СОДЕРЖАНИЕ

ТЕРМИНЫ, ОПРЕДЕЛЕНИЯ И СОКРАЩЕНИЯ

Фреймворк — программная платформа, определяющая структуру программной системы; программное обеспечение, облегчающее разработку и объединение разных компонентов большого программного проекта.

API (*Application Programming Interface*) — это программный интерфейс приложений, набор инструкций, который позволяет разным приложениям взаимодействовать друг с другом.

SQL (*Structured Query Language*) — декларативный язык программирования, применяемый для создания, модификации и управления данными в реляционной базе данных, управляемой соответствующей системой управления базами данных.

SDK (*Software Development Kit*) — набор инструментов для разработки программного обеспечения в одном устанавливаемом пакете.

IDE (*Integrated Development Environment*) — комплекс программных средств, используемый программистами для разработки программного обеспечения (ПО). Включает в себя: текстовый редактор, транслятор, средства автоматизации сборки.

Бэкэнд (англ. *Back-end*) — серверная сторона сайта или приложения, которая отвечает за его функционирование и хранение данных.

СУС — система управления складом.

СУЗ — система управления запасом.

ВВЕДЕНИЕ

Современная бизнес-среда характеризуется жесткой конкуренцией и быстро меняющимися требованиями клиентов. В результате предприятия постоянно ищут пути повышения эффективности и прибыльности своей деятельности. Одной из областей, которая может оказать значительное влияние на успех предприятия, является система управления запасами. Эффективное управление запасами может помочь предприятиям минимизировать риск возникновения товарных запасов, сократить расходы на хранение и заказы, а также повысить уровень удовлетворенности клиентов.

Управление запасами – это сложный процесс, требующий тщательного планирования, организации и контроля. В современной розничной торговле малые предприятия сталкиваются с проблемами эффективного управления запасами из-за ограниченности ресурсов и отсутствия сложных систем управления запасами. Поэтому существует потребность в доступных и удобных системах управления запасами, которые могут помочь малому бизнесу повысить эффективность работы и прибыльность. Целью данного диплома является удовлетворение этой потребности путем разработки системы управления запасами (СУЗ)для небольшого магазина с одним складом.

Целью данного диплома является программирование WMS, которая автоматизирует и оптимизирует процесс управления запасами, повышая операционную эффективность и прибыльность магазина. WMS будет разработана с учетом специфических потребностей магазина для обеспечения точного обновления запасов в режиме реального времени. Целью данного диплома является разработка удобной и эффективной WMS, которая может быть легко использована сотрудниками магазина, включая менеджера магазина, продавцов и работников склада.

Предметом данного исследования является процесс управления запасами в небольшом магазине с одним складом. Объектом данного исследования является

розничная торговля, в частности, малые розничные предприятия, которые сталкиваются с проблемами управления запасами. Предметом исследования будет разработка СУЗ, которая является доступной, удобной и масштабируемой, что делает ее идеальным решением для малого бизнеса.

Научная новизна данного диплома заключается в разработке СУЗ, специально предназначенной для малого розничного бизнеса. Система будет разработана с использованием языка программирования Python и оптимизирована с помощью математических моделей и алгоритмов. В дипломе также будет рассмотрена постановка и решение проблемы оптимизации, чтобы система обеспечивала точное обновление запасов в режиме реального времени.

Практическая значимость данного диплома заключается в его способности повысить операционную эффективность и прибыльность магазина. СУЗ обеспечит точное обновление запасов в режиме реального времени, сократит количество ошибок и неэффективности, а также повысит удовлетворенность клиентов. Масштабируемость системы означает, что она может быть расширена для удовлетворения будущих потребностей магазина и других малых предприятий. Система будет разработана с целью снижения затрат и времени, связанных с управлением запасами, тем самым повышая прибыльность магазина.

Методы научного исследования, используемые в данном дипломе, включают анализ данных, сбор требований, разработку программного обеспечения и тестирование. Структура диплома будет состоять из нескольких частей, включая анализ домена, постановку задачи, общее описание системы, математическое обеспечение и практическую реализацию. В дипломе также будут представлены руководства пользователя и техническая поддержка для сотрудников магазина.

В заключение следует отметить, что целью данного диплома является решение проблем, с которыми сталкиваются малые розничные предприятия при эффективном управлении товарными запасами. СУЗ, разработанная в данном исследовании, обеспечит доступное, удобное и масштабируемое решение для повышения операционной эффективности и прибыльности магазина. Структура

диплома будет охватывать все аспекты процесса разработки, представляя собой всеобъемлющее руководство для предприятий, стремящихся улучшить свои системы управления запасами. Разработанная в данном исследовании СУЗ будет иметь практическое значение в розничной торговле, способствуя развитию эффективных и экономически выгодных систем управления запасами для малого бизнеса.

1 Анализ предметной области

Управление складом играет важную роль в логистической цепочке. Каждое предприятие, занимающееся производством или продажей товаров, сталкивается с необходимостью хранения, учета и управления складом. Управление складом включает в себя различные операции, такие как прием товаров на склад, их хранение, отбор, упаковка и отгрузка. Кроме того, не менее важно контролировать уровень запасов на складе, чтобы вовремя пополнять их, не допуская их недостатка или избытка.

Системы управления складом позволяют автоматизировать и ускорить процессы на складе, обеспечивая точный учет товаров на складе, отслеживание их движения и контроль уровня запасов. Существует множество решений, некоторые из которых предназначены для больших складов с большим объемом товаров, а другие – для небольших складов с небольшим товарооборотом.

Управление запасами является важной частью управления складом. Управление запасами включает в себя определение оптимального уровня запасов, управление заказами и пополнением запасов, а также контроль запасов. Оптимизация управления запасами позволяет сократить издержки, связанные с хранением и управлением запасами, а также сократить риски связанные с избыточными запасами.

Системы управления складом (СУС) – это программное обеспечение, которое позволяет автоматизировать процессы на складе и управлять всеми операциями, связанными с хранением и учетом товаров на складе. СУС позволяет значительно повысить эффективность управления складом, уменьшить ошибки и снизить затраты на управление складом.

СУС включает в себя следующие функциональные модули:

- модуль приема товаров на склад – этот модуль позволяет автоматизировать процесс приема товаров на склад, включая проверку товаров на соответствие заказу и их регистрацию в системе управления складом;
- модуль хранения товаров – этот модуль позволяет управлять процессом размещения товаров на складе, включая определение места хранения, контроль запасов и оповещение о необходимости пополнения запасов;
- модуль отбора товаров – этот модуль позволяет автоматизировать процесс отбора товаров с полок для их отгрузки, включая определение оптимального маршрута перемещения товаров и контроль остатков на складе;
- модуль упаковки и отгрузки товаров – этот модуль позволяет управлять процессом упаковки товаров и их отгрузки, включая формирование товаросопроводительной документации и контроль корректности отгрузки товаров;
- модуль управления запасами – этот модуль позволяет автоматизировать процесс управления запасами на складе, включая определение оптимального уровня запасов, управление заказами и контроль запасов.

СУС подразделяются на разные типы, включая:

- локальные системы – это системы, которые установлены непосредственно на компьютерах на складе и работают только в пределах локальной сети;
- облачные системы – это системы, которые доступны через интернет и хранятся на удаленных серверах. Облачные системы позволяют работать с системой управления складом из любой точки мира и не требуют больших затрат на оборудование и его обслуживание;
- гибридные системы – это системы, которые объединяют в себе возможности локальных и облачных систем. Гибридные системы позволяют

использовать преимущества обоих типов систем и обеспечивают высокую гибкость в работе с системой управления складом.

При выборе СУС необходимо учитывать особенности конкретной компании, ее размеры и объемы товарооборота, а также функциональные требования к системе управления складом.

Преимущества использования СУС включают:

- увеличение эффективности управления складом и сокращение затрат на управление им;
- уменьшение ошибок и повышение точности учета товаров на складе;
- ускорение процессов на складе и повышение производительности труда;
- улучшение качества обслуживания клиентов;
- увеличение прозрачности и контроля над процессами на складе.
- возможность проводить анализы и получать отчеты о работе склада, что позволяет оптимизировать процессы управления складом.

Тем не менее, использование СУС также имеет свои недостатки, такие как:

- высокие затраты на внедрение системы и обучение персонала;
- необходимость постоянного обновления системы и ее адаптации к изменениям в бизнес-процессах компании;
- ограниченность функциональности системы и ее невозможность учитывать все особенности работы компании;
- необходимость постоянного обслуживания и технической поддержки системы.

В целом, использование системы управления складом является одним из важных элементов оптимизации работы склада и всей логистической цепочки компании. Правильный выбор системы и ее адаптация к особенностям компании позволяют снизить затраты на управление складом и повысить его эффективность.

Управление запасами – это процесс управления запасами товаров на складе с целью обеспечения непрерывности производственных и торговых операций

компании. Основная задача управления запасами заключается в определении оптимального уровня запасов, который позволит компании удовлетворять потребности своих клиентов, минимизировать затраты на хранение запасов и избежать потерь, связанных с устареванием товаров.

Основным инструментом управления запасами является система управления запасами (СУЗ), которая позволяет компаниям эффективно управлять запасами, контролировать их движение и максимально оптимизировать процесс управления запасами.

Система управления запасами включает в себя следующие основные компоненты:

- управление заказами – этот компонент включает в себя определение оптимального уровня запасов и управление заказами на закупку товаров. В зависимости от модели управления запасами компания может использовать различные методы определения оптимального уровня запасов, как, например, классификация товаров на складе по степени их значимости для компании;
- управление поставками – этот компонент включает в себя управление процессом поставки товаров на склад, контроль качества поставляемых товаров и управление отношениями с поставщиками;
- управление складом – этот компонент включает в себя управление процессом хранения товаров на складе, контроль запасов и управление процессом отгрузки товаров;
- управление продажами – этот компонент включает в себя управление процессом продажи товаров, контроль запасов и управление процессом отгрузки товаров клиентам;
- управление производством – этот компонент включает в себя управление процессом производства товаров, контроль запасов и управление процессом отгрузки готовой продукции клиентам;

- анализ и отчетность – этот компонент включает в себя анализ производительности системы управления запасами и получение отчетов о работе системы.

Система управления запасами позволяет компаниям оптимизировать процесс управления запасами и достичь следующих преимуществ:

- снижение затрат на хранение запасов и уменьшение потерь, связанных с устареванием товаров;
- улучшение уровня обслуживания клиентов и сокращение времени поставки товаров;
- увеличение эффективности производственных и торговых операций компании;
- увеличение гибкости и масштабируемости производства и торговых операций;
- улучшение контроля над процессом управления запасами и повышение прозрачности в работе всей логистической цепочки компании;
- возможность проводить анализы и получать отчеты о работе системы управления запасами, что позволяет оптимизировать работу всей логистической цепочки компании.

Однако использование системы управления запасами также имеет свои недостатки, такие как:

- высокие затраты на внедрение системы и обучение персонала;
- необходимость постоянного обновления системы и ее адаптации к изменениям в бизнес-процессах компании;
- ограниченность функциональности системы и ее невозможность учитывать все особенности работы компании;
- необходимость постоянного обслуживания и технической поддержки системы.

Размещение товаров на складе – это один из важнейших этапов логистической деятельности компании, который напрямую влияет на эффективность процесса управления запасами и уровень обслуживания клиентов.

Основные принципы размещения товаров на складе:

- размещение товаров должно быть организовано таким образом, чтобы обеспечить максимальную эффективность использования площади склада и минимизировать время на поиск и отбор товаров;
- товары должны быть размещены на складе с учетом их характеристик и особенностей хранения, а также с учетом частоты и объема продаж;
- на складе должна быть организована система маркировки и идентификации товаров, которая позволит быстро находить и отбирать нужные товары;
- размещение товаров на складе должно соответствовать требованиям пожарной безопасности и санитарно-гигиеническим нормам.

Существует несколько методов размещения товаров на складе:

- метод кросс-докинга – это метод, при котором товары не размещаются на складе, а сразу после прибытия на склад перегружаются на транспортное средство для доставки клиенту. Этот метод позволяет сократить время на перевалку товаров и повысить скорость доставки товаров;
- метод зонирования – это метод, при котором на складе создаются зоны с различными условиями хранения для разных групп товаров. Например, зона с низкой температурой для хранения продуктов питания, зона с малой влажностью для хранения электроники и прочее;
- метод сезонного размещения – это метод, при котором товары размещаются на складе в зависимости от сезона и спроса на них. Например, зимние товары размещаются на складе летом, а летние – зимой;
- метод размещения товаров по частоте продаж – это метод, при котором товары размещаются на складе в порядке убывания частоты продаж. Товары,

которые чаще всего продается, размещаются ближе к зоне отбора, чтобы сократить время на их отбор.

Правильно организованное размещение товаров на складе позволяет компаниям снизить затраты на управление запасами, сократить время на обработку заказов и повысить уровень обслуживания клиентов.

Отбор товаров – это процесс, при котором сотрудники склада собирают заказы клиентов из имеющихся на складе товаров. Этот процесс является одним из ключевых элементов логистической цепочки компании, так как от его эффективности зависит уровень обслуживания клиентов и время доставки товаров.

Основные принципы отбора товаров:

- отбор товаров должен быть организован таким образом, чтобы минимизировать время на отбор заказов и повысить скорость доставки товаров клиентам;
- сотрудники склада, занимающиеся отбором товаров, должны иметь высокий уровень компетенции и знаний о характеристиках и особенностях хранения товаров;
- на складе должны быть организованы зоны отбора товаров, которые позволят сотрудникам склада быстро находить нужные товары;
- отбираемые товары должны быть тщательно проверены на соответствие заказу и наличие дефектов.

Существует несколько методов отбора товаров:

- отбор товаров по заказу – это метод, при котором сотрудники склада отбирают товары для каждого заказа отдельно. Этот метод позволяет минимизировать ошибки при отборе товаров и повысить точность выполнения заказов;
- отбор товаров по партиям – это метод, при котором товары отбираются пакетами из нескольких заказов. Этот метод позволяет сократить время на отбор товаров и уменьшить количество ошибок при отборе;

- отбор товаров по зонам – это метод, при котором товары отбираются из зон, организованных на складе. Этот метод позволяет сократить время на поиск товаров и повысить скорость отбора товаров.

В процессе отбора товаров в основном используются два технологических решения:

- системы сканирования штрих-кодов – это система, которая позволяет сотрудникам сканировать штрих-коды на товарах для идентификации товара и сверки с заказом;
- автоматические системы отбора товаров – это система, которая позволяет автоматически отбирать товары из зон на складе без участия сотрудников склада.

Правильно организованный процесс отбора товаров позволяет компаниям повысить эффективность работы склада, снизить время на обработку заказов и улучшить уровень обслуживания клиентов.

Управление складом является важным элементом логистической цепочки компаний, который позволяет при правильной организации эффективно управлять запасами товаров, минимизировать затраты на хранение и улучшать уровень обслуживания клиентов. Все вышеописанные принципы должны быть адаптированы к конкретным потребностям и характеристикам каждой компании. Таким образом можно достичь максимальной эффективности управления складом и повысить конкурентоспособность компании на рынке.

2 Постановка задачи

Целью данной работы является повышение эффективности комплектации и маршрутизации продукции на складе. Для достижения поставленной цели необходимо разработать систему управления запасами на складе. В том числе должны быть решены следующие задачи:

- разработка схемы базы данных и создание необходимых таблиц для хранения информации о товарных позициях и их местонахождении на складе, местах хранения, носителях товаров и текущем уровне запасов;
- разработка пользовательского интерфейса, который обеспечит удобный интерфейс для управления инвентарными позициями и их местоположением, местами хранения, носителями товаров и уровнем запасов;
- реализация функциональности *CRUD* (*Create, Read, Update, Delete*) для управления сущностями созданных таблиц;
- разработка функциональности для регистрации инвентарных операций, таких как получение новых инвентарных позиций, размещение их на складе и отбор для продажи;
- внедрение функциональности отслеживания запасов, которая позволит системе отслеживать уровни запасов в режиме реального времени и предоставлять отчеты об уровнях запасов и их движении.

В процессе анализа существующих решений был определен функционал, который должен быть реализован в разрабатываемой системе.

На следующем этапе осуществляется выбор целевой платформы и определение необходимых вычислительных ресурсов для работы системы. В качестве платформы для данной системы будем использовать веб-приложение, т.к. браузерные приложения гибкие, универсальные, не требуют предварительной подготовки среды, позволяют сэкономить финансы компании, аппаратные ресурсы, время сотрудников.

После выбора платформы необходимо выбрать стек технологий (набор программных решений и библиотек, которые будут обеспечивать функционирование сервиса), который будет использоваться для разработки каждой части системы. Для этого необходимо провести анализ и выявить, какой язык программирования лучше подходит для решения поставленной задачи и какие возможности он предоставляет.

Информационная и даталогическая модели, функциональная схема разрабатываемого решения строятся, исходя из заложенного в систему функционала. Информационная модель отражает общую структуру сущностей, присутствующих в системе, не вдаваясь в подробности реализации каждой из них. Даталогическая модель раскрывает внутреннюю структуру сущностей, а также устанавливает связи между ними. Функциональная схема описывает все типовые сценарии взаимодействия пользователей с системой.

Основываясь на функциональной схеме и даталогической модели, наступает этап практической реализации. Разработка веб-приложения требует развертывания серверной части для тестирования разрабатываемого ПО, поэтому предварительно необходимо настроить необходимое ПО. После этого в соответствии с даталогической моделью создается база данных. Далее разрабатываются все модули системы. После окончания процесса разработки производится тестирование работоспособности системы. При возникновении ошибок необходимо произвести анализ и устранить их. На последнем этапе создается справочная документация по работе приложения.

3 Общее описание системы

Архитектура системы является клиент-серверной, где клиентом является интерфейс пользователя для управления запасами, поставками и комплектацией, а сервером – база данных и модули управления запасами, поставками и комплектацией.

Архитектура системы управления складом представляет собой трехуровневую структуру. Это позволит разделить функциональность системы на три уровня: уровень представления, уровень бизнес-логики и уровень доступа к данным.

3.1 Описание структурной схемы

В структурной схеме представлены подсистемы, их информационные обеспечения, лингвистические обеспечения, программные и математическое обеспечение, а также математическое обеспечение для всей программы и техническое обеспечение для ее корректной работы. Структурная схема представлена в приложении А.

В систему входят следующие подсистемы:

- подсистема представлеия – на уровне представления будет реализован интерфейс пользователя, который позволит управлять складскими процессами и контролировать запасы товаров. На этом уровне будет реализована функциональность управления запасами, учета товаров, контроля за сроками годности и др. Для реализации интерфейса пользователя будем использовать технологии веб-разработки, а именно *HTML*, *CSS* с использованием шаблонов *Django* и фреймворка *Bootstrap*;
- подсистема бизнес-логики – на уровне бизнес-логики будет реализована основная функциональность системы управления складом. На этом уровне будут реализованы модули управления запасами, учета товаров, контроля за сроками

годности и др. Для реализации бизнес-логики системы будем использовать язык программирования *Python*;

- подсистема доступа к данным – На уровне доступа к данным будет реализована функциональность доступа к базе данных. Для хранения данных будем использовать реляционную базу данных *PostgreSQL*. Для доступа к базе данных будем использовать язык программирования *Python*.

Для правильной работы техническое обеспечение для данной системы необходимо составить для каждого из уровней.

Уровень представления:

- Персональный компьютер со следующими характеристиками:
  - Процессор *Intel Core i3* или выше;
  - ОЗУ от 4 ГБ;
  - Видеокарта с поддержкой *OpenGL* 2.0 или выше;
  - Клавиатура и мышь.
- Терминал сбора данных (ТСД) *CipherLab RK95-2S-38K*.

Уровень бизнес-логики:

- Персональный компьютер со следующими характерисиками:
  - Процессор *Intel Core i7* или выше;
  - ОЗУ от 16 ГБ;
  - Жесткий диск *SSD*;
  - Графический адаптер с поддержкой *OpenGL* 3.3 или выше.

Уровень доступа к данным:

- Персональный компьютер со следующими характерисиками:
  - Процессор *Intel Core i5* или выше;
  - ОЗУ от 8 ГБ;
  - Жесткий диск *SSD*.

Уровень разработки системы:

- Среда разработки *IntelliJ IDEA*.

Лингвистическое обеспечение системы:

- язык программирования *Python*;
- фреймворк *Django*;
- язык гипертекстовой разметки *HTML*;
- формальный язык описания веб-страницы *CSS*;
- фреймворк *Bootstrap*;
- *SQL* запросы;
- пользовательский интерфейс – элементы навигации, диалоги, меню, формы.

Информационное обеспечение системы:

- таблица «*Vendors*»;
- таблица «*Products*»;
- таблица «*Users*»;
- таблица «*StorageUnits*»;
- таблица «*Pallets*»;
- таблица «*InventoryItems*»;
- таблица «*InventoryTransactions*»;
- таблица «*StoragePalletLinks*»;
- таблица «*PalletProductLinks*».

3.2 Описание функциональной схемы

Функциональная схема представлена в приложении Б. Функциональную схему разрабатываемой СУЗ можно разделить на три основных компонента: управление запасами, отчетность и пользовательский интерфейс. Рассмотрим каждый компонент подробнее.

1. Управление запасами.

Компонент управления запасами является ядром системы. Он позволяет пользователям управлять запасами, включая получение, складирование и комплектацию продукции. Компонент управления запасами состоит из следующих подкомпонентов:

- получение продуктов: этот подкомпонент позволяет пользователям получать продукты на склад. Пользователи могут вводить в систему данные о полученных продуктах, такие как название продукта, штрих-код, количество и местоположение. Система обновляет уровни запасов и добавляет записи транзакций для отслеживания движения запасов;

- запасы продуктов: этот подкомпонент позволяет пользователям размещать товары на паллетах. Пользователи могут выбрать продукт, который они хотят складировать, ввести местоположение паллета и количество продукта, который они складируют. Система обновляет уровни запасов и добавляет записи операций для отслеживания движения запасов;

- подбор продуктов: этот подкомпонент позволяет пользователям выбирать продукты с полок. Пользователи могут выбрать продукт, который они хотят выбрать, ввести расположение паллета и количество продукта, который они выбирают. Система обновляет уровни запасов и добавляет записи транзакций для отслеживания движения запасов.

2. Отчетность.

Компонент отчетности СУЗ предоставляет отчеты в режиме реального времени об уровне запасов, движении запасов и производительности продукции. Компонент отчетности предоставляет отчеты в режиме реального времени об уровне запасов, движении запасов и производительности продукции. Пользователи могут создавать отчеты на основе различных критериев, таких как название продукта, штрих-код, местоположение и дата.

3. Пользовательский интерфейс.

Компонент СУЗ с веб-интерфейсом пользователя позволяет сотрудникам управлять запасами, включая получение, складирование и отбор продукции. Он состоит из следующих страниц:

- список инвентарных позиций: отображает список всех инвентарных позиций в системе. Пользователи могут просматривать подробную информацию о каждой позиции, включая ее название, штрих-код, местоположение и количество;
- создание инвентарной позиции: позволяет пользователям создавать новые инвентарные позиции. Пользователи могут ввести сведения о новом элементе, включая его название, штрих-код, местоположение и количество;
- детали инвентарного объекта: отображает подробную информацию об инвентарном объекте, включая его название, штрих-код, местоположение, количество, минимальное количество, максимальное количество и историю транзакций;
- обновление инвентарной позиции: позволяет пользователям обновлять информацию о существующей инвентарной позиции, включая ее название, штрих-код, местоположение, количество, минимальное количество и максимальное количество;
- удалить инвентарную позицию: позволяет пользователям удалять существующие инвентарные позиции из системы;
- создание инвентарной операции: позволяет пользователям добавлять операции для инвентарной позиции, включая тип операции (ввоз/вывоз) и количество.

4 Описание видов обеспечения

4.1 Математическое обеспечение

Для оптимизации маршрутизации продукции на складах необходимо решить задачу кратчайшего пути. Проблема кратчайшего пути представляет собой задачу поиска оптимального пути между двумя узлами в графе с минимальной суммой весов ребер. В контексте управления запасами, для оптимизации процесса маршрутизации, необходимо найти оптимальный маршрут между множеством узлов на складе, а не только между двумя. Эта задача может быть решена с помощью алгоритма Дейкстры.

Алгоритм Дейкстры – это алгоритм поиска кратчайшего пути от одной вершины графа до всех остальных вершин. Он работает с положительными весами ребер и основан на поочередном добавлении вершин в множество S, которые имеют минимальное расстояние от исходной вершины. Алгоритм может быть использован задачи оптимизации, так как граф маршрутизации на складе является взвешенным и неориентированным. Сложность этого алгоритма составляет

$$\Theta\big(E + V \cdot log(V)\big) \quad (1)$$

где: E – количество ребер, V – количество вершин.

4.1.1 Математическая модель

Математическая модель для решения задачи маршрутизации продукции на складе описывается графом, в котором вершины соответствуют областям на складе, а ребра графа определяют длины путей между вершинами. Для алгоритма Дейкстры математическая модель представлена следующим образом:

- V – множество вершин графа, где каждая вершина представляет собой определенную область на складе.
- E – множество ребер графа, где каждое ребро представляет длину пути между вершинами.

- $w(u,v)$ – функция веса ребра графа, которая определяет расстояние между вершинами u и v.

- s – начальная вершина графа, которая представляет собой начальное положение продукции на складе.

- t – конечная вершина графа, которая представляет собой конечное положение продукции на складе.

4.1.2 Постановка математической задачи

Формулировка задачи маршрутизации продукции на складе может выглядеть следующим образом:

Дано n вершин, где каждая вершина представляет собой определенную область на складе. Каждая вершина имеет свои координаты x_i и y_i. Для каждой пары вершин i и j задана длина пути $d\text{ist}(i, j)$, которая определяет расстояние между вершинами. Также задано начальное положение продукции на складе и конечное положение, куда продукцию необходимо переместить. Необходимо найти кратчайший путь между начальным и конечным положением продукции на складе, используя заданные длины путей между вершинами.

4.1.3 Алгоритм решения задачи

1. Инициализация:

устанавливаем расстояние от начальной вершины s до всех остальных вершин графа равным бесконечности:

$$\forall\, u \in V, u \neq s\colon dist(s,u) = \infty \qquad (2)$$

- устанавливаем расстояние от начальной вершины s до самой себя равным нулю:

$$dist(s,s) = 0 \quad (3)$$

2. Основной цикл:

- для каждой вершины v, кроме начальной вершины s, находим кратчайший путь от начальной вершины s до v:

$$dist(s,v) = \min\{dist(s,u) + w(u,v)\} \quad (4)$$

где: u – предыдущая вершина на пути от s до v;

$w(u,v)$ – вес ребра между вершинами u и v.

- после нахождения кратчайшего пути до вершины v, помечаем ее как обработанную.

3. Алгоритм завершается, когда все вершины графа будут обработаны или когда будет найден кратчайший путь до конечной вершины.

Таким образом, алгоритм Дейкстры для решения задачи оптимизации маршрутизации продукции на складе заключается в нахождении кратчайших путей от начальной вершины до всех остальных вершин графа

4.2 Информационное обеспечение

В процессе проектирования базы данных была разработана ее даталогическая модель. Даталогическая модель разрабатываемой системы представлена в приложении Б.

Данные, необходимые для функционирования системы, хранятся в базе данных «*inventory_db*», которая содержит следующие таблицы:

- таблица «*StorageUnits*» содержит в себе информацию о местах хранения на складе, их штрих-кодах. Места хранения привязываны к карте склада их идентификационными номерами. Разрабатываемая система имеет два виртуальных места хранения, не имеющих штрих-коды, для упрощения работы с таблицами базы данных: места хранения с идентификационными номерами 0 (если на это место хранения ссылается паллет, это означает, что паллет находится вне склада) и 1 (паллет перемещается по складу в данный момент);
- теблица «*Vendors*» описывает поставщиков и, поля таблицы хранят название компании поставщика и краткое описание;
- таблица «*Products*» содержит в себе такую информацию о реализуемой продукции, как штрих-код производителя, название продукта, стоимость, категория продукта, поставщик;
- таблица «*Pallets*» содержит в себе информацию о паллетах, их штрих-кодах, местоположения на складе согласно местам хранения, заполненность паллета. Каждый паллет связан с местом хранения через его идентификационный номер;
- таблица «*InventoryItems*» содержит в себе товарные позиции на складе. Товарная позиция описывается штрих-кодом, который привязан к таблице «*Products*», количеством единиц продукта, датой добавления и датой истечения срока действия;

- таблица «*InventoryTransactions*» описывает складские операции с товарными позициями, такими как разгрузка, приемка, перемещение товаров, и дату проведения операции. Связка товарных позиций происходит через идентификационный номер из таблицы «InventoryItems»;

- таблица «*StorageItemLinks*» является связкой между местом хранения и товаром, хранящемся на месте хранения;

- таблица «*Employees*» содержит в себе сведения о сотрудниках, работающих на складе. В таблице хранятся имена сотрудников, должности, уровни доступа к системе и учетные данные для входа в систему.

4.3 Лингвистическое обеспечение

Основным языком разработки веб-системы был выбран язык программирования *Python*. *Python* это легкий в изучении, мощный язык программирования. Он имеет эффективные высокоуровневые структуры данных и простой, но эффективный подход к объектно-ориентированному программированию.

Django Framework – это один из самых популярных фреймворков для создания веб-приложений на *Python*. *Django* является высокоуровневым веб-фреймворком, разработанным на языке программирования *Python*. Данный фреймворк предназначен для ускорения процесса веб-разработки и обеспечивает чистый и прагматичный дизайн. Он позволяет значительно снизить трудозатраты, связанные с веб-разработкой. Таким образом, *Django* позволяет разработчикам сосредоточиться на написании собственных приложений, не тратя время на изобретение уже существующих решений.

PostgreSQL – это мощная объектно-реляционная система баз данных с открытым исходным кодом, которая использует и расширяет язык *SQL* в сочетании с множеством функций, позволяющих надежно хранить и масштабировать самые сложные рабочие нагрузки данных

При разработке клиентской части системы использовались язык разметки *HTML* и язык описания стилей *CSS* для создания визуальной части.

Взаимодействие пользователя с серверной частью системы реализуется с использованием графического интерфейса, который состоит из нескольких страниц. Веб-страницы могут включать в себя элементы, необходимые для просмотра данных (таблицы), а также элементы, необходимые для их редактирования, такие как формы ввода или функциональные кнопки.

4.4 Программное обеспечение

Для разработки приложения, необходимо использовать определенный набор программного обеспечения. В первую очередь, потребуется интегрированная среда разработки (*IDE*) *PyCharm*, которая позволяет удобно создавать и отлаживать код, а также обеспечивает доступ к различным инструментам и библиотекам, необходимым для работы с *Django*.

Помимо *IDE*, для разработки веб-приложения необходимо использовать фреймворк *Django*, который предоставляет широкий набор инструментов для создания веб-приложений. Кроме того, для создания пользовательского интерфейса приложения необходимо использовать языки разметки *HTML* и *CSS*, а также фреймворк *Bootstrap*, который предоставляет набор готовых элементов интерфейса и стилей, упрощающих создание удобного пользовательского интерфейса.

Кроме перечисленных выше инструментов, для разработки приложения в первую очередь необходимо использовать интерпретатор *Python*, который позволяет выполнять код, написанный на этом языке программирования. В *PyCharm IDE*, как правило, уже встроен интерпретатор *Python*, который можно настроить под цели конкретного проекта.

4.5 Техническое обеспечение

Для функционирования серверной части системы необходим виртуальный или выделенный сервер. Целесообразно использовать виртуальный сервер, так как серверная часть приложения является легковесной, а также использование виртуального сервера дает разработчику полный контроль над развернутым ПО.

Для разработки, отладки и использования необходимо наличие персонального компьютера, клавиатуры, мыши, монитора или ноутбука. Минимальные требования: четырехъядерный 64-разрядный процессор с тактовой частотой 1,6 ГГц или выше, 6 Гб ОЗУ, 40Гб свободного пространства на накопителе.

Для работы с клиентской частью приложения необходим доступ к сети Интернет. Также рекомендовано, но не обязательно наличие терминала сбора данных. Минимальные требования: процессор с тактовой частотой 1,4 ГГц или выше, 2 Гб ОЗУ, 16 Гб свободного пространства на накопителе, физическая клавиатура 25 или 28 кнопок, *Wi-Fi IEEE* 802.11 модуль.

4.6 Методическое обеспечение

При первом запуске главной страницы пользователю необходимо пройти аутентификацию, введя в форму логин и пароль. После этого откроется главная страница, в котором расположено меню. Пользователю необходимо выбрать необходимую функциональность, нажав соответствующую кнопку:

- «список позиций»;
- «создание позиции»;
- «редактирование позиции»;
- «удаление позиции»;
- «детали паллета»;
- «создание операции»;

- «детали операций»;
- «карта склада».

5 Защита информации

В современном мире информация играет важную роль, поэтому защита от утечек конфиденциальной информации является важным аспектом в различных секторах деятельности. Защита информации представляет собой комплекс мер, направленных на предотвращение утечки защищаемых данных и непреднамеренных или несанкционированных воздействий на защищаемые данные. В контексте разработки системы, применены следующие методы защиты информации:

- аутентификация пользователей, с использованием пароля;
- использование протокола *HTTPS*;
- хэширование паролей пользователей;
- встроенные механизмы защиты фреймворка *Django*.

Чтобы войти в систему, пользователю необходимо ввести логин и пароль. Если пользователь не зарегистрирован в системе, то администратор имеет возможность зарегистрировать пользователя в системе и предоставить ему право доступа к ее ресурсам, право на изменение информации, хранящейся в базе данных, право на запись новой информации. Данный функционал обеспечивает конфиденциальность информации.

Протокол *HTTPS* является защищенной версией протокола *HTTP*, который используется для передачи данных в Интернете. Он обеспечивает безопасную передачу данных между клиентом и сервером, используя шифрование данных и аутентификацию. *HTTPS* использует криптографические протоколы для шифрования данных, передаваемых между клиентом и сервером, что позволяет обеспечить конфиденциальность информации, передаваемой между клиентом и сервером. *HTTPS* обеспечивает аутентификацию сервера и клиента, что позволяет убедиться в том, что данные не передаются злоумышленникам, а также обеспечивает защиту от некоторых видов атак, в частности, от атаки типа "человек посередине".

Хэширование паролей пользователей (процесс преобразования пароля в строку случайных символов, которая служит в качестве проверочного значения вместо самого пароля) позволяет сохранить пароли пользователей в безопасном виде и предотвратить их утечку в случае взлома базы данных или другого способа несанкционированного доступа к информации.

При использовании хэширования паролей, пользователь при регистрации вводит свой пароль, который затем хэшируется и сохраняется на сервере. При входе в систему, введенный пользователем пароль также хэшируется, и полученное значение сравнивается с сохраненным значением хэша. Если значения совпадают, то пользователь получает доступ к системе. Такая процедура позволяет предотвратить утечку паролей в случае несанкционированного доступа к базе данных. Даже если злоумышленники получат доступ к хэшам паролей, они не смогут прочитать их и использовать для входа в систему. На сервере *Linux* широко используется алгоритм хэширования паролей *SHA*-512, который обеспечивает высокую степень защиты при правильной настройке и использовании.

Фреймворк Django имеет встроенные механизмы для защиты от атак, таких как *SQL*-инъекции и *XSS*-атаки. *SQL*-инъекция (*SQL Injection*) - это атака на веб-приложение, которая позволяет злоумышленнику получить несанкционированный доступ к базе данных приложения. Атака происходит путем внедрения вредоносного *SQL*-кода в форму или *URL*-адрес, которые обрабатываются приложением и передаются в базу данных.

XSS (*Cross-Site Scripting*) - это атака на веб-приложение, которая позволяет злоумышленнику внедрять вредоносный код на страницу и заставлять ее выполняться в браузере пользователя. Атака происходит путем внедрения вредоносного скрипта в *URL*-адрес, форму или любое другое веб-приложение, которое позволяет пользователю вводить данные.

Как правило, обе атаки используются для кражи личных данных пользователей, например, логинов и паролей. Злоумышленник может изменить или

удалить данные из базы данных, выполнить другие вредоносные действия, такие как перенаправление пользователя на другой сайт или внедрение вредоносного кода на страницу. *Django* использует специальные методы защиты, такие как фильтрация ввода данных, проверка на валидность вводимых данных, использование параметризованных запросов к базе данных, правильная настройка конфигурации базы данных и сервера, чтобы минимизировать возможность атак.

6 Практическая реализация

6.1 Подготовка среды выполнения

Чтобы начать программировать на языке программирования *Python*, нужно установить интерпретатор *Python*. Интрепретатор выступает в роли дешифровщика исходного кода в байт-код, который впоследствии выполняется. Существует несколько реализаций *Python*, и среди них для разработки была выбрана реализация *CPython*, так как она считается эталонной, поддерживает большинство активно используемых платформ и распротраняется под свободной лицензией. Была скачана и установлена версия *Python* 3.10.5.

Разрабатываемая система является веб-приложением. Для разработки серверной части приложения был выбран фреймворк *Django*. *Django* - это веб-фреймворк, написанный на языке программирования *Python,* который предоставляет множество преимуществ. Он позволяет быстро создавать веб-приложения благодаря готовым шаблонам, аутентификации и административному интерфейсу. Фреймворк имеет встроенное объектно-реляционное отображение, что упрощает работу с базой данных и избавляет от необходимости писать сложные *SQL*-запросы. *Django* имеет встроенные механизмы для защиты от атак, таких как *SQL*-инъекции и *XSS*-атаки. Фреймворк позволяет создавать крупные и сложные веб-приложения, которые могут обрабатывать большое количество запросов и пользователей.

Для разработки приложения в целом необходимо установить среду разработки. Была выбрана одна из самых популярных интегрированных сред разработки на настоящий момент для языка *Python – PyCharm* версии *Professional* на основе *IntelliJ IDEA*. Версия *Professional* была выбрана по следующим соображениям:

- поддержка *Django*: *PyCharm* предоставляет полноценную поддержку *Django*, включая автоматическое создание и обновление шаблонов, автодополнение для моделей и представлений.

- интеграция с *Git*: *PyCharm* имеет встроенную поддержку Git, позволяющую легко контролировать версии кода, создавать ветки и выполнять слияния.

- удобная отладка: *PyCharm* предоставляет мощный отладчик *Python*, который позволяет удобно отслеживать выполнение кода и исправлять ошибки.

- Возможность развертывания приложения на удаленном сервере и интеграцию с системами управления базами данных.

6.2 Проектирование веб-приложения

Фреймворк *Django* обеспечивает архитектуру паттерна *Model-View-Controller* (*MVC*) при помощи слабо связанных готовых компонентов. Паттерн *MVC* разделяет аспекты приложения (логику ввода, бизнес-логику и логику пользовательского интерфейса), обеспечивая при этом свободную связь между ними.

В фреймворке Django архитектурный паттерн *MVC* принято обозначать как *Model-View-Template (MVT)*. Паттерн *MVT* разделяет обязанности по обработке данных (*Model*), отображению пользовательского интерфейса (*View*) и управлению потоком информации (*Template*).

- *Model* (Модель) представляет данные и бизнес-логику приложения. Она определяет структуру и поведение таблиц базы данных через классы Python, известные как модели. Модели обрабатывают операции с базой данных, такие как запрос, вставка, обновление и удаление записей. Объектно-реляционное отображение (*ORM*) *Django* сопоставляет эти модели с таблицами базы данных, упрощая взаимодействие с базой данных;

- *View* (Отображение, Вид) отвечает за обработку пользовательских запросов и генерирование ответов. В *Django* представление – это функция *Python* или представление на основе класса, которое принимает *HTTP*-запросы и возвращает *HTTP*-ответы. Представления взаимодействуют с моделями для

получения данных или манипулирования ими по мере необходимости. Они выполняют любую необходимую обработку данных и передают обработанные данные в шаблон для рендеринга;

- *Template* (Шаблон) определяет презентационный уровень приложения. Шаблоны – это *HTML* файлы, которые включают специальные теги шаблонов Django и фильтры. Теги шаблонов позволяют вставлять динамическое содержимое, операторы потока управления и рендеринг переменных в структуру *HTML*. Шаблоны обеспечивают способ отображения данных, полученных из представлений, в отформатированном виде. Их можно повторно использовать и расширять для поддержания последовательного стиля на нескольких страницах.

Ниже приведена последовательность событий того, как Django отображает веб-страницы:

- пользователь отправляет запрос – когда пользователь обращается к *URL*, запрос отправляется на сервер *Django*. Сервер идентифицирует соответствующий шаблон *URL* в конфигурации *URL* проекта.;

- маршрутизация *URL* – диспетчер *URL Django* сопоставляет запрашиваемый *URL* с зарегистрированными шаблонами *URL*. Соответствующий шаблон *URL* ассоциируется с определенным представлением;

- обработка представления – вызывается определенная функция представления или представление на основе класса. Представление выполняет необходимые операции, такие как получение данных из базы данных с помощью моделей, обработка данных и генерация ответа;

- рендеринг шаблона – представление передает обработанные данные в шаблон. Шаблон отображается путем замены тегов и фильтров шаблона на реальные данные. Полученный *HTML* вместе с данными формирует ответ, который отправляется обратно пользователю;

- отправка ответа пользователю – ответ, содержащий отображенный *HTML*, отправляется обратно в браузер пользователя. Браузер пользователя

интерпретирует *HTML*, преобразуя его в веб-страницу, которую пользователь может видеть и взаимодействовать с ней

Этот цикл повторяется для каждого запроса пользователя, позволяя *Django* динамически генерировать веб-страницы на основе запрошенных *URL*, представлений и шаблонов.

Типичный проект *Django* следует определенной файловой структуре для организации своих компонентов. Описание файловой структуры проекта *Django* следующее:

- корень проекта – каталог верхнего уровня, содержащий весь проект *Django*. Он служит в качестве основного контейнера для всех компонентов проекта. Корневой каталог проекта обычно называется в честь самого проекта;
- файл *manage.py* – утилита командной строки, предоставляемая Django для взаимодействия с проектом. Она используется для выполнения различных задач, таких как запуск серверов разработки, управление миграциями баз данных и выполнение команд управления. Этот файл находится в корневом каталоге проекта;
- файлы конфигурации проекта:
  - *settings.py* – этот файл содержит специфические для проекта настройки и конфигурации, такие как настройки базы данных, конфигурация статических файлов, установленные приложения, промежуточное ПО и т.д;
  - *urls.py* – этот файл определяет шаблоны *URL* для проекта и связывает их с определенными представлениями;
  - *wsgi.py* – этот файл используется для развертывания и служит точкой входа для WSGI-совместимых серверов;
  - *asgi.py* – этот файл используется для развертывания с *ASGI*-серверами.

- статические файлы:
  - каталог «*static»* используется для хранения статических файлов, таких как *CSS, JavaScript*, изображения и т.д. По умолчанию статический каталог расположен в корне проекта, но он также может быть размещен в каталоге каждого приложения для статических файлов, специфичных для конкретного приложения.
- шаблоны:
  - каталог templates используется для хранения *HTML* шаблонов, используемых для рендеринга представлений. Он содержит *HTML* файлы с тегами и фильтрами шаблонов *Django*. По соглашению, каталог шаблонов располагается в корне проекта, но он также может быть размещен в каталоге каждого приложения для специфических шаблонов.
- приложения:
  - проекты *Django* состоят из нескольких приложений, каждое из которых выполняет определенную функциональность. Приложения — это модульные компоненты, которые могут разрабатываться независимо и подключаться к нескольким проектам. Каждое приложение имеет свой собственный каталог, который содержит файлы, специфичные для этого приложения, такие как модели, представления, шаблоны и статические файлы.
- медиафайлы:
  - каталог медиафайлов используется для хранения загружаемых пользователем файлов, таких как изображения или документы. Он отделен от каталога статических файлов, чтобы упростить управление и обслуживание пользовательского контента.
- база данных и миграции:
  - *Django* использует бэкенды баз данных для управления сохранением данных. Файлы, связанные с базой данных, включают

конфигурацию базы данных в *settings.py* и сгенерированные миграции базы данных в каталоге каждого приложения.

Это высокоуровневый обзор файловой структуры проекта *Django*. Важно отметить, что хотя эта структура является общепринятой, она может быть изменена в соответствии с конкретными требованиями проекта.

Разработанный проект состоит из нескольких программных модулей (приложений Django). Структура проекта следующая:

- приложение *poll* – модуль мониторинга и учета операций;
- приложение *inventory* – модуль мониторинга и учета товара на складе;
- приложение *logistics* – модуль мониторинга (мест хранения и паллетов на складе) и принятия решений по размещению и перемещению товара на складе.

6.3 Разработка веб-приложения

Файл шаблона *layout.html* содержит *HTML* разметку общего вида приложения. В дополнение он содержит интерфейсный инструментарий Bootstrap 5, который подключается через сеть доставки контента *CDN*. Разметка приложения проиллюстрирована на рисунке 6.1 и представляет собой экран, разделенный на две части: меню навигации по веб-сайту и функциональная (правая) часть экрана. Общий вид приложения и использование функционального пространства на примере некоторых состояний использования представлен на рисунках 6.2, 6.3, 6.4.

Все остальные шаблоны расширяют этот шаблон, не нарушая общий вид приложения, за исключением шаблона входа в систему *login.html*, при отображении которого не выводится меню навигации.

При входе на сайт, при условии, что пользователь еще не прошел авторизацию, отображается форма авторизации. После ввода логина и пароля и отправки формы через модель User встроенного Django пакета авторизации django.contrib.auth проверяется наличие пользователя в базе данных. При успешной

авторизации пользователю выводится главная страница, откуда он через меню навигации может выполнять действия.

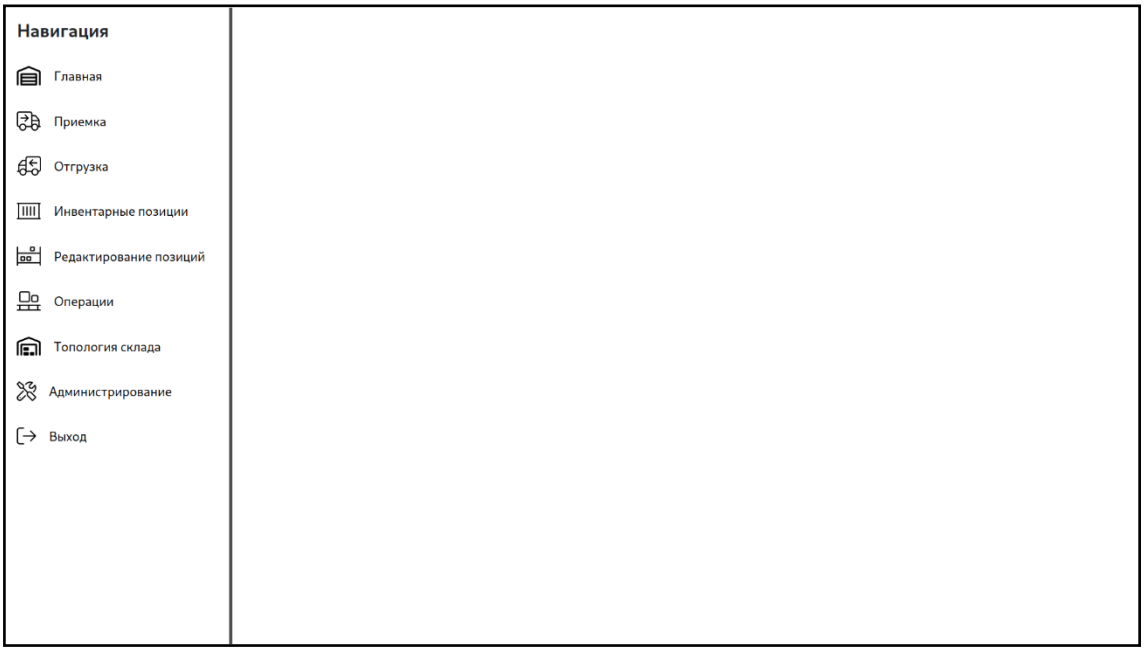


Рисунок 6.1 – Разметка веб-приложения

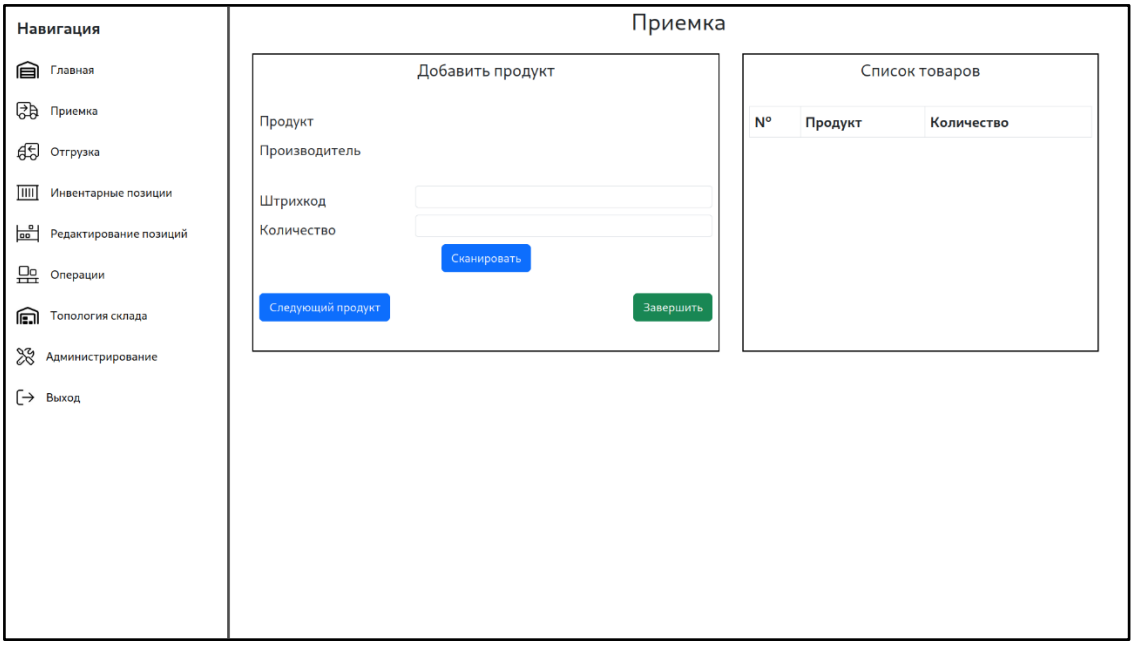


Рисунок 6.2 – Начальная стадия приемки

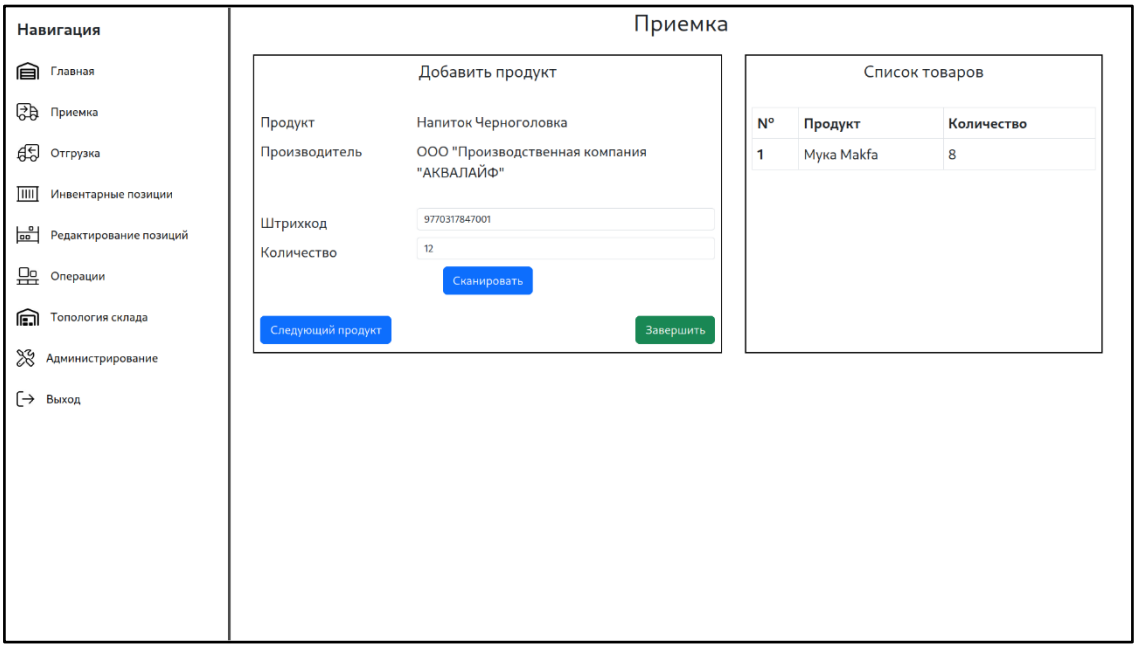


Рисунок 6.3 – Стадия приемки после сканирования штрихкода

№	Продукт	Паллет	Количество	Дата
1	Напиток Coca-Cola	P_1234	50	2021-10-01 14:30:00
2	Шоколад Alpen Gold	P_2345	25	2021-10-02 10:00:00
3	Суп Knorr	P_3456	40	2021-10-03 16:45:00
4	Макароны Barilla	P_4567	30	2021-10-04 12:15:00
5	Сыр Gouda	P_5678	15	2021-10-05 09:30:00
6	Колбаса Докторская	P_6789	20	2021-10-06 11:45:00
7	Молоко Простоквашино	P_7890	45	2021-10-07 17:00:00
8	Хлеб Бородинский	P_8901	35	2021-10-08 13:30:00
9	Йогурт Activia	P_9012	60	2021-10-09 08:00:00
10	Сок Rich	P_0123	55	2021-10-10 15:15:00
11	Масло сливочное Молочная страна	P_2341	20	2021-10-11 10:30:00
12	Кетчуп Heinz	P_3412	15	2021-10-12 11:00:00
13	Колбаса Венская	P_4123	30	2021-10-13 14:15:00
14	Картофель Lay's	P_1234	40	2021-10-14 16:45:00
15	Сок Сандора	P_2345	50	2021-10-15 09:30:00

Рисунок 6.4 – Список позиций на складе

Меню навигации состоит из 8 кнопок: «Главная», «Приемка», «Отгрузка», «Инвентарные позиции», «Редактирование позиций», «Операции», «Топология склада», «Администрирование», «Выход». При нажатии на каждую из них, за

исключением кнопки «Выход», пользователь будет перенаправлен на соответствующую страницу. При нажатии кнопки «Выход» пользователь будет деавторизован и перенаправлен на страницу авторизации. Файл *urls.py* связывает *URL*-адреса с *Views* (видами) – функциями, описанными в файле *views.py*. Это означает, что при переходе по определенному *URL* адресу, будет отображен соответствующий вид, который отображается через шаблоны.

Весь пользовательский интерфейс основан на использовании *HTML* шаблонов в связке с языком шаблонов *Django*; он добавляет возможность динамически изменять данные в шаблонах, что позволяет эффективно использовать одну веб-страницу для отображения или изменения различных данных.

Разработка функциональности *CRUD* (*Create, Read, Update, Delete*) в системе управления запасами включала следующие шаги.

- Создание моделей:
  - модели были созданы с помощью классов моделей Django в *models.py*. Каждая модель представляла собой таблицу базы данных и связанные с ней поля;
  - модели определяли структуру и отношения между сущностями, такими как инвентарные позиции, местоположения, места хранения и перевозчики продукции;
  - поля были определены в моделях для хранения соответствующей информации для каждой сущности, включая такие атрибуты, как название товара, *SKU*, количество, местоположение, метки времени и т.д;
  - отношения между моделями устанавливались с помощью таких полей, как внешние ключи, отношения "многие ко многим" или "один к одному".
- разработка представлений:

- представления были созданы для обработки различных операций *CRUD* для каждой сущности с использованием классов и функций представления *Django*;

- для создания записей, представление было реализовано с использованием *Django CreateView* или пользовательских функций представления для обработки данных, представленных через формы;

- для чтения записей были разработаны представления, использующие *Django ListView* или пользовательские функции представления для получения и отображения записей из базы данных;

- обновление записей осуществлялось путем реализации представлений с помощью функции *Django UpdateView* или пользовательских функций представления для получения и обновления определенных записей на основе пользовательского ввода;

- удаление записей осуществлялось путем создания представлений с использованием *Django DeleteView* или пользовательских функций представления для удаления записей из базы данных.

- дизайн форм:

- формы были разработаны с использованием классов форм Django в *forms.py*, чтобы обеспечить удобный интерфейс для ввода и проверки данных;

- классы форм были созданы для представления полей и атрибутов, необходимых для каждой операции *CRUD*;

- типы полей, правила валидации и сообщения об ошибках были определены в классах форм для обеспечения целостности данных и предоставления значимой обратной связи пользователям;

- классы форм были связаны с соответствующими представлениями для обработки отправленных форм и обработки данных.

- создание шаблонов:

- шаблоны были разработаны с использованием системы шаблонов *Django* для определения структуры и макета веб-страниц;
- каждая операция *CRUD* обычно имеет свой собственный набор шаблонов, таких как шаблоны создания, чтения, обновления и удаления, которые использовались для отображения соответствующего содержимого;
- шаблоны включали *HTML*-разметку вместе с тегами шаблонов Django и фильтрами для динамического отображения данных, полученных из представлений и моделей;
- формы были встроены в шаблоны, чтобы пользователи могли вводить или редактировать данные и выполнять операции *CRUD*.
- конфигурация *URL*:
- *URL* были настроены в *urls.py* для привязки представлений к определенным *URL*, что позволило пользователям получить доступ к функциональности *CRUD*;
- шаблоны *URL* были определены с помощью регулярных выражений или сопоставления путей, чтобы определить подходящее представление для вызова на основе запрошенного *URL*;
- каждая операция *CRUD* обычно имеет свой собственный шаблон URL, связанный с соответствующим представлением.

Была разработана схема базы данных для хранения информации о товарно-материальных ценностях, их расположении на складе, местах хранения, носителях продукции и текущих уровнях запасов. Для представления этих объектов были созданы таблицы с соответствующими отношениями, установленными с помощью внешних ключей и других ограничений. При разработке схемы были учтены такие необходимые поля, как название товара, *SKU*, количество, местоположение, временные метки для обновлений и любые дополнительные атрибуты.

Определение модели. Классы моделей *Django*, определенные в *models.py*, были использованы для представления таблиц базы данных и создания объектно-

реляционного отображения (*ORM*). Модели были созданы для каждой сущности, включая инвентарные позиции, местоположения, места хранения и носители продукции. Отношения между этими моделями были определены с помощью таких полей, как внешние ключи, отношения "многие ко многим" или "один к одному". Поля модели были определены для хранения соответствующей информации для отслеживания инвентаризации, такой как название товара, *SKU*, количество, местоположение, временные метки и т.д.

Разработка пользовательского интерфейса. Пользовательский интерфейс (*UI*) был разработан для обеспечения удобного интерфейса для управления инвентарными предметами и их местоположением, местами хранения, носителями продукции и уровнями инвентаризации. Система шаблонов *Django* была использована для создания *HTML* шаблонов для отображения страниц отслеживания запасов. Шаблоны были разработаны для представления информации об уровне запасов, их местонахождении и движении в ясной и организованной форме. Были созданы формы, позволяющие пользователям вводить данные для различных операций с запасами, таких как получение новых запасов, размещение товаров на складе или выбор товаров для продажи.

Регистрация инвентарных операций. Был разработан функционал для регистрации операций инвентаризации, таких как получение новых позиций, размещение их на складе или выбор позиций для продажи. Для обработки логики и данных, представленных через формы, были реализованы представления. Эти представления взаимодействовали с классами модели для выполнения необходимых операций, таких как обновление уровней запасов, запись временных меток и изменение информации о местоположении.

Отслеживание запасов в режиме реального времени и отчетность. Отслеживание запасов в режиме реального времени было реализовано для контроля уровня запасов и предоставления актуальной информации о количестве запасов. Были созданы представления и шаблоны для отображения отчетов об

уровне запасов, движении и других соответствующих показателях. Система использовала запросы и механизмы фильтрации для получения необходимых данных из базы данных и представления их в осмысленном виде. Инструменты визуализации или диаграммы использовались для представления тенденций развития запасов, наличия запасов или другой статистической информации, что расширяло возможности отчетности.

Связь между табличными сущностями базы данных и пользователем происходит через *Models* (модели). Модели — это классы, описанные в файле *models.py*. Каждый такой класс содержит в себе свойства (переменные), название которых является названием поля в таблице базы данных, а сам класс является таблицей. Так, класс *Pallet*, который содержит переменные barcode типа *CharField*, *location* типа *ForeignKey* является описанием таблицы *Pallet* с полями barcode (символьная строка) и location (внешний ключ). Модели должны наследоваться от класса *Model* пакета *django.db*.

Добавление или изменение сущностей осуществляется благодаря специальным формам. Формы – это классы, описанные в файле *forms.py* Эти классы хранят в себе информацию о том, к какой модели (таблице базы данных) применять изменения, редактируемые поля модели и атрибуты полей формы для желаемого отображения. Классы форм имеют метод *save(),* который сохраняет изменения, примененные к необходимой модели. Классы форм наследуются от класса *Form* пакета *django.forms*

Другой способ изменения, добавления, изменения или удаления сущностей – использовать динамически изменяемые страницы. Это специальные *Views* из пакета *django.views.generic*. Они удобны в использовании, когда есть большое количество сущностей таблицы, к которым надо иметь доступ. В таком случае *URL* диспетчеру передается некоторый ключ сущности (обычно это первичный ключ), после через язык шаблонов *Django* генерируется контент для необходимой сущности на веб-странице, что позволяет не создавать для каждого

объекта отдельную *HTML*-страницу. Так, в проекте очень эффективно используется возможность создавать объекты класса *InventoryItem*, для которого создан класс-форма *InventoryItemForm* со свойствами product типа *ForeignKey*, location типа *ForeignKey*, sku типа *CharField* (фиксированной длины), amount типа *IntegerField*, update_date типа *DateTimeField, expire_date* типа *DateField*. Форма передается специальному классу *InventoryItemUpdateView*, наследуемый от класса *UpdateView*, после чего шаблон *inv_upd* отображает страницу изменения/добавления инвентарной позиции. Для отображения инвентарных позиций, используется класс *InventoryItemDetailView*, наследуемый от *DetailView*.

Представленные выше способы не взаимоисключают друг друга: динамически изменяемая страница должна использовать формы для добавления или изменения модели. Удаление происходит без формы, через класс *DeleteView*

ЗАКЛЮЧЕНИЕ

В заключение следует отметить, что разработка системы управления запасами для склада была успешно завершена. Система была разработана для повышения эффективности комплектации и маршрутизации продукции на складе. Цель проекта заключалась в разработке системы, которая позволила бы автоматизировать процесс отслеживания запасов, повысить точность управления запасами и сократить время, необходимое для выполнения задач, связанных с инвентаризацией.

В рамках данной курсоваой работы были разработаны математическое, программное, техническое, лингвистическое, методическое и информационное обеспечения после проведения анализа существующих решений. Также были созданы сервер, сайт сформированы базы данных. Реализованы механизмы защиты информации в виде аутентификации и авторизации на сайте.

СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1. *Python* 3.11.3 *official documentation* [Электронный ресурс] / *Python Software Foundation*. – Режим доступа: *https://docs.python.org/3/*

2. *PostgreSQL: The World's Most Advanced Open Source Relational Database* [Электронный ресурс] / *PostgreSQL* – Режим доступа: *https://www.postgresql.org/*

3. *Inventory management software* [Электронный ресурс]: свободная энциклопедия Wikipedia – Режим доступа: *https://en.wikipedia.org/wiki/Inventory_management_software*

4. *Inventory Management System* [Электронный ресурс] / *ScienceDirect* – Режим доступа: *https://www.sciencedirect.com/topics/economics-econometrics-and-finance/inventory-management-system*

5. *Warehouse Management: A Complete Guide* [Электронный ресурс] / *ScienceDirect* – Режим доступа: https://www.sciencedirect.com/science/article/pii/S1877050917313120

ПРИЛОЖЕНИЕ А

(обязательное)

Структурная схема

Система

Подсистема представления
Информационное обеспечение
1. Таблицы вывода инвентарных позиций
2. Вывод схемы складского помещения
Лингвистическое обеспечение
1. Текстовые поля ввода-вывода, кнопки, таблицы, формы
Программное обеспечение
1. Разметка сайта
2. Шаблоны Django
3. Скрипты для отображения интерфейса

Подсистема бизнес-логики
Информационное обеспечение
1. Лог-файлы состояния системы
Программное обеспечение
1. Python-файлы
Математическое обеспечение
1. Модель поиска кратчайшего пути в графе
2. Модель задачи о рюкзаке

Подсистема доступа к данным
Информационное обеспечение
1. Таблицы БД
Программное обеспечение
1. PostgreSQL запросы
2. Скрипты инициализации и управления БД

Общесистемное программное обеспечение
1. язык программирования Python
2. фреймворк Django
3. язык гипертекстовой разметки HTML
4. формальный язык описания веб-страницы CSS
5. фреймворк Bootstrap
6. SQL запросы

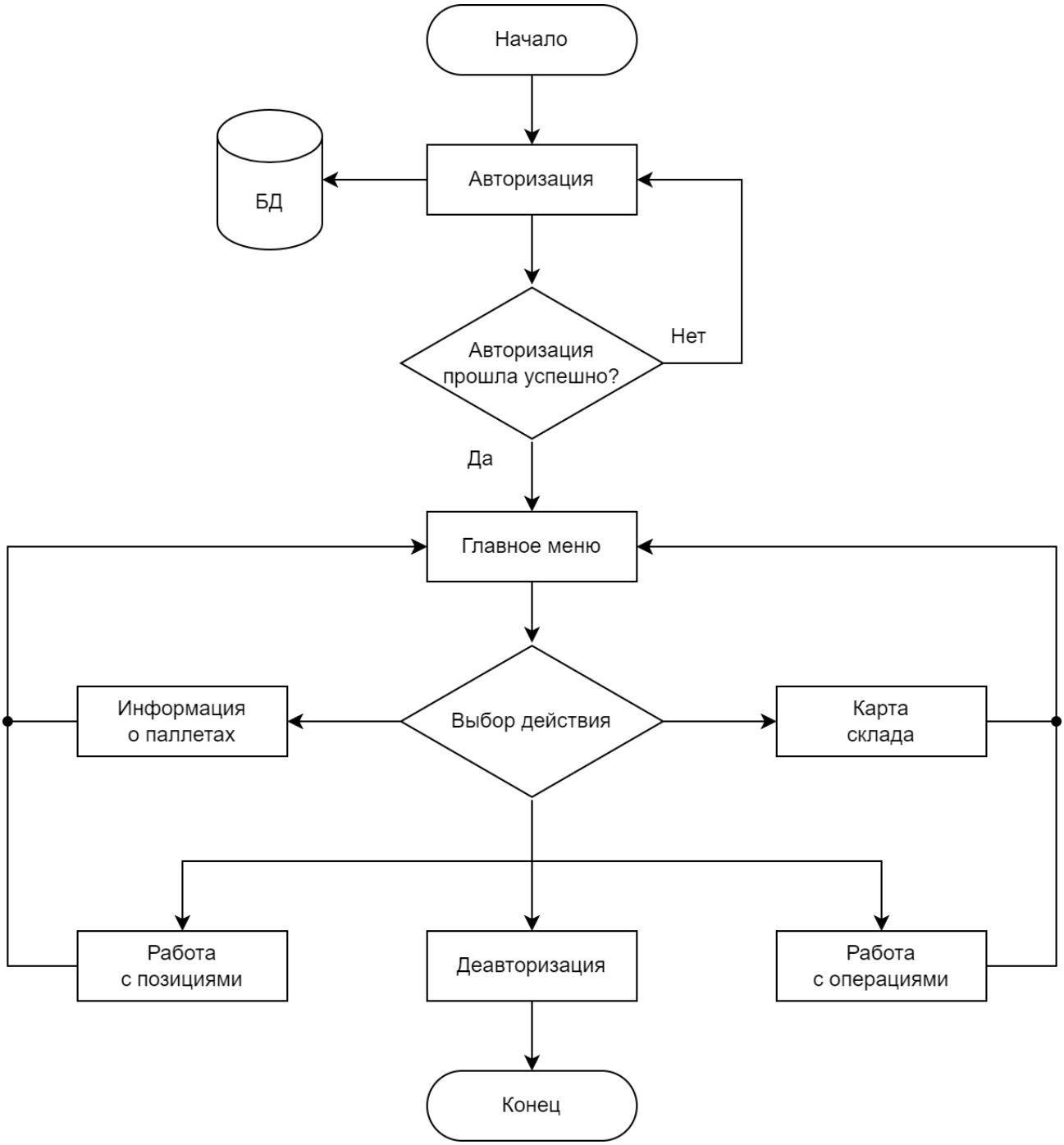
Техническое обеспечение
1.. Пк для персонала:
а. Процессор Intel Core i3 или выше
б. ОЗУ от 4 ГБ
в. Видеокарта с поддержкой OpenGL 2.0 или выше
г. Клавиатура и мышь
2. Терминал сбора данных (ТСД) CipherLab RK95-2S-38K
3. Пк:
а. Процессор Intel Core i7 или выше
б. ОЗУ от 16 ГБ
в. Жесткий диск SSD
г. Графический адаптер с поддержкой OpenGL 3.3 или выше
4. Пк:
а. Процессор Intel Core i5 или выше
б. ОЗУ от 8 ГБ
в. Жесткий диск SSD

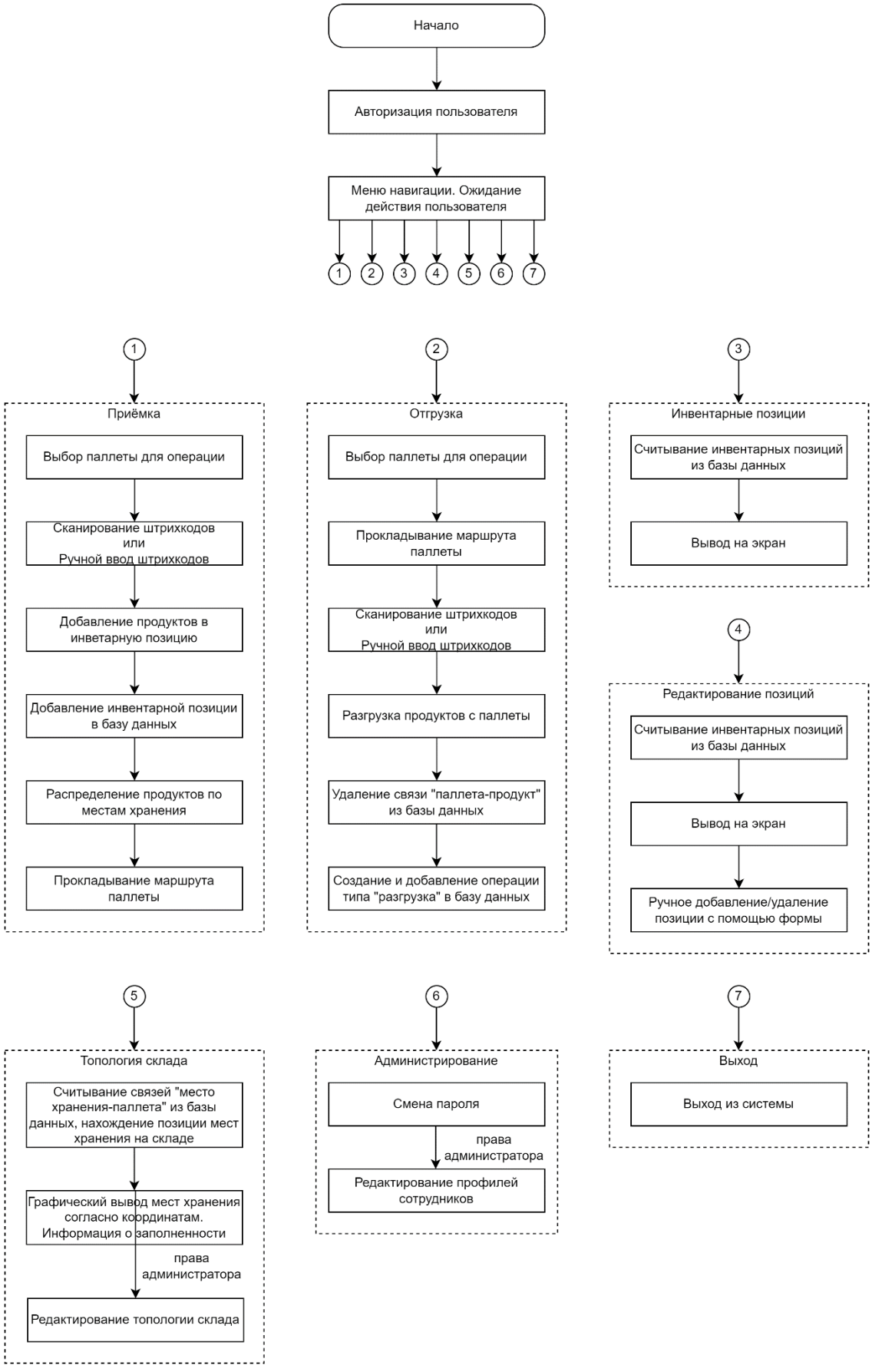
Общесистемное методическое обеспечение
1. Справочники по Python, PostgreSQL
2. Документация по фреймворку Django

ПРИЛОЖЕНИЕ Б

(обязательное)

Функциональная схема




Начало
Авторизация пользователя
Меню навигации. Ожидание действия пользователя
1
2
3
4
5
6
7
Приёмка
Выбор паллеты для операции
Сканирование штрихкодов или Ручной ввод штрихкодов
Добавление продуктов в инветарную позицию
Добавление инвентарной позиции в базу данных
Распределение продуктов по местам хранения
Прокладывание маршрута паллеты
Отгрузка
Выбор паллеты для операции
Прокладывание маршрута паллеты
Сканирование штрихкодов или Ручной ввод штрихкодов
Разгрузка продуктов с паллеты
Удаление связи "паллета-продукт" из базы данных
Создание и добавление операции типа "разгрузка" в базу данных
Инвентарные позиции
Считывание инвентарных позиций из базы данных
Вывод на экран
Редактирование позиций
Считывание инвентарных позиций из базы данных
Вывод на экран
Ручное добавление/удаление позиции с помощью формы
Топология склада
Считывание связей "место хранения-паллета" из базы данных, нахождение позиции мест хранения на складе
Графический вывод мест хранения согласно координатам. Информация о заполненности
права администратора
Редактирование топологии склада
Администрирование
Смена пароля
права администратора
Редактирование профилей сотрудников
Выход
Выход из системы

ПРИЛОЖЕНИЕ В

(обязательное)

Даталогическая схема