**3. Проектно-конструкторская часть**

# **3.1 Разработка структуры программной системы**

Приложение представляет собой клиент-серверное приложение, которое является четырёхуровневым. Клиент-серверное приложение – это вычислительная или сетевая архитектура, в которой задания или сетевая нагрузка распределены между поставщиками услуг, называемыми серверами, и заказчиками услуг, называемыми клиентами. Фактически клиент и сервер — это программное обеспечение.

Четырёхуровневое клиент-серверное (рисунок 4) приложение содержит в себе:

* Клиент, который представляет пользовательский интерфейс для взаимодействия с остальными уровнями приложения. В данной курсовой работе клиентом будет являться веб-браузер, который будет передавать веб-серверу запросы на получение HTML-страниц, изображений, файлов и других данных, обозначенных URL-адресами.
* HTTP сервер, который принимает HTTP-запросы от клиентов, обычно веб-браузеров, и выдающий им HTTP-ответы, как правило, вместе с HTML-страницей, изображением, файлом, медиа-потоком или другими данными,
* Сервер приложения, на котором осуществляется работа с серверными компонентами разрабатываемой системы. На данном уровне осуществляется вся бизнес-логика и управление базой данных.
* Сервер базы данных, выполняет обслуживание и управление базой данных и СУБД, а также отвечает за целостность и сохранность данных и обеспечивает операции ввода-вывода при доступе клиента к информации.

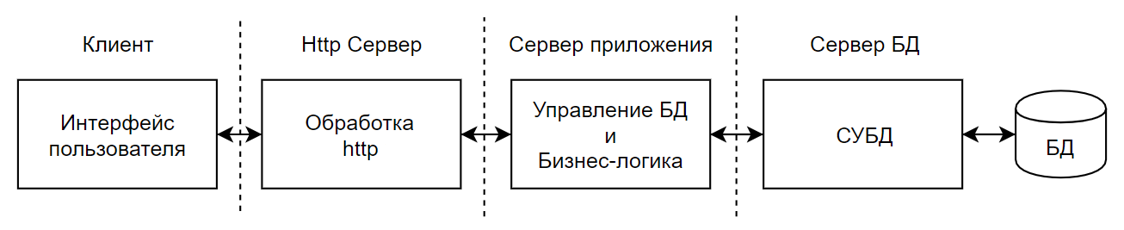


Рисунок 4 – Компоненты системы

Последовательность разработки системы:

1) Проектирование базы данных.

2) Создание логики серверной части.

3) Создание интерфейса.

4) Заполнение базы данных.

5) Тестирование.

Основным бизнес-процессом в заведениях, а соответственно и в разрабатываемой системе является обслуживание пользовательских заказов. Для удобства формализации бизнес-процессов и действий проектируемой системы использовалась нотация IDEF0.

Контекстная диаграмма – это модель, которая представляет из себя набор иерархических действий. Каждое действие преобразует некоторый объект или набор объектов. Самое высшее действие иерархии называется действием контекста – это уровень, описывающий непосредственно систему. Уровни ниже являются декомпозициями и представляют из себя подпроцессы родительского действия.

На рисунке 5 изображена контекстная диаграмма, описывающая процесс обработки заказа. Входным параметром являются позиции товаров. Информация о доставке и база данных системы отвечают за управление в данном процессе. Механизмы и действующие лица для выполнения данной работы – система, продавец и пользователь. На выход данного процесса поступают доставленный товар и оценка покупателя.

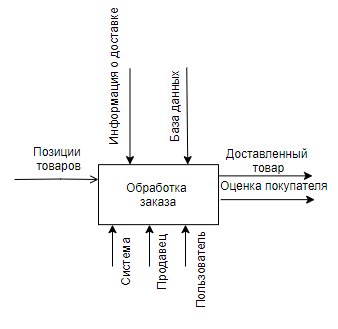


Рисунок 5 – IDEF0 контекстная диаграмма бизнес-процесса «обработки заказа»

На рисунке 6 представлена декомпиляция функции «обработка заказа» для описания каждой подсистемы и их взаимодействие. Процесс обработки заказа можно разделить на пять последовательных этапов:

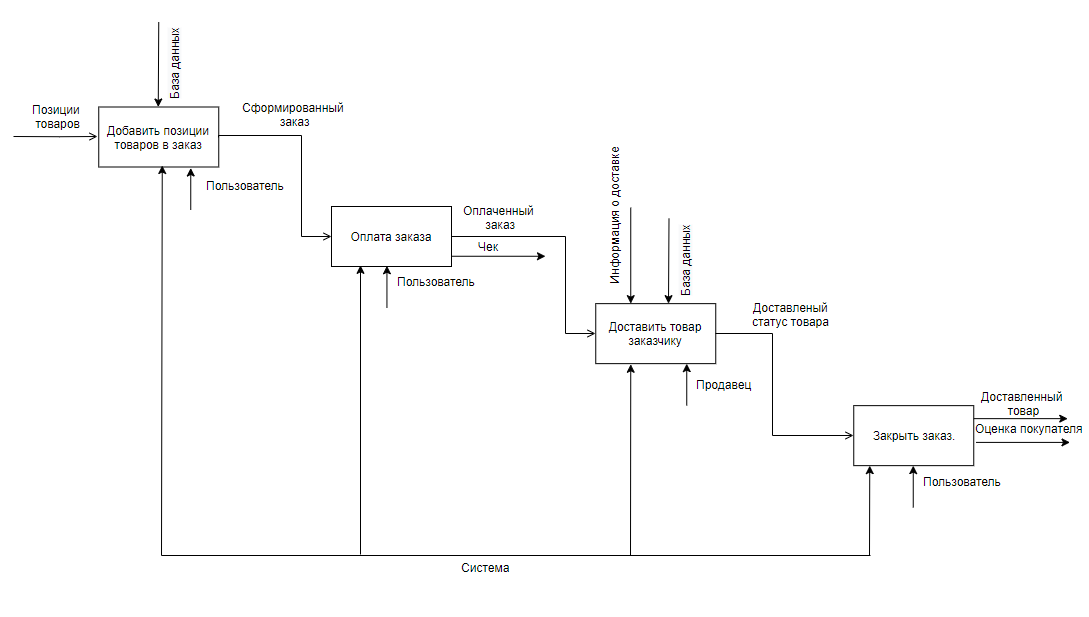


Рисунок 6 – IDEF0 диаграмма декомпозиции бизнес-процесса «обработки заказа»

1. Создание заказа.

На данном этапе система добавляет все созданные позиции с товарами в один заказ. Входным параметром являются позиции заказа. За управление в данном процессе отвечает база данных. Механизмы и действующие лица для выполнения данной работы – система и пользователь. В результате выполнения данного процесса на выход поступает сформированный заказ.

1. Оплата товара.

После того как пользователь закончить составлять свой заказ система предоставляет пользователю счет, который тот должен оплатить. Входным параметром является сформированный заказ. Механизмы и действующие лица для выполнения данной работы – это пользователь и система. В результате выполнения работы данного процесса поступают оплаченный заказ, а также чек.

1. Доставка товара.

После получения информации о том, что заказ был оплачен система уведомляет продавца об отправке товара выбранным способом доставки заказчику. За управление в данном процессе отвечает база данных и информация о доставке. Входным параметром является оплаченный заказ. Механизмы и действующие лица для выполнения данной работы - продавец и система. В результате выполнения работы данного процесса будет доставленный статус товара.

1. Закрытие заказа.

На последнем этапе пользователь получает товар, уведомляет систему о его получении и ставит оценку товару и магазину. Входным параметром является доставленный статус товара. Механизмы и действующие лица для выполнения данной работы – пользователь и система. Результатом выполнения работы данного процесса будет являться доставленный товар и оценка пользователя

# **3.2 Логическая схема данных**

Модель «сущность-связь» представляет собой графическую нотацию, которая основывается на блоках и соединительных линиях. С их помощью можно описать объекты модели и отношения между ними. ER имеет несколько основных понятий, таких как сущности, атрибуты и связи. На рисунке 6 представлена логическая схема базы данных разрабатываемой системы.

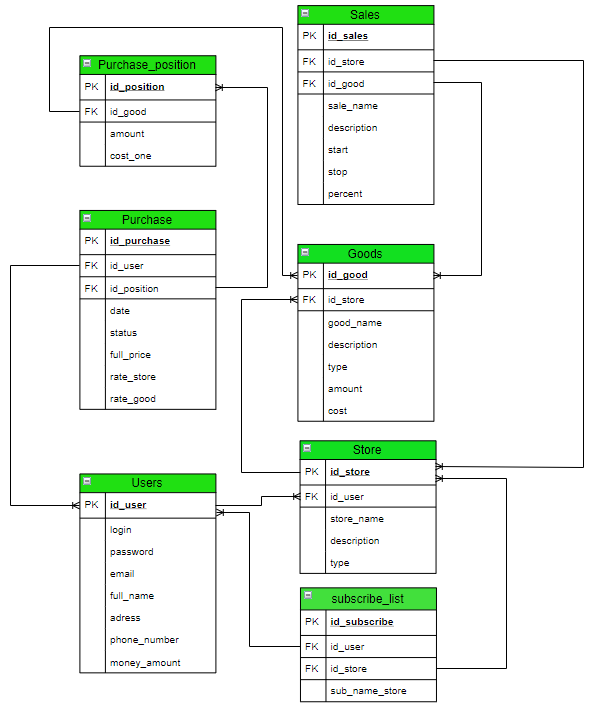


Рисунок 6 – Логическая схема базы данных

В результате анализа предметной области на данном этапе разработки системы были выделены следующие сущности:

Сущностью для хранения данных о покупателях и продавцах является Пользователь. Он содержит в себе такие атрибуты как фамилия, имя и отчество, адрес для доставки, контактный номер и денежный счет в онлайн-магазине. Будучи покупателем, пользователь может сделать множество заказов в различных магазинах, а также подписаться на новостную рассылку о новых товарах и акциях. Чтобы стать продавцом пользователь может создать несколько магазинов. Также есть поля для хранения данных о логине, пароле для входа на сайт и E-Mail для восстановления профиля. Наличие магазина у пользователя решает является ли пользователь продавцом

Следующей сущностью разрабатываемой системы является Магазин. Он содержит в себе атрибуты названия, описания и типа, означающий чем он занимается. В один магазин может быть добавлено множество товаров. Продавец–владелец от имени магазина может создавать акции на свои товары. Также магазина есть автоматическая рассылка уведомлений при создании новой акции и при добавлении нового товара.

Сущностью для наполнения магазина является Товар, реализующий поля для хранения данных о предметах, которые пользователи продают. Он содержит в себе название, описание, тип товара, стоимость одного экземпляра и количество на продажу. Если количество равно NULL, значит товар либо удален продавцом, либо скрыт от возможности покупки. Связь между товаром и магазином подразумевает, что у одного магазина может быть множество индивидуальных товаров, то есть один товар не может продаваться в двух магазинах.

Сущность Акция реализует поля для хранения данных описывающие скидки, которые делают продавцы в своих магазинах: название события, описание, дата начала и окончания акций и процент скидки на товары. Владелец может сделать множество акций на свои товары, при этом акция может принадлежать только одному магазину.

Для описания пользовательских заказов товаров есть сущность Заказ, хранящий в себе такие атрибуты как дата заказа, статус, который описывает состояние заказа, полная стоимость, оценка магазина и товара. Оценка будет играть ключевую роль для экспертной системы при составлении рекомендации на главном экране магазина индивидуально каждому пользователю. Заказ может быть только у одного пользователь, при этом в одном заказе может быть множество позиций заказа.

Сущность Позиция заказа описывает возможность наличия в одном заказе нескольких товаров. Атрибуты данной сущности хранят данные о количестве товара и его цене. В одном заказе может быть множество позиций, при этом в позиции заказа может быть только один товар.

На рисунке 7 изображена инициализация подключения к базе данных в настройках проекта Django.

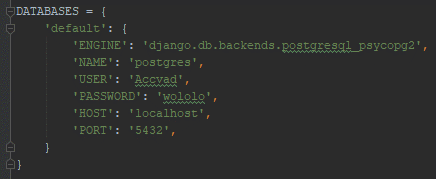


Рисунок 7 – Инициализация подключения к БД

Веб-приложения Django получают доступ и управляют данными через объекты Python, которые называются моделями. Модели определяют структуру хранимых данных, включая типы полей, их максимальный размер, значения по умолчанию и т. д. Определение модели не зависит от основной базы данных. После выбора базу данных, работать с ней напрямую не нужно. Можно написать свою структуру модели и код, а Django сделает всю работу, связанную с базой данных. На рисунке 8-9 изображены структуры данных пользователя, магазина и товаров.

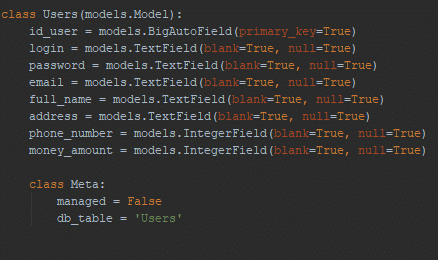


Рисунок 8 – Структура данных “Users”

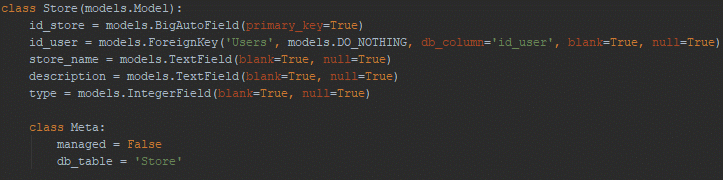


Рисунок 9 – Структура данных “Goods”

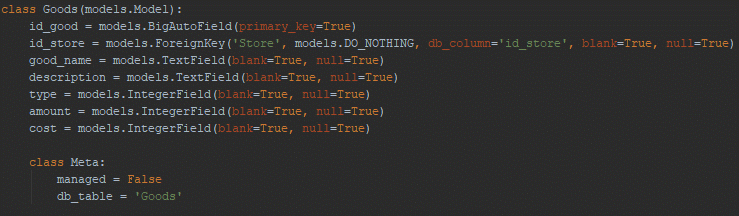


Рисунок 10 – Структура данных “Store”

**4.** **Проектно-технологическая часть**

# **4.1 Технологические решения, поддерживающие эксплуатационный цикл программы**

При проектировании системы был использован свободный фреймворк для веб-приложений на языке Python - Django. Он реализует архитектурный паттерн Model-View-Template, который является модификацией распространённого в веб-программировании паттерна Model-View-Controller.

Архитектура Model-View-Controller – это схема разделения данных приложения, пользовательского интерфейса и управляющей логики на три отдельных компонента: модель, представление и контроллер — таким образом, что модификация каждого компонента может осуществляться независимо.

Концепция MVC позволяет разделить модель, представление и контроллер на три отдельных компонента:

* Модель пре доступ к данным, обрабатывается слоем работы с базой данных. Он осуществляет запросы в базу данных и проверку на корректность. Модель не зависит от представления, а следовательно, не имеет возможности визуализировать данные и не имеет точек взаимодействия с пользователем. Модель строится таким образом, чтобы отвечать на запросы, изменяя своё состояние, при этом уведомляя «наблюдателей».
* Представление (View) отвечает за получение необходимых данных из модели и отправляет их пользователю.
* Контроллер обеспечивает общение между пользователем и системой. Он контролирует и направляет данные от пользователя к системе и наоборот.

Основная цель применения данной концепции состоит в отделении бизнес-логики от её визуализации. За счёт такого разделения повышается возможность повторного использования кода. Наиболее полезно применение данной концепции в тех случаях, когда пользователь должен видеть одинаковые данные в различны. В частности, выполняются следующие задачи:

* К одной модели можно присоединить несколько видов, при этом не затрагивая реализацию модели.
* Не затрагивая реализацию видов, можно изменить реакции на действия пользователя — для этого достаточно использовать другой контроллер;
* Ряд разработчиков специализируется только в одной из областей: либо разрабатывают графический интерфейс, либо разрабатывают бизнес-логику. Поэтому возможно добиться того, что программисты, занимающиеся разработкой бизнес-логики, вообще не будут осведомлены о том, какое представление будет использоваться.

Модель данного паттерна изображена на рисунке 11.

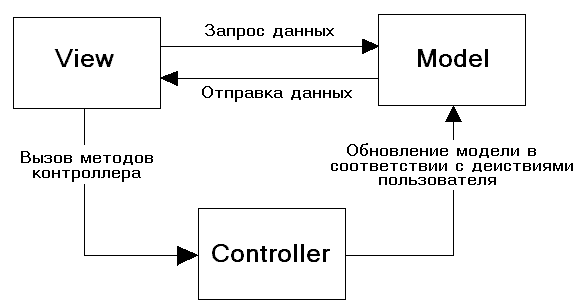


Рисунок 11 – Паттерн Model-View-Controller

Архитектура Model-View-Template незначительно отличается от Model-View-Controller. Основное различие между этими двумя шаблонами заключается в том, что Django сам обеспечивает общение между пользователем и системой, оставляя нас с шаблоном. Шаблон представляет собой файл HTML, смешанный с языком шаблонов Django.

Как и в MVC модель отвечает за бизнес-логику, методы, свойства и другие элементы, связанные с манипуляцией данными.

Представление (view) решает три задачи: принимает HTTP-запросы, реализует бизнес-логику, определённую методами и свойствами, отправляет HTTP-ответ в ответ на запросы. То есть он получает данные от модели и предоставляет шаблонам (templates) доступ к этим данным или предварительно обрабатывает данные и затем предоставляет к ним доступ шаблонам.

Шаблон (Template) — это текстовый файл, определяющий структуру или файлы с HTML-кодом, с полями для подстановки, используемыми для представления актуального содержимого. В Django реализован мощный движок шаблонов и собственный язык разметки. View может динамически создавать HTML страницы, используя HTML шаблоны и заполняя их данными из модели (model). Содержимое файлов может быть статическим или динамическим. Шаблоны не содержат бизнес-логики из-за чего они только отображают данные.

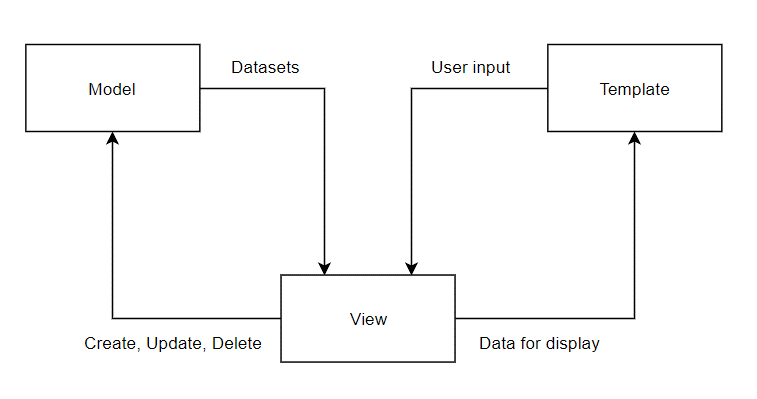


Рисунок 12 –Паттерн Model-View-Template