знакомство с JSX – React. - [Электронный ресурс] - URL: https://ru.reactjs.org/docs/introducing-jsx.html (дата обращения: 20.09.2024)

21. SQLite Documentation [Электронный ресурс] - URL: https://www.sqlite.org/docs.html (дата обращения: 20.09.2024)

Приложение А

Листинги некоторых программных модулей



```
using LabStorage.Database;
using LabStorage.Server.Services.User;
using Microsoft.AspNetCore.Authentication.Cookies;
using Microsoft.EntityFrameworkCore;
using Microsoft.OpenApi.Models;
using System.Reflection;

namespace LabStorage.Server
{
    public class Program
    {
        public static void Main(string[] args)
        {
            var builder = WebApplication.CreateBuilder(args);

            // Add services to the container.

            builder.Services.AddControllers();
            // Learn more about configuring Swagger/OpenAPI at https://aka.ms/aspnetcore/swashbuckle
            builder.Services.AddEndpointsApiExplorer();
            builder.Services.AddSwaggerGen(options =>
            {
                options.MapType<DateOnly>(() => new OpenApiSchema
                {
                    Type = "string",
                    Format = "date"
                });
            });

            string connectionString = builder.Configuration["Database"]!;
            builder.Services.AddDbContext<MainDbContext>(options => options.UseSqlite(connectionString, o => o.MigrationsAssembly(Assembly.GetExecutingAssembly().FullName)));

            builder.Services.AddAuthentication(CookieAuthenticationDefaults.AuthenticationScheme)
                .AddCookie(a =>
                {
                    a.SlidingExpiration = true;
                    a.Events.OnRedirectToLogin = context =>
                    {
                        context.Response.Clear();
                        context.Response.StatusCode = StatusCodes.Status401Unauthorized;
                        return Task.FromResult(0);
                    };
                    a.Events.OnRedirectToAccessDenied = context =>
                    {
                        context.Response.Clear();
                        context.Response.StatusCode = StatusCodes.Status403Forbidden;
                        return Task.FromResult(0);
                    };
                });

            builder.Services.AddTransient<UserService>();

            var app = builder.Build();

            app.UseDefaultFiles();
            app.UseStaticFiles();

            // Configure the HTTP request pipeline.
            if (app.Environment.IsDevelopment())
            {
                app.UseSwagger();
                app.UseSwaggerUI();
            }

            app.UseAuthentication();
            app.UseAuthorization();

            app.MapControllers();

            app.MapFallbackToFile("index.html");

            using (var serviceScope = app.Services.CreateScope())
            {
                serviceScope.ServiceProvider.GetService<MainDbContext>()!.Database.Migrate();
            }

            app.Run();
        }
    }
}
```

Рисунок А.1 – Листинг кода файла Program.cs

```cs
using LabStorage.Database;
using LabStorage.Server.Errors;
using LabStorage.Server.Services.User;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using System.Security.Claims;

namespace LabStorage.Server.Controllers.Api
{
    [ApiController]
    [Authorize]
    [CheckAccess]
    [Route("/api/[controller]/[action]")]
    public abstract class ApiController(MainDbContext mainDbContext, UserService userService) : ControllerBase
    {
        protected MainDbContext MainDbContext = mainDbContext;
        protected UserService UserService = userService;
        private LoggedUser? loggedUser = null;

        protected IActionResult ApiOk(object? data = null)
        {
            return Ok(new ApiOkResult(data));
        }

        protected IActionResult ApiError(int code, object? error)
        {
            return Ok(new ApiErrorResult(new ApiErrorData(code, error)));
        }

        protected IActionResult ApiError(Error code, object? error)
        {
            return ApiError((int)code, error);
        }

        protected IActionResult ApiError(ApiError error) => ApiError(error.Code, error.Message);

        protected async Task<LoggedUser?> GetLoggedUser()
        {
            if (loggedUser != null) return loggedUser;
            int.TryParse(HttpContext.User.FindFirstValue("Id"), out int id);
            return loggedUser = await UserService.GetLoggedUser(id);
        }

        public record ApiOkResult(object? Data) { public bool Success { get; } = true; }
        public record ApiErrorResult(ApiErrorData Error) { public bool Success { get; } = false; }
        public record ApiErrorData(int Code, object? Message);
    }
}
```

Рисунок А.2 – Листинг кода файла ApiController.cs

routes > (authed) > admin > +page.svelte > script

```svelte
<script>
    import CreateUser from './CreateUser.svelte';
    import UserRow from './UserRow.svelte';
    export let data;

    let createUserDialog;
</script>

<h1>Пользователи</h1>

<button on:click={createUserDialog.showModal()}>Добавить</button>
<CreateUser bind:this={createUserDialog} />

{#if data.users.length > 0}
    <table class="striped">
        <thead>
            <tr>
                <th>Логин</th>
                <th>Роли</th>
                <th>Заблокирован</th>
                <th></th>
            </tr>
        </thead>
        <tbody>
            {#each data.users as item}
                <tr>
                    <UserRow user={item} />
                </tr>
            {/each}
        </tbody>
    </table>
{:else}
    <p>Нет пользователей.</p>
{/if}

```

Рисунок А.3 – Листинг кода файла +page.svelte, раздел «admin»

routes > (authed) > batch > +page.svelte > script > find

```svelte
<script>
    import { goto } from '$app/navigation';
    import { sendRequest } from '$lib/js/request.js';
    import ErrorModal from '$lib/components/ErrorModal.svelte';

    let errorDialog;

    let id;
    let exists = true;

    async function find() {
        const res = await sendRequest(fetch, 'GET', `/api/batch/checkExistence?id=${id}`);
        if (res.status === 200) {
            const body = await res.json();
            if (body.success) {
                if (body.data) {
                    goto(`/batch/${id}`)
                } else {
                    exists = false;
                }
            } else {
                errorDialog.showModal();
            }
        } else {
            errorDialog.showModal();
        }
    }
</script>

<h1>Поиск партии</h1>

<form style="max-width:100px;">
    <label>
        ID партии
        <input type="number" min="1" on:input={() => (exists = true)} bind:value={id} />
    </label>
</form>

<button on:click={find}>Найти</button>

{#if !exists}
    <p>
        <small class="color-red">Партия не найдена</small>
    </p>
{/if}

<ErrorModal bind:this={errorDialog} />
```

Рисунок А.4 – Листинг кода файла +page.svelte, раздел «batch»