Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет
имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К КУРСОВОЙ РАБОТЕ

НА ТЕМУ:

Создание приложения, для управления данными о товарах, хранящихся на складе

Студент ИУ7-62б _____ Блохин Д.М.
(Группа) (Подпись, дата) (И.О.Фамилия)

Руководитель курсовой работы _____ Романова Т. Н.
(Подпись, дата) (И.О.Фамилия)

2022 г.

Аннотация

Курсовой проект на тему: «Создание приложения, для управления данными о товарах, хранящихся на складе».

Автор: Блохин Дмитрий Михаулович.

Руководитель: Романова Татьяна Николаевна.

В работе произведен обзор различных архитектур клиент-серверных приложений, моделей данных(дореляционной, реляционной, постреляционной), выбраны подходящие архитектура и модель БД. Так же Была спроектирована база данных, изображена ER-диаграмма, use-case диаграмма. Было разработано приложение с использованием Spring boot, PostgreSQL, Vue.js. Также были подобраны необходимые для базовых таблиц индексы.

Работа выполнена на 25 листах с использованием 8 источников. Содержит 6 таблиц, 11 схем.

Содержание

Введение

В современном мире все сферы деятельности связаны с использованием информационных технологий, в частности, сети Интернет. Сейчас уже невозможно представить какую-либо компанию или организацию без своего сайта.

Недавно веб-сайты состояли только из HTML-страниц, но сейчас для разработки крупных проектов используется система "клиент-сервер веб-приложение. Главной особенностью веб-приложения является возможность обмена, обработки и изменения информации. Такое взаимодействие можно представить только с использованием баз данных, которые позволяют управлять большим количеством информации.

Целью проекта является разработка базы данных для клиент-серверного приложения, которое предоставляет возможность управления данными о товарах, находящихся на складе.

Для достижения цели поставлены следующие задачи:

1. формализовать задачу и определить необходимый функционал;
2. провести анализ существующих СУБД;
3. спроектировать базу данных;
4. выбрать подходящие языки программирования, спроектировать архитектуру программы;
5. реализовать пользовательский интерфейс.

1 Аналитическая часть

В данном разделе проводится обзор архитектуры программного комплекса, анализ существующих моделей баз данных и выбор наиболее подходящего для решения поставленных задач. Обозреваются трехуровневоя и двухзвенная архитектуры, дореляционная, реляционная и постреляционная модели а также производтся сравнение этих моделей с целью выбора наиболее подходящей и эффективной для реализации поставленных цели и задачей.

1.1 Формализация задачи

В соответствии с поставленными целями необходимо разработать базу данных для клиент-серверного веб-приложения для складского учета. Для каждого пользователя предусмотрен свой набор функций.

Представитель компании:

1. вход в аккаунт с помощью логина и пароля;
2. просмотр информации о товарах компании на складе;

Работник склада:

1. вход в аккаунт с помощью логина и пароля;
2. поиск и просмотр информации о различных товарах;
3. добавление, редактирование и удаление информации о товарах на складе;

1.2 Аналоги приложения

На рынке уже существует множество различных приложений для складского учет, но наиболее широко используемы это ЕКАМ и 1С Торговля и склад. Преимуществами ЕКАМ является интеграция с контрольно-кассовой техникой и торговым оборудованием а также удобный и современный интерфейс. Преимуществами 1С являются раздельное ведение учета разных направлений и автоматизация документаоборота. Для требуемого приложении будут взяты одни из особенностей ЕКАМ и 1С, чтобы при дальнейших разработке и улучшении приложения оно было конкурентоспособным.

1.3 Трехуровневая архитектура

Трехуровневая архитектура приложений — это модульная клиент-серверная архитектура, которая состоит из уровня представления, уровня приложения и уровня данных.

Уровень представления является клиентским приложением, которое переносит задачи по обработке информации на сервер. Примером этого уровня может служить компьютер с браузером, на котором отображается веб-приложение.

Серверная часть располагается на уровне приложения, в ней сосредоточена большая часть бизнес-логики. Обработка информации также происходит на уровне сервера.

Сервер базы данных – это уровень данных, реализуется, как правило, средствами систем управления базами данных, подключение к этому компоненту обеспечивается только с уровня сервера приложений.

На рисунке 1.1 представлен пример трёхуровневой архитектуры.

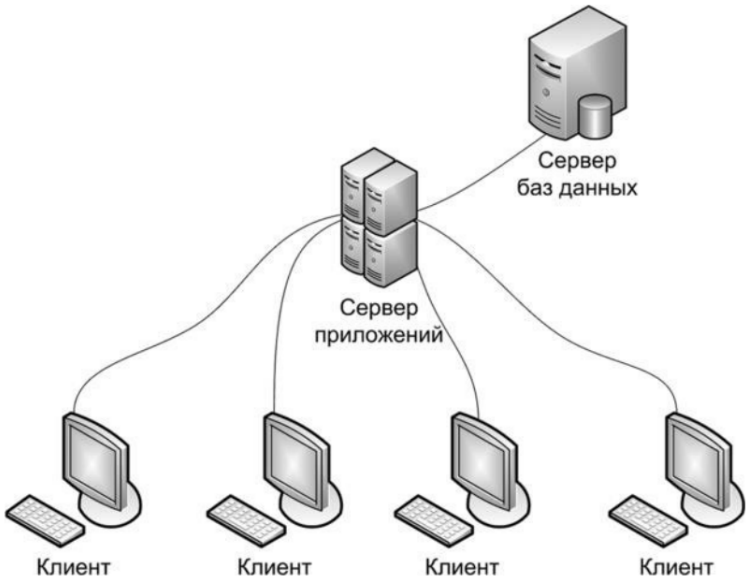

Рисунок 1.1 – Пример трёхуровневой архитектуры

В простейшей конфигурации сервер приложения и сервер базы данных может находится на одном компьютере. Однако с точки зрения безопасности и масштабируемости сервер базы данных должен находится на отдельном устройстве, к которому по сети подключаются остальные серверы приложений.

Достоинства трёхуровневой архитектуры:

1. масштабируемость – при необходимости способность выдержать увеличение нагрузки;
2. конфигурируемость - изолированность уровней друг от друга позволяет быстро переконфигурировать систему при возникновении сбоев;
3. низкие требования к скорости сети между клиентами и сервером.

Недостатки:

1. более высокая сложность создания приложения;
2. высокие требования к производительности серверов приложений и базы данных.

1.4 Двухзвенная клиент-серверная архитектура

В архитектуре "клиент-сервер"программное обеспечение разделено на две части - клиентскую часть и серверную часть. Задача клиентской части (программы-клиента) состоит во взаимодействии с пользователем, передаче пользовательского запроса серверу, получение запроса от серверной части (программы-сервера) и представление его в удобном для пользователя виде. Программа-сервер же обрабатывает запросы клиента и выдает ответы.

На рисунке 1.2 представлена двухзвенная клиент-серверная архитектура.

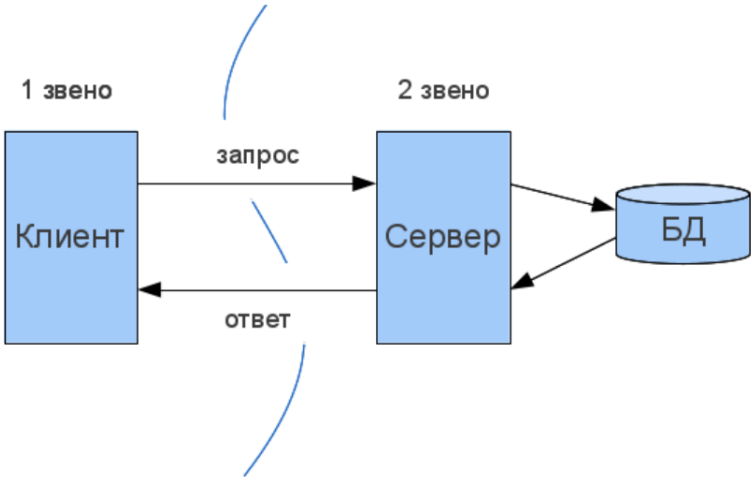

Рисунок 1.2 – Двухзвенная клиент-серверная архитектура

Двухзвенная архитектура используется в клиент-серверных системах, где сервер отвечает на клиентские запросы напрямую и в полном объеме, при этом используя только собственные ресурсы. Т.е. сервер не вызывает сторонние сетевые приложения и не обращается к сторонним ресурсам для выполнения какой-либо части запроса.

Достоинства двухзвенной архитектуры:

1. нет дублирования кода;
2. безопасность персональных данных;
3. нет необходимости обслуживания нескольких серверов;
4. существенная часть проектных задач становится уже решенной.

Недостатки:

1. при неработоспособности сервера клиенты также не смогут продолжать работу.

1.5 Виды СУБД

Системы управления базами данных (СУБД) – это высокоуровневое программное обеспечение, работающее с низкоуровневыми API. Для решения проблем создавались различные виды СУБД. Два основных направления – реляционные (SQL) и нереляционные (NoSQL). Их сравнение представлено в таблице 1:

	SQL	NoSQL
Структура хранения	Однозначно определенная (двумерные таблицы)	Нет ограничений
Поддержка	Широко распространенная СУБД	Только набирающая популярность
Запросы	При помощи языка SQL	База реализует свой способ работы с данными
Объединение таблиц	с помощью join	Join отсутствует
Структура	Статическая	Статическая или динамическая

Таблица 1

Данные в работе заранее определены и должны иметь четкую структуру, поэтому в работе используется реляционная модель.

1.6 Модели БД

Основной состовляющей базы данных является модель данных. Модель данных – это объединение структур данных и операций обработки. С помощью модели данных представляются информационные объекты и взаимосвязи между ними. Существует множество разновидностей баз данных, но их всех можно классифицировать по модели данных на три модели, каждая из которых может внутри себя классифицироваться на другие, схожие между собой модели данных.

1. дореляционная модель;
2. реляционная модель;
3. постреляционная модель.

1.6.1 Дореляционная модель

Дореляционная модель данных была определена еще до того, как реаляционная модель была широко использована. Дореляционная модель в основном основана либо на иерархической модели либо на сетевой модели, которые будут рассмотрены далее.

Сетевые базы данных относятся к теоретико-графовым моделям и является дореляционной моделью. Сети – это естественный способ представления отношений между объектами базы данных и связей между этими объектами(таблицы баз данных или сущности). Сетевую модель можно представить в виде ориентированного графа, который состоит из узлов и ребер. Узлы направленного графа – это ни что иное, как объекты сетевой базы данных, а ребра такого графа показывают связи между объектами сетевой модели данных, причем ребра показывают не только саму связь, но и тип связи (связь один к одному или связь один ко многим), это продемонстрировано на рисунке 1.3.

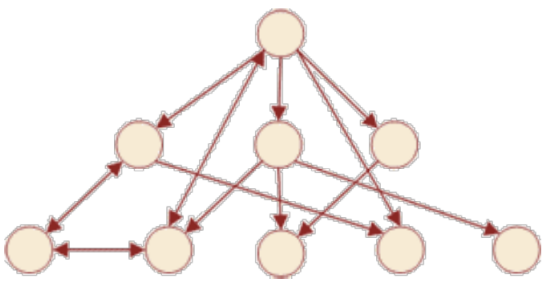

Рисунок 1.3 – Пример структуры сетевой модели БД

Достоинством такой модели является возможность строить множественные связи между объектами. Однако главным недостаток остается невозможность изменить структуру после ввода данных.

Иерархическая модель БД также опирается на теорию графов, так как её можно назвать частным случаем сетевой модели. В её основе лежит древовидная структура, где между объектами существуют связи, каждый объект может включать в себя несколько объектов более низкого уровня. Такие объекты находятся в отношении предка (объект более близкий к корню) к потомку (объект более низкого уровня), это продемонстрировано на рисунке 1.4.

Иерархическая модель появилась раньше сетевой, однако она более простая, а вследствие менее эффективная.

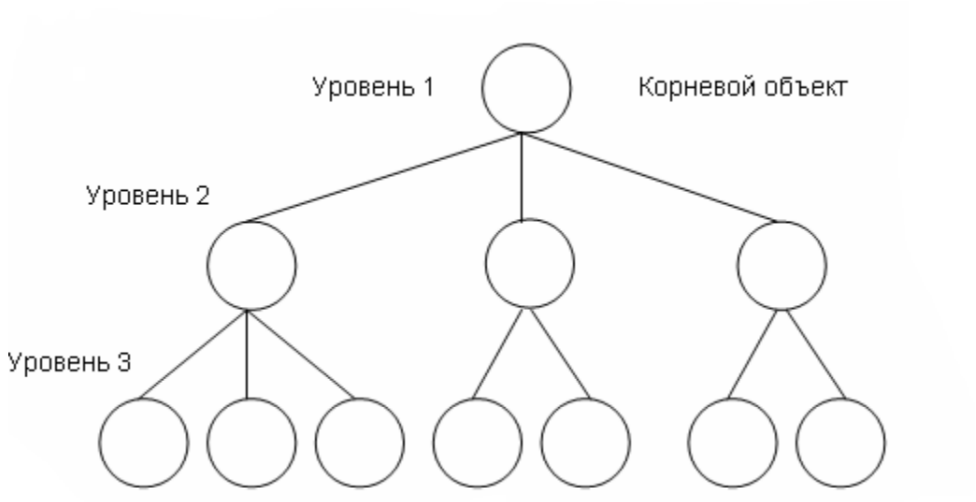

Рисунок 1.4 – Пример структуры иерархической модели БД

1.6.2 Реляционная модель

Реляционная модель есть представление БД в виде совокупности упорядоченных нормализованных отношений. Для реляционных отношений характерны следующие особенности:

1. любой тип записи содержит только простые (по структуре) элементы данных;
2. порядок кортежей в таблице несуществен;
3. упорядочение значащих атрибутов в кортеже должно соответствовать упорядочению атрибутов в реляционном отношении;
4. любое отношение должно содержать один атрибут или более, которые вместе составляют уникальный первичный ключ;
5. если между двумя реляционными отношениями существует зависимость, то одно отношение является исходным, второе - подчиненным;
6. чтобы между двумя реляционными отношениями существовала зависимость, атрибут, служащие первичным ключом в исходном отношении, должны также присутствовать в подчиненном отношении.

Преимуществом такой модели является простота реализации и независимость данных друг от друга. Недостатком можно считать то, что расходуется много памяти для поддержания всех данных в виде таблиц, а также низкая скорость обработки информации.

На рисунке 1.5 представлен пример реляционной модели БД.

Рисунок 1.5 – Пример структуры реляционной модели БД

1.6.3 Постреляционная модель

Постреляционная модель фактически является расширением реляционной модели, так как она снимает ограничения на неделимость данных в следствие чего могут быть использованы многозначные поля значения которых состоят из подзначений, и этот набор значений воспринимается как самостоятельная таблица, внутри главной таблицы. В качестве примеров будут рассмотрены модель ключ-значение, документо-ориентированная модель и многомерная модель.

Информация в многомерной модели представляется в виде многомерных массивов, называемых гипперкубами. В одной такой базе данных, построенной на многомерной модели, может храниться множество гипперкубов, на основе которых есть возможность проводить совместный анализ показателей. Конечный пользователь в качестве внешней модели данных получает для анализа определенные срезы, проекции кубов, представляемые в виде обычных двумерных таблиц или графиков.

По сравнению с реляционной моделью данных многомерная организация данных обладает более высокой информативностью. Для того чтобы убедиться в этом, рассмотрено многомерное представление данных и оно сопоставлено с реляционным (таблица 1.2 и рисунок 1.6).

Наименования продукта	Квартал	Выпуск
Сыр	I	20
Сыр	II	30
Сыр	III	25
Сыр	IV	15
Творог	I	20
Творог	II	25
Масло	III	15

Таблица 1.2. Реляционное представление

В примере на рисунке 1.7 каждое значение выпуска однозначно определяется комбинацией временного отрезка(квартала), именованием товара и названием цеха.

Наименование продукта	Выпуск по кварталу			
	I	II	III	IV
Сыр	20	30	25	15
Творог	20	25		
Масло			15	

Рисунок 1.6 – Многомерное представление

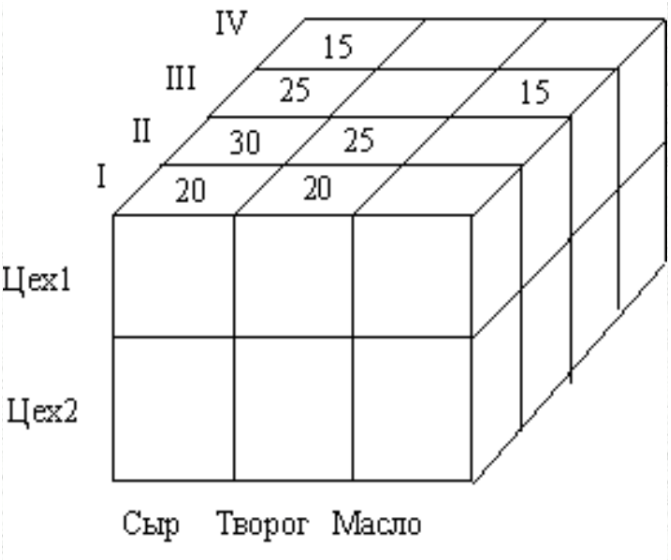

Рисунок 1.7 – Пример трехмерной модели

Модель ключ-значение является самой простой моделью, имеет быструю запись, также хранит хеш-таблицу ключей, где каждый ключ связан с непрозрачным бинарным объектом, также хорошо индексируется. Чаще всего применяется для хранения и работы со значениями датчиков, курсами акций и результатов промежуточной обработки. Структура данной модели представлена на рисунке 1.8.

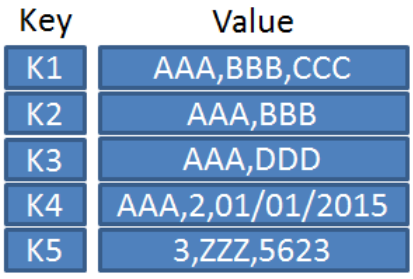

Рисунок 1.8 – Пример ключ-значение

Самой популярной СУБД для модели ключ-значение является Redis. Строки в Redis представляют из себя бинарно-безопасные строки, то есть строка Redis может содержать любые данные, от буквенно-цифровых символов до изображений JPEG.

Документоориентированная модель предназначеная для хранения иерархических структур, то есть документов. Структурно СУБД с такой моделью представляют из себя документные хранилища со структурой дерева или леса. Исходя из названия, документоориентированная модель чаще всего применяется для хранения документов. На рисунке 1.9 изображен пример структуры документа, где видна структура дерева.

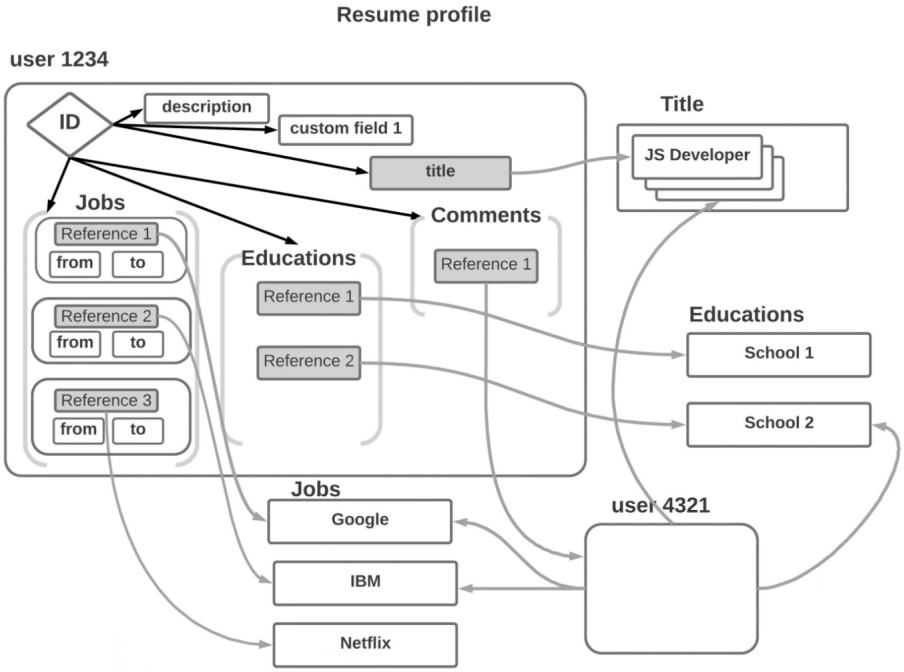

Рисунок 1.9 – Пример документоориентированной модели

Одной из самых популярных документоориентированнхы СУБД является MongoDB. Она написана на C++ и использует документы подобные JSON.

Таким образом, для сравнения всех моделей можно обратиться к таблице 1.2 :

	Дореляционная	Реляционная	Постреляционная
Актуальность (количество используемых решений в современных проектах и рейтинг TOPDB)	низкая	высокая	средняя
Изменяемая структура	нет	да	да

Таблица 1.2

Сравнить скорость выполнения запросов между данными моделями довольно проблематично, так как дореляционная модель во многом зависит от реализации, а в постреляционной модели скорость может различаться между такими моделями как документоориентированная и ключ-значение, так как они принципиально разные и используются для разных задач.

1.7 Вывод

Исходя из рассмотренных ранее вариантов архитектуры и моделей данных, и поставленных цели и задач, наиболее оптимальным решением станет комбинация реляционной модели базы данных и двухзвенной клиент-серверной архитектуры, так как это позволит реализовать поставленные цель и задачи, не затрачивая при этом большие вычислительные мощности, не усложняя программную архитектуру и сохраняя весь необходимый функционал.

2 Конструкторская часть

В этом разделе будут представлены различные диаграммы и будет спроектирована база данных. Представлены таблицы базы данных, которые необходимы для реализации складского приложения, для хранения информации об автомобильных запчастях различных компаний, хранящихся на складе. Также будет рассмотрен фреймворк Spring, как наиболее подходящий для реализации поставленных цели и задачей и будет рассмотрен MVC.

2.1 Use-Case Диаграмма

На рисунке 2.10 представлена Use-Case диаграмма. В работе необходимо реализовать три роли: представитель компании, работник склада и администратор.

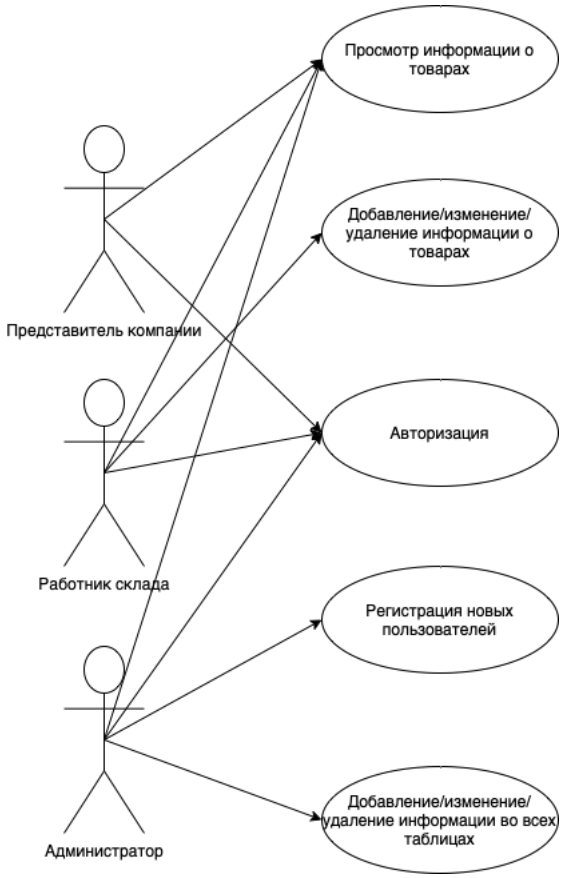

Рисунок 2.10 – Диаграмма вариантов использования

Представитель компании может просматривать информацию о товарах своей компании, а так же входить в систему с использованием логина и пароля.

Работник склада может искать информацию о различных товарах по ключевому слову, добавлять/изменять/удалять информацию о различных товарах и входить в систему с использованием логина и пароля.

Администратор может все то же самое что и работник склада, а так же может регестрировать новых пользователей.

2.2 Проектирование базы данных

База данных для данного проекта состоит из трех основных таблиц. В таблице Item должна храниться вся необходимая информация о товаре, то есть автомобильных запча-

стях: название, количество товара(в штуках) в упаковке, количество упаковок, артикул товара, номер накладной, вес товара(упаковки), а так же номер компании, которой принадлежит данный товар и номер полки, на которой хранится данный товар. Все эти данные минимально необходимы для складского учета. В таблице Shelf хранится информация о полках, на которых хранятся товары: номер полки(название), максимально допустимый вес и свободный, который изменяется в зависимости от того, сколько товаров и с каким весом находятся на полке. В таблице User хранится информация о пользователях: логин, пароль и роль. Данное проектирование базы данных исключает лишние связи между таблицами и отлично подходит для складского учета.

1. Item — таблица с информацией о различных товарах;
2. User — таблица с информацией о пользователях;
3. Shelf — таблица с информацией о полках;

На рисунке 2.2 изображена ER-диаграмма.

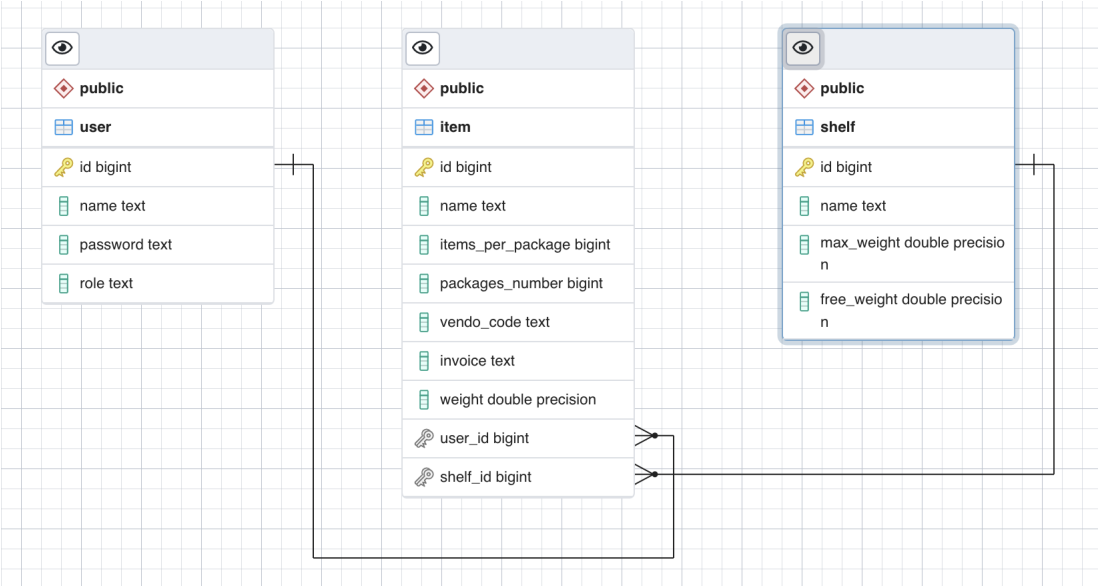

Рисунок 2.11 – ER-диаграмма

Информация хранимая в таблице Item:

1. id — целочисленное поле, идентификационный номер товара;
2. name — символьное поле, название товара;
3. itemsPerPackage— целочисленное полео, количество товаров в упаковке;
4. packagesNumber— целочисленное поле, количество упаковок;
5. vendorCode— символьное поле, артикул;
6. invoice— символьное поле, номер накладной;
7. weight— вещественное число, вес товара;
8. shelfId — целочисленное поле, идентификационный номер полки, на которой хранится товар;

9. userId — целочисленное поле, идентификационный номер компании-владельца товара.

Информация хранимая в таблице User:

1. id — целочисленное поле, идентификационный номер пользователя;
2. name — символьное поле, логин пользователя;
3. password — символьное поле, пароль пользователя;
4. role— символьное поле, роль пользователя.

Информация хранимая в таблице Shelf:

1. id — целочисленное поле, идентификационный номер заметки;
2. number — символьное поле, номер полки;
3. maxWeight — вещественное число, максимальный вес который может находиться на полке;
4. freeWeight— вещественное число, вес, сколько можно еще добавить на полку.

2.3 Spring Boot и модели

В spring jdbc модели отображают информацию о данных. Они содержат поля и поведение данных, при этом одна модель соответствует одной таблице в базе данных.

Spring Boot – это один из многочисленных проектов экосистемы Spring, но в отличие от большинства своих «собратьев» он не решает какую-либо конкретную задачу, а представляет собой скорее новый этап развития Spring в целом [1].

Цель Spring Boot состоит в том, чтобы упростить процесс разработки приложений на основе Spring при помощи их создания на основе уже готовых наборов программных компонентов (так называемых, «starter» пакетов), которые уже включают набор того, что необходимо для решения той или иной задачи и сконфигурированы соответствующим образом.

На рисунке 2.12 представлена схема архитектуры Spring.

2.4 MVC

MVC, то есть Model View Controller предполагает разделение приложения на три отдельных компонента: Модель (Model), Представление (View), Контроллер (Controller). Это позволяет производить модификацию какого-либо компонента независимо от других.

Model - компонент бизнес-логики приложения, предоставляет данные и методы работы с ними.

View - компонент, который отвечает за взаимо действие с пользователем, необходим для отображения данных, полученных в результате работы модели.

Controller - компонент, отвечающий за обработку действий пользователя, перенаправляет данные от пользователя к модели и наоборот.

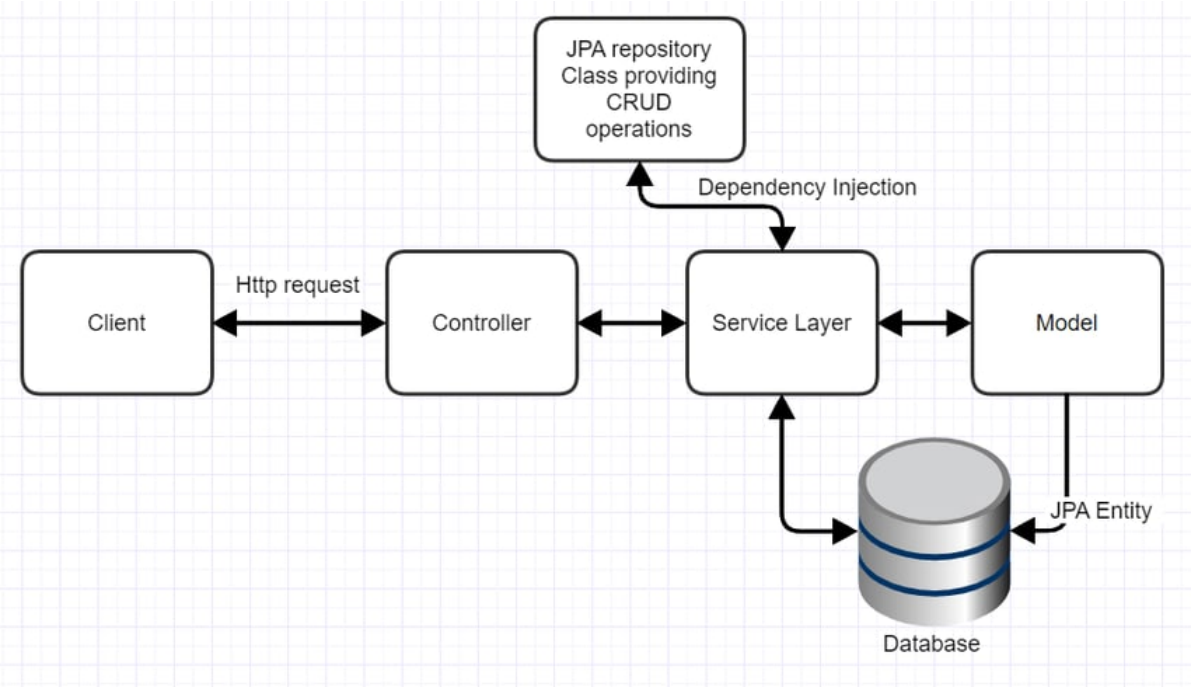

Рисунок 2.12 – Схема архитектуры Spring

2.5 Вывод

Была представлена Use-Case диаграмма приложения для представителя компании, работника склада и администратора, была спроектирована база данных и описаны все таблицы необходимые выполнения поставленных цели и задач, и спроектированы таким образом, чтобы в них хранилась вся необходимя информация о товарах на складе для автомобильных запчастей и никакой другой излишней, описаны Spring framework, MVC и была показана диаграмма архитектуры приложения.

3 Технологическая часть

3.1 Средства реализации

В качестве языка программирования для реализации программы были выбраны языки Java [2] и JavaScript, так как Java является одним из наиболее популярных и удобных языков для back-end приложений [3]., а у JavaScript [4]. есть фреймворк VueJS [5]., который является очень гибким и удобным для использования конечным пользователем, а так же с его помощью front-end приложение легко настраивается для различных устройств. В качестве среды разработки для реализации программы были выбраны продукты jetBrains: IntellijIDEA и WebStorm [6]. Они позволяет быстро и комфортно работать с приложениями, вести контроль версий а так же работать с большим количеством классов.

3.2 Выбор СУБД

Для проекта были рассмотрены три СУБД: PostgreSQL, MySQL и SQLite.

SQLite — это библиотека, встраиваемая в приложение, которое ее использует. Будучи файловой БД, она предоставляет отличный набор инструментов для более простой (в сравнении с серверными БД) обработки любых видов данных.

Когда приложение использует SQLite, их связь производится с помощью функциональных и прямых вызовов файлов, содержащих данные (например, баз данных SQLite), а не какого-то интерфейса, что повышает скорость и производительность операций.

Преимущества:

1. файловая: вся база данных хранится в одном файле, что облегчает перемещение бд;
2. стандартизированная: SQLite использует SQL;
3. отлично подходит для разработки и тестирования: во время этапа разработки большинству требуется масштабируемое решение.

Недостатки:

1. отсутствие пользовательского управления: многие БД предоставляют пользователям возможность управлять связями в таблицах в соответствии с привилегиями, но у SQLite такой функции нет;

MySQL — это самая популярная из всех крупных серверных БД. Приложения общаются с базой данных через процесс-демон.

Преимущества:

1. простота: Существует много сторонних инструментов, включая визуальные, облегчающих начало работы с БД;
2. мощность и масштабируемость: MySQL может работать с действительно большими объемами данных, и неплохо походит для масштабируемых приложений;
3. скорость: пренебрежение некоторыми стандартами позволяет MySQL работать производительнее.

Недостатки:

1. известные ограничения: по определению, MySQL не может сделать все, что угодно, и в ней присутствуют определенные ограничения функциональности;
2. застой в разработке: хотя MySQL и является open-source продуктом, работа над ней сильно заторможена.

PostgreSQL — это СУБД, ориентирующаяся в первую очередь на полное соответствие стандартам и расширяемость. PostgreSQL, или Postgres, пытается полностью соответствовать SQL-стандартам ANSI/ISO.

Преимущества:

1. поддержка сообществом;
2. поддержка сторонними организациями: несмотря на очень продвинутые функции, PostgreSQL используется в многих инструментах, связанных с РСУБД;
3. PostgreSQL можно программно расширить за счет хранимых процедур;

Недостатки:

1. производительность: В простых операциях чтения PostgreSQL может уступать конкурентам;

Собирая основные характеристики в одну таблицу:

	SQLite	MySQL	PostgreSQL
Серверная	нет	да	да
Скорость (относительно друг друга) [7]	высокая	низкая	средняя

Таблица 3.1

Таким образом, в качестве СУБД подходят MySQL и PostgreSQL, но PostgreSQL работает быстрее, поэтому была выбрана именно PostgreSQL [8]..

3.3 Реализация моделей хранения данных

Листинг для класса Item:

```
@Entity
@Data
@Table(name = "item")
public class Item {
  @Id
  @GeneratedValue(strategy = GenerationType.SEQUENCE, generator = "
     item_id_gen")
  @SequenceGenerator(name="item_id_gen", sequenceName = "item_id_seq",
     allocationSize = 1)
  private Long id;
  private String name;
  private Long itemsPerPackage;
  private Long packagesNumber;
  private String vendorCode;
  private String invoice;
  private Double weight;
  @ManyToOne
  @JoinColumn(name = "shelf_id")
  private Shelf shelf;
  private Long userId;
}
```

Листинг для класса Shelf:

```
  @Data
  @Entity
  @Table(name = "shelf")
  public class Shelf {
    @Id
    @GeneratedValue(strategy = GenerationType.SEQUENCE, generator = "
       shelf_id_gen")
    @SequenceGenerator(name="shelf_id_gen", sequenceName = "shelf_id_seq",
       allocationSize = 1)
    private Long id;
    private String number;
    private Double maxWeight;
    private Double freeWeight;
  }
```

Листинг для класса User:

```
@Data
@Entity
@Table(name = "\"user\"")
public class User implements UserDetails {
  @Id
  @GeneratedValue(strategy = GenerationType.SEQUENCE, generator = "
     user_id_gen")
  @SequenceGenerator(name="user_id_gen", sequenceName = "user_id_seq",
     allocationSize = 1)
  private Long id;
  @Column(name = "name")
  private String username;
  private String password;
  private String role;
  @Transient
  private boolean accountNonExpired = true;
  @Transient
  private boolean accountNonLocked = true;
  @Transient
  private boolean credentialsNonExpired = true;
  @Transient
  private boolean enabled = true;

  @Override
  public Collection<? extends GrantedAuthority> getAuthorities() {
    if (role == null) {
      return Collections.emptyList();
    }
    return List.of(new SimpleGrantedAuthority(role));
  }
}
```

Роли были добавлены в базу данных с помощью Liquibase. Доступы пользователей к таблицам предоставлены таким образом, чтобы пользователи получали доступ к тем таблицам, представленным на ER-диаграмме, которые им необходимы для всех доступных им действиям:

```
  <sql>
  CREATE ROLE wh_worker WITH LOGIN
  PASSWORD 'worker_pass' CONNECTION LIMIT -1;
  GRANT ALL ON shelf TO wh_worker;
  GRANT ALL ON item TO wh_worker;
  GRANT USAGE, SELECT ON SEQUENCE shelf_id_seq TO wh_worker;
  GRANT USAGE, SELECT ON SEQUENCE item_id_seq TO wh_worker;
  </sql>
  <sql>
  CREATE ROLE wh_representative
  WITH LOGIN
  PASSWORD 'rep_pass' CONNECTION LIMIT -1;
  GRANT SELECT ON item TO wh_representative;
  GRANT SELECT ON shelf TO wh_representative;
  </sql>
  <sql>
```

```
CREATE ROLE wh_admin
WITH LOGIN SUPERUSER
PASSWORD 'admin_pass'
CONNECTION LIMIT -1;
</sql>
```

3.4 Реализация контроллера

Листинг контроллера для Title, где представлены все необходимые запросы для представителей компаний и работников склада, которые представляю поиск, удаление, добавление и обновление информации о товарах на складе:

```
@RestController
@RequestMapping("/item")
@RequiredArgsConstructor
public class ItemController {
  private final ItemService itemService;

  @GetMapping
  public List<Item> getAll() {
    return itemService.findAllByUserId();
  }

  @GetMapping("/{id}")
  public Item getById(@PathVariable Long id) {
    return itemService.findById(id);
  }

  @GetMapping(params = {"name"})
  public List<Item> getAllByName(@RequestParam String name) {
    return itemService.findAllByName(name);
  }

  @GetMapping(params = {"vendorCode"})
  public List<Item> getAllByVendorCode(@RequestParam String vendorCode)
     {
    return itemService.findAllByVendorCode(vendorCode);
  }

  @DeleteMapping("/{id}")
  public void deleteById(@PathVariable Long id) {
    itemService.deleteById(id);
  }

  @PostMapping
  @ResponseStatus(HttpStatus.CREATED)
  public Item create(@RequestBody ItemDto itemDto) {
    return itemService.save(itemDto);
  }

  @PostMapping("/{id}")
  public Item update(@PathVariable Long id, @RequestBody ItemDto itemDto
     ) {
```

```
        return itemService.update(id, itemDto);
    }
}
```

3.5 Триггер

Реализация триггера, для таблицы Shelf, который обновляет значение freeWeight, когда добавляется запись в таблицу Item. Хранение информации о доступном весе крайне важна с точки зрения безопасности:

```
<createProcedure>
CREATE OR REPLACE FUNCTION insert_free_weight_trigger()
RETURNS TRIGGER AS
$func$
BEGIN
UPDATE shelf SET free_weight = free_weight - NEW.weight WHERE id = NEW.
    shelf_id;
RETURN NEW;
END
$func$ LANGUAGE plpgsql;
</createProcedure>
<sql>
CREATE TRIGGER insert_free_weight
AFTER INSERT ON item
FOR EACH ROW
EXECUTE FUNCTION insert_free_weight_trigger();
</sql>
```

3.6 Интерфейс приложения

Интерфейс является важной составляющей складского приложения, так как люди работающие за многие годы уже привыкли к определенному расположению данных. Дизайн данного приложения содержит в себе элементы дизайна EKAM и 1C, все данные о товаре распологаются в одну строку, все необходимые поля для поиска находятся сверху, вход в аккаунт справа сверху, и также данные о товаре располагаются в таком порядке, чтобы вся информация шла в логическом порядке, сначала количество товара, затем артикул и номер накладной, потом вес и номер полки на которой находится, интерфейся изображен на рисунке 3.13.

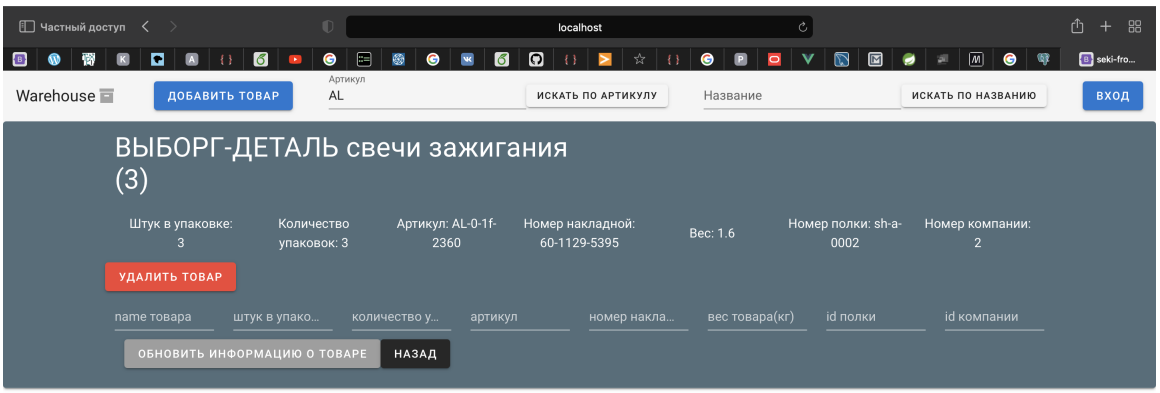

Рисунок 3.13 – Интерфейс программы

4 Эксериментальная часть

В эекспериментальной части будет определен набор простых или составных индексов для базовых таблиц. Индексы создаются в основном для тех таблиц, к котором чаще всего применяются запросы, в данном приложении запросы применяются к таблице item, для поиска по названию и артикулу, следовательно индексы следует создать именно для них.

```
CREATE INDEX name_index ON item (name);
CREATE INDEX vendor_code_index ON item (vendor_code);
```

Для сравнения скорости обработки запросов до и после добавления индексов, использовалась программа JMeter, для стресс-тестирования, было использовано десять потоков и замерено сто раз для каждого случая и вязты средние значения, результаты замеров продемонстрированы в таблице 4.1.

	Load time(мс)	Connect time(мс)	Latency(мс)
Без индекса	37,6	34,8	36,1
С индексом	36,1	33,2	34,7

Таблица 4.1

Исходя из результатов в таблице, можно сделать вывод, что добавление индексов ускорило время обработки запроса в среднем на полторы милисекунды, что достаточно неплохо, учитывая не самый большой объем данных в таблице item, две тысячи записей. Для приложений складского учета индексы для часто используемых таблиц очень важны, так как при увеличении количества записей в таблице время "выигранное"созданием индекса будет возростать, так же как и при увеличении количества потоков. Остается лишь недостаток индекса, индекс требует пересоздания при добавлении иили удалении записей из таблицы, но тот эффект который дает индекс для данной таблицы, перекрывает этот недостаток.

Заключение

В рамках курсового проекта была формализована задача и определен необходимый функционал, проведен анализ существующих СБУБД, спроектирована база данных для хранения информации о товарах, хранящихся на складе. Для реализации поставленной задачи были выбраны языки программирования Java и JavaScript. В качестве используемой СУБД была выбрана PostgreSQL, а для разработки приложения - Web-фреймворк VueJS. Была также спроектирована архитектура программы. Был реализован пользовательский интерфейс, созданы все таблицы, настроены тригеры, реализована аутентификация, ролевая модель на уровне базы данных. Также был определен набор индексов для таблицы item, и рассмотрены результаты после стресс-тестирования. В данной работе цель достигнута и все задачи выполнены.

В дальнейшем возможна модификация приложения, реструктуризация базы данных для увеличения обьема информации о товарах и полках, улучшения уровня безопасности. По сравнению со схожими широко-распространенными приложениями, данный проект имеет ряд недостатков, но так же есть и преимущества, такие как интуитивно понятный интерфейс, использование современной и широкоподдерживаемой PostgreSQL.

Список использованных источников

1. Spring Documentation. [Электронный ресурс] // Pivotal, Inc. 2022 11 мая URL: https://docs.spring.io/ (Дата обращения: 30.05.2022)

2. Сьерра Кэти [Sierra Kathy] Изучаем Java: пер. с англ. // Эксмо, 2012

3. Java Documentation [Электронный ресурс] // Oracle. 2022 27 марта URL: https://docs.oracle.com/en/java/ (Дата обращения: 01.06.2022)

4. What is Vue.js? [Электронный ресурс] // Vueschool. 2022 13 января URL: https://vuejs.org/v2/guide/ (Дата обращения: 16.04.2022)

5. Что такое Vue.js. Первое приложение. [Электронный ресурс] // Metanit 2017 29 августа URL: https://metanit.com/web/vuejs/1.1.php (Дата обращения: 27.05.2022)

6. JavaScript [Электронный ресурс] // Mozilla Corporation. 2022 27 марта URL: https://developer.mozilla.org/ru/docs/Web/JavaScript (Дата обращения: 28.3.2022)

7. Сравнение скорости выполнения SQL-запросов к разным БД. [Электронный ресурс] // Habr 2017 11 ноября URL: https://habr.com/en/sandbox/112090/ (Дата обращения: 30.05.2022)

8. PostgreSQL [Электронный ресурс] // The PostgreSQL Global Development Group. 2022 3 февраля URL: https://www.postgresql.org/docs/ (Дата обращения: 16.04.2022)