МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ  
Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ   
ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет Информационных технологий

Кафедра Информационных систем и технологий

Специальность 1-98 01 03 «Программное обеспечение информационной безопасности мобильных систем»

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА КУРСОВОГО ПРОЕКТА**

По дисциплине «Базы данных»

Тема: «Реализация базы данных для онлайн-маркетплейса с применением средств мониторинга состояния СУБД»

**Исполнитель**

студент 2 курса 7 группы \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Д.П. Нечай-Ницевич

подпись, дата

**Руководитель**

ассистент \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Е.И. Комарова

должность, уч. звание подпись, дата

Допущен к защите \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_

дата, подпись

Курсовой проект защищен с оценкой

Руководитель \_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_ Е.И. Комарова

подпись дата инициалы и фамилия

**Содержание**

[Введение 5](#_Toc135302831)

[1 Постановка задачи 6](#_Toc135302832)

[2 Проектирование базы данных 7](#_Toc135302833)

[2.1 Вывод проектирования базы данных 12](#_Toc135302834)

[3 Разработка объектов базы данных 13](#_Toc135302835)

[3.1 Роли и пользователи 13](#_Toc135302836)

[3.2 Таблицы 14](#_Toc135302837)

[3.3 Триггеры 17](#_Toc135302838)

[3.4 Процедуры 19](#_Toc135302839)

[3.4.1 Процедуры администратора 19](#_Toc135302840)

[3.4.2 Процедуры клиента 21](#_Toc135302841)

[3.4.3 Процедуры гостя 22](#_Toc135302842)

[3.4.4 Процедуры курьера 22](#_Toc135302843)

[3.5 Функции 22](#_Toc135302844)

[3.6 Индексы 24](#_Toc135302845)

[3.7 Последовательности 25](#_Toc135302846)

[4 Импорт и экспорт в JSON 26](#_Toc135302847)

[4.1 Импорт данных таблицы в формате JSON 26](#_Toc135302848)

[4.2 Экспорт данных таблицы в формате JSON 26](#_Toc135302849)

[5 Тестирование производительности 28](#_Toc135302850)

[5.1 Тестирование базы данных 29](#_Toc135302851)

[5.2 Тестирования приложения 30](#_Toc135302852)

[6 Описание технологии и ее применения в базе данных. 32](#_Toc135302853)

[6.1 Использование pgAdmin в качестве инструмента мониторинга СУБД..32](#_Toc135302854)

[6.2 Транзакции в секунду (Transactions per Second) 32](#_Toc135302855)

[6.3 Блоки записаны (Blocks I/О) 33](#_Toc135302856)

[6.4 Активность базы данных (Database Activity) 33](#_Toc135302857)

[6.5 Кортежи входящие/исходящие (Tuples In/Out) 33](#_Toc135302858)

[6.6 Вывод 34](#_Toc135302859)

[7 Краткое описание приложения для демонстрации 36](#_Toc135302860)

[7.1 Пример работы приложения от лица Администратора 36](#_Toc135302861)

[7.2 Пример работы приложения от лица Клиента 37](#_Toc135302862)

[8 Руководство пользователя 38](#_Toc135302863)

[8.1 Руководство использования приложения 38](#_Toc135302864)

[Заключение 41](#_Toc135302865)

[Список используемых источников 42](#_Toc135302866)

[Приложение А 43](#_Toc135302867)

[Приложение Б 48](#_Toc135302868)

[Приложение В 49](#_Toc135302869)

[Приложение Г 56](#_Toc135302870)

[Приложение Д 58](#_Toc135302871)

Введение

Онлайн-маркетплейсы стали неотъемлемой частью современного ритейла, предоставляя возможности для удобного выбора и покупки товаров, а также содействуя взаимодействию между продавцами и покупателями. Их популярность продолжает расти благодаря широкому ассортименту, гибким ценовым стратегиям и удобству использования. Это вызывает увеличение объема транзакций, а также количества пользователей и продавцов, что требует качественных технологических решений для поддержания высокого уровня обслуживания.

Ключевым элементом любого онлайн-маркетплейса является система управления базами данных (СУБД), которая обеспечивает хранение, обработку и защиту информации о продуктах, заказах, клиентах и курьерах. Учитывая значительные объемы информации, с которыми работают онлайн-маркетплейсы, разработка и внедрение базы данных такого проекта требует глубоких знаний и умений в области информационных технологий.

Кроме того, для обеспечения стабильной и эффективной работы системы необходимо использовать средства мониторинга состояния СУБД, позволяющие своевременно обнаруживать и устранять возможные проблемы. Такие инструменты помогают предотвратить возможные сбои, обеспечивают непрерывность и надежность работы системы, а также оптимизацию процессов обработки и хранения данных.

Задачи работы включают в себя анализ требований к базе данных, разработку структуры базы данных, создание модели данных, реализацию базы данных и ее тестирование.

Требования к курсовой работе включают в себя выполнение следующих шагов:

* изучение требований к базе данных ресторана;
* проектирование структуры базы данных, включая таблицы, связи и ограничения;
* реализация базы данных с использованием СУБД PostgreSQL;
* проведение тестирования базы данных.

Цель данного курсового проекта заключается в разработке базы данных для онлайн-маркетплейса с применением средств мониторинга состояния СУБД, обеспечивая эффективность, безопасность и надежность работы системы. Разработка такой базы данных будет включать не только проектирование структуры и создание объектов базы данных, но и разработку оптимальных алгоритмов обработки информации и управления ресурсами. Важным аспектом проекта также является создание интуитивно понятного и удобного пользовательского интерфейса для взаимодействия с базой данных.

# Постановка задачи

В рамках курсового проекта ставится задача спроектировать инфраструктуру базы данных для онлайн-маркетплейса. Необходимо проанализировать предметную область, определить основные объекты и их атрибуты, такие как продукты, заказы, клиенты, продавцы и курьеры. На основе полученных данных разрабатывается схема базы данных, включающая таблицы, связи между ними, ограничения целостности, профили безопасности, пользователи, триггеры, хранимые процедуры, функции и индексы.

Разработка объектов базы данных осуществляется с использованием СУБД PostgreSQL и языка SQL. Также предусмотреть заполнение таблиц данными, импортируемыми из различных источников, таких как JSON файлы.

Следует применить аналитические инструменты для изучения данных в базе данных онлайн-маркетплейса и выявления полезных функций и преимуществ. На основе полученных результатов разрабатывается приложение или веб-интерфейс для демонстрации функционала и возможностей базы данных. Приложение должно предоставлять возможности управления продуктами, заказами, клиентами, продавцами и курьерами, а также анализа работы онлайн-маркетплейса с использованием отчетов и других инструментов.

Дополнительно, неотъемлемым компонентом разрабатываемого проекта является интеграция технологии мониторинга состояния СУБД. С применением инструментов мониторинга, таких как PgAdmin, возможно наблюдение за разнообразными показателями производительности. К ним относятся параметры использования ЦПУ, памяти и дискового пространства, данные об активных сессиях и другие ключевые метрики. Благодаря данной технологии можно своевременно определить и устранить потенциальные проблемы, связанные с производительностью и стабильностью работы системы. Таким образом, обеспечивается повышение надежности и эффективности функционирования онлайн-маркетплейса.

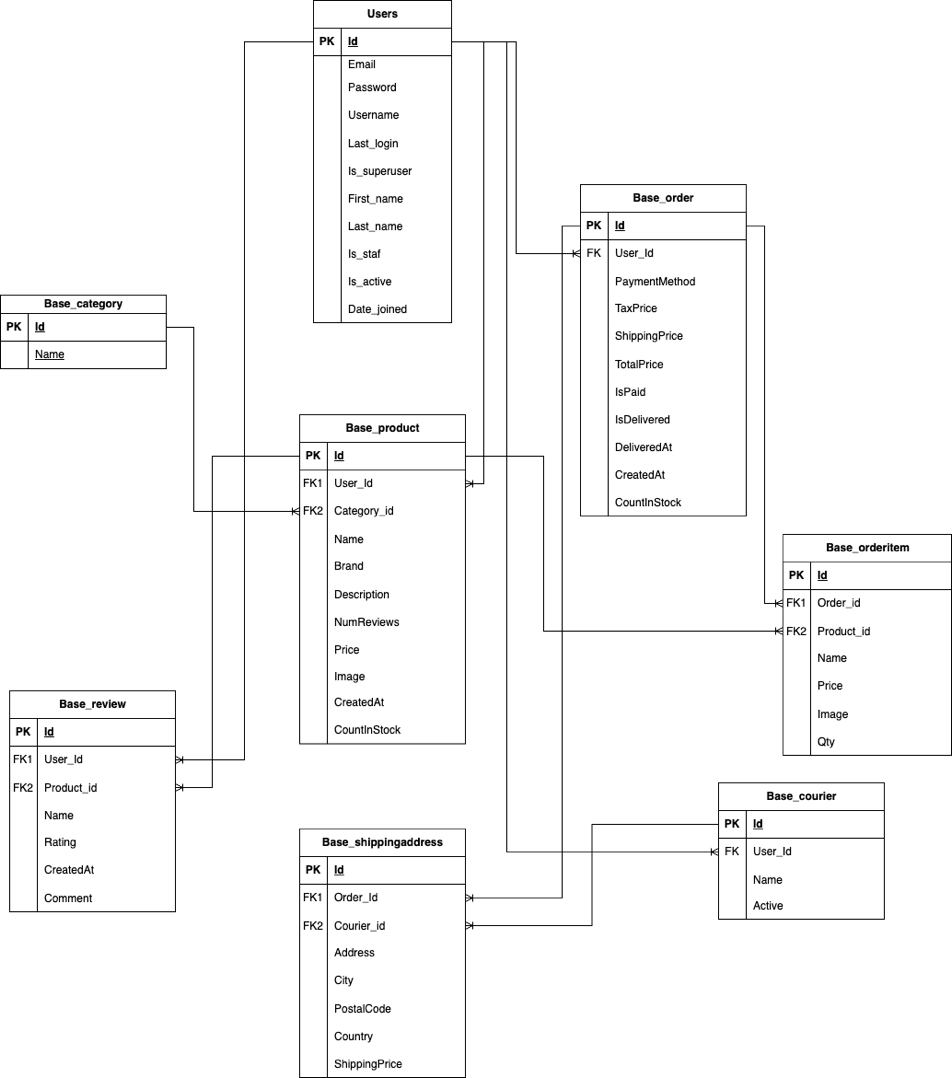
Необходимо провести тестирование производительности базы данных на больших объемах данных (не менее 100 000 строк) и внести корректировки в структуру базы данных при выявлении проблем с производительностью или масштабируемостью. Завершающим этапом является подготовка документации, описывающей процесс разработки и внедрения базы данных, а также предоставляющей рекомендации по дальнейшему совершенствованию проекта.

# Проектирование базы данных

Для начала создадим отдельную базу данных для курсового проекта. Определим таблицы, которые будут использоваться в проекте.

Далее определим необходимые таблицы для нашего онлайн-маркетплейса. Разнообразие данных, с которыми может работать крупный онлайн-маркетплейс, зачастую требует сотен таблиц. Это могут быть данные о клиентах, продавцах, товарах, заказах, отзывах, доставке, оплате и многих других аспектах бизнес-процессов. Однако, для наших учебных целей, мы ограничимся минимальным, но все еще репрезентативным набором таблиц. В нашем случае, база данных маркетплейса будет содержать таблицы для продуктов, пользователей, заказов, отзывов, курьеров и доставок, а также дополнительные таблицы для работы веб-приложения.

Предварительная диаграма основных таблиц и их отношения представлены на рисунках 2.1.



**Рисунок 2.1 – Основные таблицы базы данных**

Основные таблицы включают base\_Category, base\_Product, auth\_User, base\_Order, base\_Order\_Item, base\_Review, base\_ShippingAdress. Префиксы base и auth в названиях таблиц на рисунке 1 необходимы для корректной работы веб-приложения, поэтому в дальнейшем названия таблиц в пояснительной записке будем использовать без них.

Когда пользователь совершает покупку, создается запись в таблице Order, а соответствующие данные о товарах добавляются в таблицу Order\_Item. Такой подход позволяет объединить информацию о заказе и информацию о проданных товарах в одной структуре данных с минимальным количеством полей.

Таблица "Order" содержит основную информацию о заказах, включая время создания, способ оплаты и статус оплаты. Простая структура этой таблицы позволяет эффективно хранить и отслеживать информацию о платежах в контексте каждого заказа. Вся необходимая информация о платеже хранится в этой таблице, что обеспечивает компактность и эффективность базы данных при отслеживании заказов и информации о платежах.

Таблица "ShippingAddress" содержит информацию о доставке заказов, включая адрес доставки, метод доставки и статус доставки. Эта таблица позволяет хранить и отслеживать данные о доставке для каждого заказа, обеспечивая удобство и надежность процесса доставки.

Таблица "Review" предназначена для хранения информации об оценках товаров. Пользователи могут оставлять отзывы о товарах, и эта таблица позволяет сохранять эти отзывы в базе данных. Она обеспечивает возможность отслеживать и анализировать оценки товаров, что помогает пользователям принимать информированные решения при покупке товаров на маркетплейсе.

Таблица "User" содержит информацию о зарегистрированных пользователях маркетплейса, включая их имя пользователя, электронную почту и пароль. Эта таблица позволяет отслеживать активность пользователей, обеспечивать безопасность и аутентификацию, а также предоставлять различные функции и привилегии в зависимости от роли пользователя.

Таблица "Product" содержит информацию о товарах, доступных на маркетплейсе. Она включает данные о названии товара, описании, изображении, цене, количестве и продавце. Эта таблица играет ключевую роль в отображении и представлении товаров на платформе, а также обеспечивает функциональность поиска, фильтрации и сортировки товаров.

Таблица "Category" используется для хранения информации о категориях товаров на маркетплейсе. Она включает идентификатор категории и название категории. Эта таблица помогает пользователям навигироваться по маркетплейсу и быстро находить нужные товары, связывая категории с соответствующими товарами.

В целом, структура базы данных для онлайн-маркетплейса на основе PostgreSQL включает несколько таблиц, каждая из которых хранит и отслеживает определенную информацию о заказах, платежах, доставке, отзывах пользователей, пользователях, товарах и категориях. Это обеспечивает эффективность, надежность и функциональность базы данных, позволяя маркетплейсу эффективно управлять и предоставлять услуги своим клиентам.

Комбинированное использование этих таблиц обеспечивает полную и информативную базу данных для онлайн-маркетплейса, позволяющую отслеживать заказы, платежи, доставку, отзывы пользователей и другую важную информацию, необходимую для успешной операции маркетплейса.

Структура базы данных, разработанная на основе PostgreSQL с использованием технологии мониторинга состояния СУБД, позволяет не только эффективно управлять данными и обеспечивать безопасность, но и проводить анализ производительности и оптимизировать работу системы, гарантируя высокое качество обслуживания и удовлетворение потребностей клиентов.

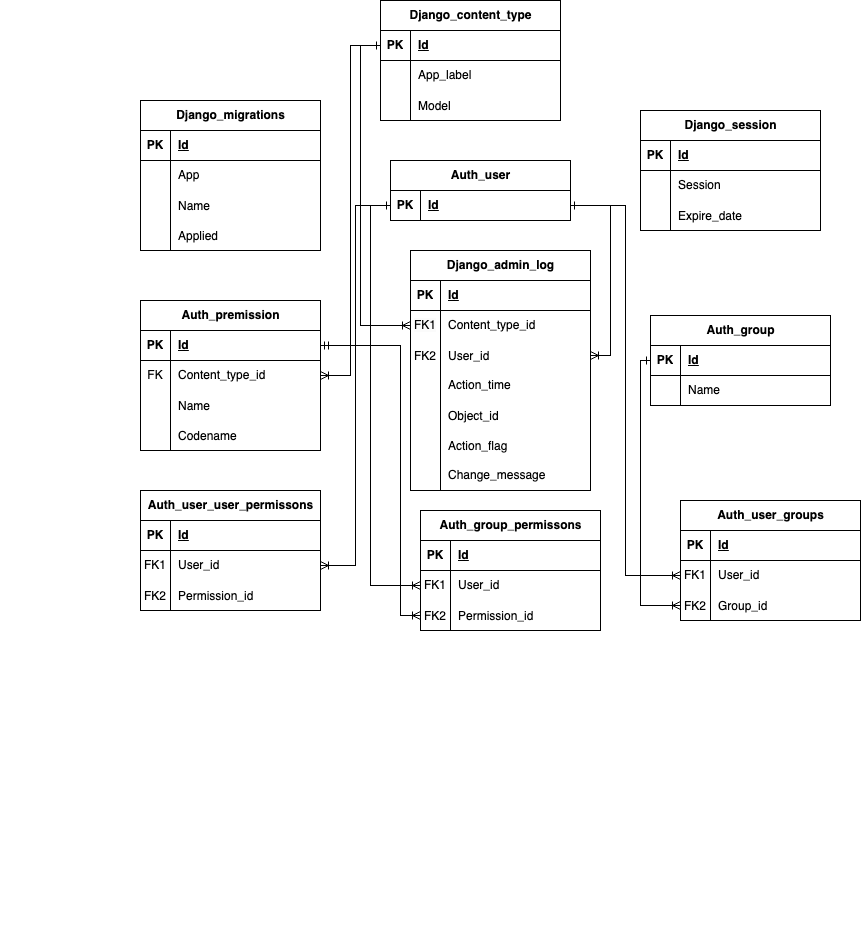


Рисунок 2.2 – Дополнительный таблицы базы **данных**

На рисунке отображены дополнительные таблицы базы данных, необходимы для работы веб-приложения Онлайн-маркеплейса: Django\_content\_type, Django\_migrations, Django\_session, auth\_permission, Django\_admin\_log, Auth\_user\_user\_permissions.

Таблица Django\_content\_type используется для хранения информации о типах контента в Django. Эта таблица содержит информацию о каждом из моделей приложения, включая имя приложения и имя модели. Система аутентификации Django использует эту таблицу для управления разрешениями и определения того, какие операции могут выполнять разные пользователи.

Таблица Django\_migrations отслеживает все миграции базы данных, которые были применены к проекту Django. Каждая миграция представляет собой набор изменений, примененных к структуре базы данных. Эта таблица помогает гарантировать, что все экземпляры проекта используют одну и ту же версию структуры базы данных.

Таблица Django\_session используется для хранения данных сессий пользователей. Когда пользователь входит в систему, Django создает новую сессию и сохраняет ее в этой таблице. Это позволяет Django отслеживать состояние каждого пользователя между различными запросами.

Таблица auth\_permission хранит информацию о разрешениях, связанных с различными моделями в Django. Разрешения используются для ограничения доступа пользователей к определенным частям приложения. Каждое разрешение связано с определенным типом контента, и Django использует эту таблицу для проверки, имеет ли пользователь необходимые разрешения для выполнения определенных действий.

Таблица Django\_admin\_log хранит информацию обо всех действиях, выполненных в административной панели Django. Каждая запись в этой таблице содержит информацию о том, какой пользователь выполнил действие, какое действие было выполнено, и когда оно было выполнено. Это обеспечивает учетность и позволяет администраторам отслеживать действия пользователей в системе.

Все эти таблицы являются важными составляющими для реализации основной функциональности онлайн-маркетплейса и обеспечения удобства использования платформы для пользователей и администраторов с помощью веб-приложения.

Для обеспечения гибкости управления доступом к базе данных созданы роли admin, customer, courier и guest. Это позволяет разграничивать доступ к данным и обеспечивать безопасность системы.

Взаимодействие с системой продемонстрировано с использованием подробной Use-case диаграммы, которая наглядно отражает различные группы пользователей и их возможные действия в рамках системы. На рисунке 2 представлены клиент, гость, администратор и курьер, а также их взаимодействие с функциями онлайн-маркетплейса.

Данная диаграмма помогает лучше понять, как пользователи взаимодействуют с системой и какие функциональные возможности им доступны. Каждая группа пользователей имеет свои уникальные действия и привилегии, которые они могут выполнять в системе.

Особенностью системы является использование хранимых процедур и триггеров для обеспечения целостности данных и автоматизации определенных процессов. Например, при создании заказа происходит автоматическое обновление количества товаров, а также расчет скидки на товар, если применимо. Такой подход повышает эффективность и надежность системы, минимизирует возможность ошибок и обеспечивает плавное выполнение задач.

Эта Use-case диаграмма представляет важную часть разработки и позволяет легче понять функциональные возможности системы и взаимодействие пользователей с ней. Она служит основой для дальнейшего проектирования и реализации системы, а также оптимизации процессов и обеспечения удобства использования для пользователей.

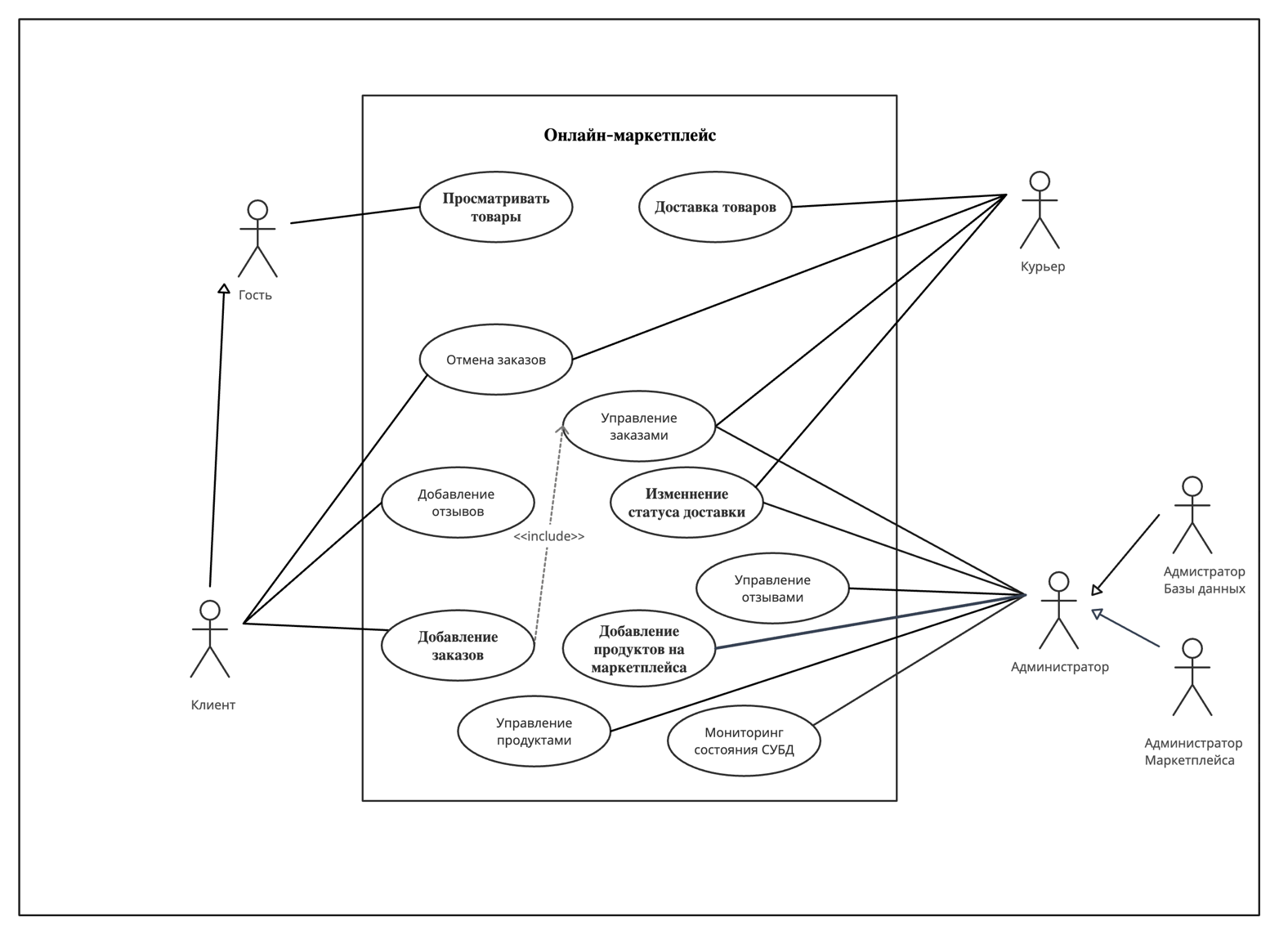


Рисунок 2 – Use-case диаграмма

Разработка индексов, а также триггеров и других объектов, необходимых для работы системы по задуманному сценарию, будет рассмотрена в следующем пункте пояснительной записки.

Этот этап разработки является важным для обеспечения эффективности и оптимизации работы системы. Создание индексов позволяет ускорить поиск и доступ к данным в базе данных, что особенно важно при больших объемах данных. Триггеры и другие объекты обеспечивают автоматическое выполнение определенных действий или проверки при определенных событиях или изменениях данных.

В процессе разработки будут определены и созданы необходимые индексы, чтобы улучшить производительность выполнения запросов и операций с данными. Также будут разработаны триггеры для автоматического выполнения определенных задач, например, обновление связанных данных или проверка прав доступа.

## Вывод проектирования базы данных

В результате проектирования базы данных для онлайн-маркетплейса были определены основные таблицы и их связи. Ключевые таблицы включают информацию о категориях товаров, самих товарах, пользователях, заказах, отзывах и доставке. Дополнительные таблицы использованы для работы веб-приложения и включают информацию о типах контента, миграциях базы данных, сессиях пользователей, разрешениях доступа и административных действиях.

Такое проектирование базы данных обеспечивает возможность хранения и отслеживания данных о продуктах, заказах, пользовательской активности и других аспектах функциональности маркетплейса. Отношения между таблицами позволяют эффективно организовать данные и обеспечить целостность информации.

База данных разработана с учетом минимального, но все еще репрезентативного, набора таблиц, чтобы упростить структуру базы данных и облегчить ее управление.

# Разработка объектов базы данных

## Роли и пользователи

Как уже описывалось выше, у нас будет 4 роли – администратор, клиент, гость и курьер. Каждая роль определена в соответствии с уровнем доступа и возможностями, которые она предоставляет пользователям.

Администратор в нашей системе обладает наивысшим уровнем доступа. Этот пользователь имеет полный контроль над всеми аспектами маркетплейса, включая управление товарами, обработку заказов и модерацию отзывов. Администратор также может управлять учетными записями пользователей и предоставлять или отнимать у них доступ к определенным функциям маркетплейса.

Клиент – это зарегистрированный пользователь, который осуществляет покупки на маркетплейсе. Этот пользователь имеет доступ к персональным функциям, таким как создание и управление своим списком желаний, просмотр истории заказов, оставление отзывов на товары и управление личными данными.

Гость – это незарегистрированный пользователь, который может просматривать товары и оставлять отзывы, но не может осуществлять покупки. Этот пользователь имеет ограниченный доступ к функциям маркетплейса и не может взаимодействовать с администратором или курьером. Однако гость может в любой момент зарегистрироваться и стать клиентом.

Курьер – это пользователь, ответственный за доставку заказов. Этот пользователь имеет доступ к информации о заказах, которые ему нужно доставить, и может общаться с клиентами и администратором по вопросам, связанным с доставкой. Курьер не имеет доступа к управлению товарами или модерации отзывов.

Такие различные роли помогают создать структурированную и безопасную среду для взаимодействия пользователей с нашим онлайн-маркетплейсом.

Ниже предоставлены листинг 3.1 – создания всех ролей и пользователей.

-- Создание роли администратора

CREATE ROLE admin\_role WITH LOGIN PASSWORD 'admin\_password';

GRANT ALL PRIVILEGES ON DATABASE marketplace TO admin\_role;

-- Создание роли клиента

CREATE ROLE client\_role WITH LOGIN PASSWORD 'client\_password';

GRANT SELECT ON base\_product, base\_review TO client\_role;

GRANT INSERT, UPDATE, DELETE ON base\_Order, base\_orderitem, base\_review TO client\_role;

-- Создание роли гостя

CREATE ROLE guest\_role WITH LOGIN PASSWORD 'guest\_password';

GRANT SELECT ON base\_product, base\_review TO guest\_role;

-- Создание роли курьера

CREATE ROLE courier\_role WITH LOGIN PASSWORD 'courier\_password';

GRANT SELECT, UPDATE ON base\_shippingaddress TO courier\_role;

Листинг 3.1 – Создание ролей и пользователей

Администратору предоставляются все привилегии для управления базой данных.

Клиенту предоставляются привилегии только для чтения данных из определенных таблиц и для вставки, обновления и удаления данных только из таблиц, связанных с его учетной записью.

Гость может только просматривать информацию о товарах и отзывах, но не может вносить изменения.

Курьеру предоставляются права на просмотр и обновление информации о доставке.

## Таблицы

Таблицы являются неотъемлемой частью любой реляционной базы данных. Краткая характеристика каждой из таблиц была предоставлена в разделе 2, а код их создания можно увидеть в Приложении А. Ниже мы рассмотрим каждую таблицу подробнее.

Таблица User хранит персональные данные пользователей состоит из одиннадцати столбцов, их структура продемонстрирована в таблице 3.1.

Таблица 3.1 – User

|  |  |  |
| --- | --- | --- |
| Название | Тип данных | Описание |
| id | INTEGER | Идентификатор пользователя |
| password | VARCHAR(128) | Хэш пароля пользователя |
| last\_login | TIMESTAMP WITH TIME ZONE | Время последнего входа |
| is\_superuser | BOOLEAN | Является ли суперпользователем |
| username | VARCHAR(150) | Имя пользователя |
| first\_name | VARCHAR(150) | Имя |
| last\_name | VARCHAR(150) | Фамилия |
| email | VARCHAR(254) | Электронная почта |
| is\_staff | BOOLEAN | Является ли сотрудником |
| is\_active | BOOLEAN | Активен ли аккаунт |
| date\_joined | TIMESTAMP WITH TIME ZONE | Дата регистрации |

Таблица Shipping\_Address, представленная в таблице 3.2, играет важную роль в системе управления нашего онлайн-маркетплейса, поскольку она содержит в себе ключевые данные, связанные с доставкой товаров.

Таблица 3.2 – ShippingAddress

|  |  |  |
| --- | --- | --- |
| Название | Тип данных | Описание |
| address | VARCHAR(200) | Адрес |
| city | VARCHAR(200) | Город |
| postalCode | VARCHAR(200) | Почтовый код |
| country | VARCHAR(200) | Страна |
| shippingPrice | NUMERIC(7,2) | Цена доставки |
| \_id | INTEGER | Идентификатор адреса доставки |
| order\_id | INTEGER | Идентификатор заказа |
| courier\_id | INTEGER | Идентификатор курьера |

Таблица Review, которая подробно описана в таблице 3.3, играет критическую роль в рамках нашего онлайн-маркетплейса, поскольку она хранит отзывы пользователей о продуктах. Эти данные предоставляют ценный обратный связной канал от наших клиентов, позволяя улучшать качество предлагаемых товаров и услуг.

Таблица 3.3 – Review

|  |  |  |
| --- | --- | --- |
| Название | Тип данных | Описание |
| name | VARCHAR(200) | Имя пользователя оставившего отзыв |
| rating | INTEGER | Рейтинг продукта |
| comment | TEXT | Комментарий к отзыву |
| \_id | INTEGER | Идентификатор отзыва |
| product\_id | INTEGER | Идентификатор продукта |
| user\_id | INTEGER | Идентификатор пользователя |
| createdAt | TIMESTAMP WITH TIME ZONE | Время создания отзыва |

Таблица Product, подробно описанная в таблице 3.4, занимает центральное место в структуре нашего онлайн-маркетплейса, поскольку она содержит информацию обо всех товарах, предлагаемых на платформе. Эти данные необходимы для обеспечения корректного отображения товаров и обработки заказов.

Таблица 3.4 – Product

|  |  |  |
| --- | --- | --- |
| Название | Тип данных | Описание |
| name | VARCHAR(200) | Название продукта |
| brand | VARCHAR(200) | Бренд продукта |
| category\_id | INTEGER | Идентификатор категории |
| description | TEXT | Описание продукта |
| rating | NUMERIC(7,2) | Рейтинг продукта |
| numReviews | INTEGER | Количество отзывов |
| price | NUMERIC(7,2) | Цена прод |
| countInStock | INTEGER | Количество на складе |
| createdAt | TIMESTAMP WITH TIME ZONE | Время создания продукта |
| id | INTEGER | Идентификатор продукта |
| user\_id | INTEGER | Идентификатор пользователя |
| image | VARCHAR(100) | Изображение продукта |

Таблица OrderItem, представленная в таблице 3.5, играет ключевую роль в системе управления нашего онлайн-маркетплейса, поскольку она хранит данные о каждом товаре в заказе. Без этой информации невозможно было бы корректно обработать и выполнить заказы клиентов.

Таблица 3.5 – OrderItem

|  |  |  |
| --- | --- | --- |
| Название | Тип данных | Описание |
| name | VARCHAR(200) | Название товара |
| qty | INTEGER | Количество товара |
| price | NUMERIC(7,2) | Цена товара |
| image | VARCHAR(200) | Изображение товара |
| id | INTEGER | Идентификатор товара в заказе |
| order\_id | INTEGER | Идентификатор заказа |
| product\_id | INTEGER | Идентификатор продукта |

Таблица Order, которую мы подробно рассмотрим далее в таблице 3.6, имеет важное значение для функционирования нашего онлайн-маркетплейса, так как в ней собрана вся информация о заказах. Это обеспечивает эффективное управление процессом заказа и доставки товаров.

Таблица 3.6 – Order

|  |  |  |
| --- | --- | --- |
| Название | Тип данных | Описание |
| paymentMethod | VARCHAR(200) | Метод оплаты |
| taxPrice | NUMERIC(7,2) | Налог на цену |
| shippingPrice | NUMERIC(7,2) | Цена доставки |
| totalPrice | NUMERIC(7,2) | Общая цена |
| isPaid | BOOLEAN | Оплачен ли заказ |
| paidAt | TIMESTAMP WITH TIME ZONE | Время оплаты |
| isDelivered | BOOLEAN | Доставлен ли заказ |
| deliveredAt | TIMESTAMP WITH TIME ZONE | Время доставки |
| id | INTEGER | Идентификатор заказа |
| user\_id | INTEGER | Идентификатор пользователя |
| createdAt | TIMESTAMP WITH TIME ZONE | Время создания заказа |
| paymentMethod | VARCHAR(200) | Метод оплаты |

Таблица Courier, описанная в таблице 3.7, играет важную роль в системе управления нашего онлайн-маркетплейса, поскольку она содержит в себе данные о курьерах, ответственных за доставку товаров. Без этих данных невозможно было бы организовать надлежащую доставку.

Таблица 3.7 – Courier

|  |  |  |
| --- | --- | --- |
| Название | Тип данных | Описание |
| id | INTEGER | Идентификатор курьера |
| name | VARCHAR(200) | Имя курьера |
| active | BOOLEAN | Активен ли курьер |
| user\_id | INTEGER | Идентификатор пользователя |

Таблица Category, подробно представленная в таблице 3.8 , играет существенную роль в нашем онлайн-маркетплейсе, так как она содержит информацию о категориях товаров. Это позволяет клиентам удобно навигировать по ассортименту и выбирать товары в соответствии с их потребностями.

Таблица 3.8 – Category

|  |  |  |
| --- | --- | --- |
| Название | Тип данных | Описание |
| id | INTEGER | Идентификатор категории |
| name | VARCHAR(200) | Название категории |

После создания таблиц они были заполнены небольшим количеством тестовых данных для проверки работоспособности и разработки объектов следующих пунктов.

## Триггеры

В рамках данной базы данных было создано пять триггеров.

Первый триггер – BEFORE UPDATE на таблице Order, он отвечает за автоматическое обновление поля isPaid до TRUE, когда статус заказа будет изменен на 'Paid'.

Второй триггер – BEFORE INSERT на таблице Order, он проверяет наличие товара на складе перед созданием заказа. Если товара нет в наличии, то заказ не будет создан.

Третий триггер – AFTER INSERT на таблице Order, он автоматически уменьшает количество товара на складе на единицу при создании нового заказа.

Четвертый триггер – BEFORE UPDATE на таблице Order, он отвечает за автоматическое оповещение админа, если время из поля deliveryTime больше текущего, а статус заказа еще не ‘Delivered’.

Пятый триггер – BEFORE UPDATE на таблице Order, он отвечает за автоматическое обновление поля isDelivered до TRUE, когда статус заказа будет изменен на 'Delivered'.

Более подробно рассмотрим последний триггер и процесс его создания. Внутри него мы используем функцию update\_delivery\_status, которая проверяет, был ли статус заказа изменен на 'Delivered'. Если да, то поле isDelivered обновляется до TRUE. Это обеспечивает автоматическое обновление поля isDelivered при изменении статуса заказа, что упрощает управление статусами доставок и заказов.

Создание функции для триггера: Первым шагом является создание функции, которая будет вызываться триггером при каждом обновлении статуса доставки в таблице Order. Функция проверяет, если новый статус равен 'Delivered', то поле isDelivered обновляется до TRUE. Создание функции продемонстрированно в листинге 3.2.

-- Создание роли функции обновления заказа

CREATE OR REPLACE FUNCTION update\_delivery\_status()

RETURNS TRIGGER AS $$

BEGIN

-- сравниваем статус доставки

IF NEW.status = 'Delivered' THEN

NEW."isDelivered" := TRUE;

END IF;

RETURN NEW;

END;

$$ LANGUAGE plpgsql;

Листинг 3.2 – Update\_delivery\_status функция для триггера

Создание триггера: Теперь создадим сам триггер, который будет вызывать эту функцию при каждом обновлении статуса доставки. Код создания триггера отображен в листинге 3.3.

CREATE TRIGGER delivery\_status\_trigger

BEFORE UPDATE ON public."base\_order"

FOR EACH ROW

WHEN (NEW.status <> OLD.status)

EXECUTE FUNCTION update\_delivery\_status();

Листинг 3.3 – Триггер обновления статуса доставки

Тестирование триггера: Для демонстрации работы триггера, мы добавляем новый заказ, обновляем его статус и проверяем, что поле isDelivered обновляется в соответствии с новым статусом. Все этапы продемонстрированы соответственно на рисунке 3.1, листинге 4 и рисунке 3.2.

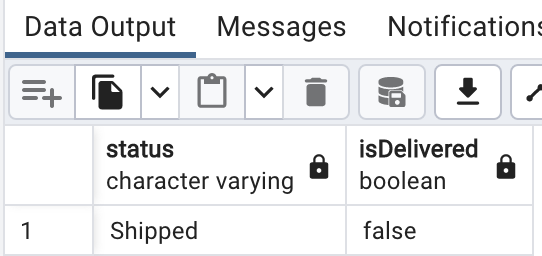


Рисунок 3.1 – Статус заказа и поле isDelivered

-- Устанавливаем роль админа

SET ROLE admin\_role;

-- Добавляем новый заказ

CALL public.add\_order('TestDeliveryMethod', 10.00, 5.00, 115.00, 1);

-- Проверяем, что статус заказа установлен в 'Shipped'

SELECT "status" FROM public."base\_order" WHERE "DeliveryMethod" = 'TestDeliveryMethod';

-- Обновляем статус заказа на 'Delivered'

UPDATE public."base\_order" SET "status" = 'Delivered' WHERE "DeliveryMetod" = 'TestDeliveryMethod';

Листинг 3.4 – Моделирование события доставки

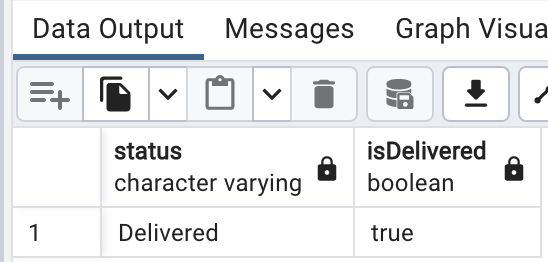


Рисунок 3.2 – Результат выполнения триггера

Полный скрипт на создание триггеров представлен в приложении Б.

## Процедуры

### Процедуры администратора

Для администратора будет реализовано 10 процедур: добавление категории, добавление товара, добавление курьера, удаление товара, просмотра пользователей, просмотр курьеров, просмотра товаров, просмотр всех заказов, удаление пользователей, удаление товаров. Рассмотрим процедуру создания новой категории – Листинг 3.5.

-- Процедура добавления категории

CREATE OR REPLACE PROCEDURE public.add\_category(p\_name VARCHAR)

LANGUAGE plpgsql

AS $$

BEGIN

-- вставляем данные категории

INSERT INTO public.base\_category(name) VALUES(p\_name);

END;

$$;

-- Отзываем права на процедуру у всех

REVOKE ALL ON PROCEDURE public.add\_category(VARCHAR) FROM PUBLIC;

-- Даем право на процедуру админу

GRANT EXECUTE ON PROCEDURE public.add\_category(VARCHAR) TO admin;

-- Вызов процедуры создание тестовой категории

CALL public.add\_category('TestCategory');

Листинг 3.5 – Процедура для создания новой категории

На рисунке 3.1 видим, что после вызова процедуры, категория успешно добавлена.

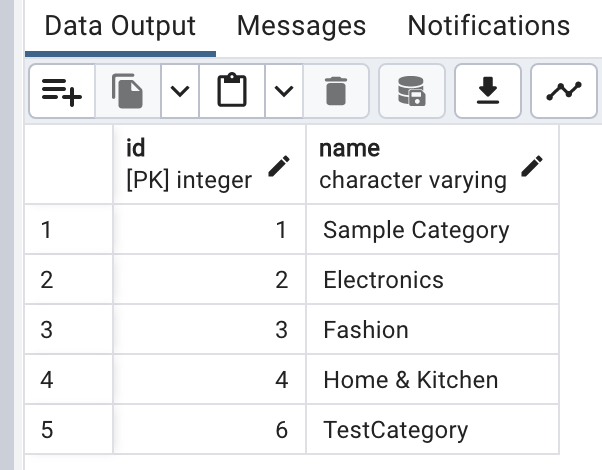


Рисунок 3.3 – Тестовая категория добавлена

Таже рассмотри детально процедуру просмотра пользователей, скрипт создания которой отображен в листинге 3.6.

CREATE OR REPLACE PROCEDURE public.view\_users(out ref REFCURSOR)

LANGUAGE plpgsql

AS $$

BEGIN

OPEN ref FOR SELECT id, username, first\_name, last\_name, email FROM public."auth\_user";

END;

$$;

REVOKE ALL ON PROCEDURE public.view\_users(REFCURSOR) FROM PUBLIC;

GRANT EXECUTE ON PROCEDURE public.view\_users(REFCURSOR) TO admin\_role;

BEGIN;

CALL public.view\_users(@ref);

FETCH ALL FROM @ref;

COMMIT;

Листинг 3.6 – Процедура просмотра всех пользователей

У нас есть процедура view\_users, которая не принимает входных параметров. Она возвращает один выходной параметр типа REFCURSOR. Необходимость в его использовании заключается в том, что в PostgreSQL нельзя просто обернуть SELECT запрос в процедуру – данные обязательно должны куда-то записываться. В данном случае, процедура view\_users открывает курсор, который содержит результаты запроса SELECT, выбирающего данные пользователей. Эти данные возвращаются как REFCURSOR.

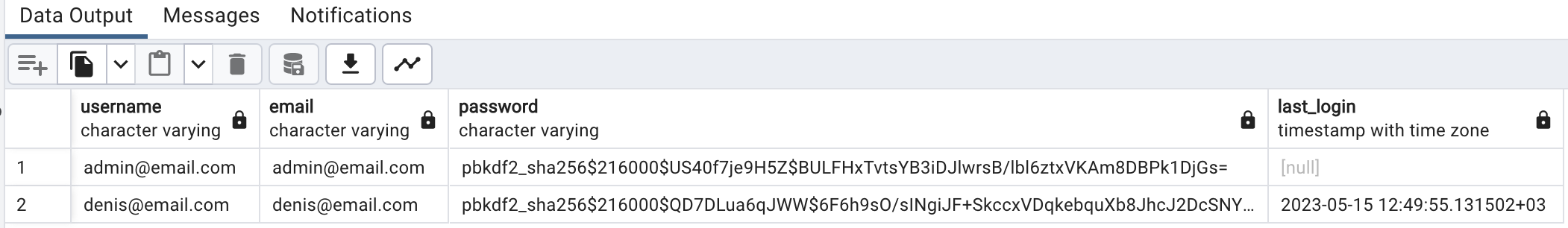


Рисунок 3.4 – Результат выполнения процедуры

### Процедуры клиента

Для клиента онлайн-маркетплейса будут реализованы шесть процедур: просмотр товаров, добавление товара в заказ, просмотр заказов, добавление отзыва к товару, оценка качества доставки и обновление информации об адресе доставки. Рассмотрим процедуру просмотра товаров – Листинг 3.7.l

-- Процедура добавления отзыва к товару

CREATE OR REPLACE PROCEDURE public.add\_review(

p\_name VARCHAR,

p\_id INTEGER,

p\_createdAt TIMESTAMP WITH TIME ZONE,

p\_comment VARCHAR,

p\_rating INTEGER,

p\_user\_id INTEGER,

p\_product\_id INTEGER

)

LANGUAGE plpgsql

AS $$

BEGIN

INSERT INTO public."base\_review"(name,\_id,"createdAt","comment", "rating", "user\_id", "product\_id")

VALUES(p\_name, p\_id, p\_createdAt, p\_comment, p\_rating, p\_user\_id, p\_product\_id);

END;

$$;

Листинг 3.7 – Процедура добавления отзыва

Для процедуры добавления отзыва на товар, мы принимаем следующие входные параметры: имя отзыва, идентификатор отзыва, текст комментария, рейтинг отзыва, идентификатор пользователя и идентификатор продукта.

Также устанавливаем время создания отзыва в поле createdAt с помощью функции CURRENT\_TIMESTAMP. Результат этого действия – новый отзыв на товар добавляется в базу данных. Вызов этой процедуры представлен в листинге 3.8.

-- Вызов процедуры добавления отзыва к товару

CALL public.add\_review('Test Review', 1, CURRENT\_TIMESTAMP, 'Great product!', 5, 4, 5);

Листинг 3.8 – Вызов процедуры добавления отзыва

Как видно на рисунке 3.5 видно, что отзыв успешно добавлен.

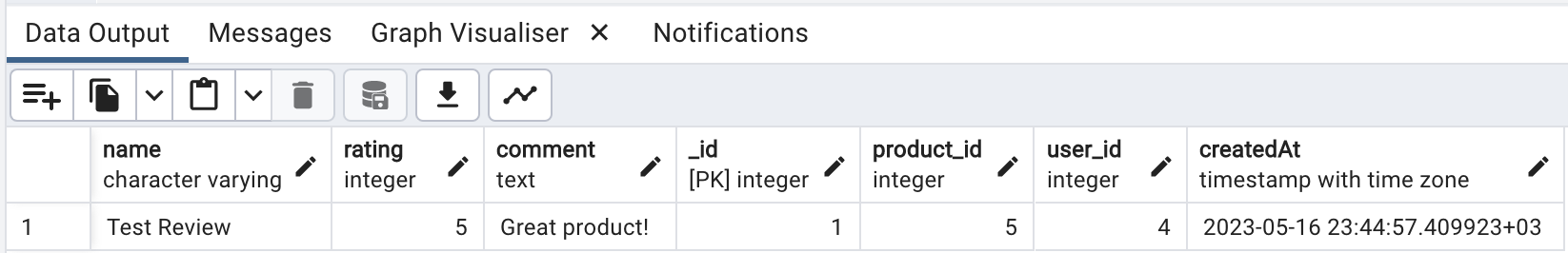


Рисунок 3.5 – Успешное добавление отзыва

### Процедуры гостя

Для гостя онлайн-маркетплейса будут реализованы две процедуры: просмотр товаров и просмотр отзывов о товаре. Эти процедуры аналогичны некоторым процедурам, которые были реализованы для клиента, но адаптированы для уникальных потребностей гостя.

Полный скрипт для создания вышеописанных хранимых процедур можно найти в приложении В.

### Процедуры курьера

Для курьера онлайн-маркетплейса будут реализованы четыре процедуры: просмотр заказов, обновление статуса доставки, просмотр подробностей заказа и регистрация курьера. Эти процедуры аналогичны некоторым процедурам, которые были реализованы для клиента, но адаптированы для уникальных потребностей курьера.

Полный скрипт для создания вышеописанных хранимых процедур можно найти в приложении В.

## Функции

В рамках курсового проекта возникла необходимость разграничить использование функций и процедур. Отличия между ними заключаются в сфере применения и используемых механизмах. Процедуры применяются для выполнения сложного набора операций, часто включающих управление транзакциями, в то время как функции, как правило, возвращают значение и не могут управлять транзакциями.

Были разработаны несколько функций, описанных в Приложении Г, для создания триггеров, а также функция регистрации курьера. В контексте приложения это действие представляет собой создание новой записи и возврат результата операции. Решение использовать функцию было обусловлено ее способностью возвращать значение, что оказалось полезным для обработки и обратной связи результата регистрации.

Отмечу, что было принято решение не использовать функцию регистрации курьера в текущем приложении. В процессе разработки стало ясно, что для удобства работы курьеров лучше разработать мобильное приложение. Это позволит им легко получать информацию о заказах и адресах доставки, а также управлять своим аккаунтом во время пути. Поэтому было решено отложить реализацию функционала курьера до создания мобильного приложения в будущем.

Создания функции представлено в листинге 3.9.

-- Создание функцкии регистрации курьера

CREATE OR REPLACE FUNCTION register\_courier(

courier\_name VARCHAR,

user\_name VARCHAR,

first\_name VARCHAR,

last\_name VARCHAR,

user\_password VARCHAR,

user\_email VARCHAR

) RETURNS TEXT AS $$

DECLARE

new\_courier\_id INTEGER;

BEGIN

-- создаем пользователя для курьера

INSERT INTO auth\_user (username,first\_name,last\_name, password, email, is\_staff, is\_active, is\_superuser, date\_joined)

VALUES (user\_name,first\_name,last\_name, user\_password, user\_email, FALSE, TRUE, FALSE, NOW())

RETURNING id INTO new\_courier\_id;

-- создаем курьера

INSERT INTO base\_courier (user\_id, name, active)

VALUES (new\_courier\_id, courier\_name, TRUE);

-- выводим сообщение об успешной регистрации

RETURN 'Courier successfully registered with ID ' || new\_courier\_id;

EXCEPTION WHEN unique\_violation THEN

-- либо о неуспешной

RETURN 'Registration failed. Courier with this name already exists.';

END;

$$ LANGUAGE plpgsql;

Листинг 3.9 – Функция регистрации курьера

Теперь зарегистрируем курьера – Листинг 3.10.

select register\_courier('CourierName','Petkin','palich', 'Username', 'Password', 'email@example.com');

Листинг 3.10 – Регистрация курьера

На рисунке 3.6 видно сообщение об успешном добавлении курьера.

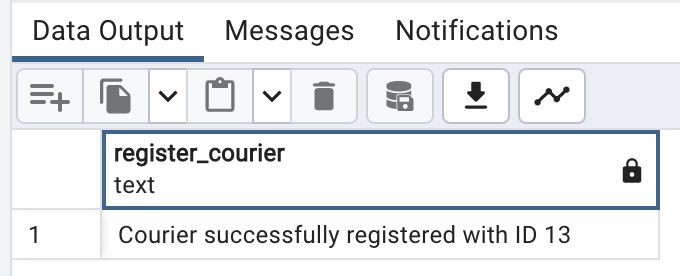


Рисунок 3.6 – Успешное добавление курьера

## Индексы

Индексация является ключевым инструментом оптимизации баз данных. PostgreSQL автоматически создает индексы для столбцов, определенных как PRIMARY KEY или UNIQUE, что обеспечивает высокую скорость операций поиска, обновления или удаления записей по этим столбцам. Тем не менее, в зависимости от частоты запросов к определенным столбцам, создание дополнительных индексов вручную может быть выгодно для улучшения производительности.

В данном проекте были создано несколько индексов, таких как:

* Auth\_index на поле username в таблице auth\_user для ускорения поиска пользователей по имени пользователя.
* Order\_status\_index на поле status в таблице Order для ускорения поиска заказов по статусу.
* Courier\_active\_index на поле active в таблице Courier для быстрого выявления активных курьеров.
* Order\_user\_index на поле user\_id в таблице Order для ускорения поиска заказов конкретного пользователя.
* Order\_user\_index на поле user\_id в таблице Order для ускорения поиска заказов конкретного пользователя.

Они помогут ускорить поиск по определенным полям в соответствующих таблицах.

Следует отметить, что основной целью создания этих индексов было улучшение скорости поиска данных. Однако, добавление индексов также увеличивает время выполнения операций вставки, обновления или удаления данных, так как индексы также нужно обновлять.

В разделе 5 данного проекта будет проведено детальное рассмотрение результатов и эффективности индексации на примере таблицы Order.

Реализация остальных индексов продемонстрирована в приложении Д.

## Последовательности

Для автоматического инкрементирования числовых идентификаторов в таблицах можно использовать последовательности. Для этого мы создадим последовательности для каждой из наших таблиц.

# Импорт и экспорт в JSON

# 4.1 Импорт данных таблицы в формате JSON

JSON (JavaScript Object Notation) – это стандартный текстовый формат для представления структурированных данных на основе синтаксиса объекта JavaScript. JSON является одним из наиболее популярных форматов для хранения данных из – за его широкого применения в веб-разработке, простоте в использовании, гибкости и возможности чтения человеком.

В листинге 4.1 представлена процедура, выполняющая вставку данных в таблицу Category из формата JSON.

CREATE OR REPLACE PROCEDURE import\_to\_json(path\_to\_file text)

LANGUAGE plpgsql

AS $$

BEGIN

CREATE TEMP TABLE temp (data jsonb);

EXECUTE format('COPY temp (data) FROM %L', path\_to\_file);

INSERT INTO "base\_category" ("id", "name")

SELECT (data->>'id')::integer,data->>'name'

FROM temp;

DROP TABLE temp;

END;

$$;

Листинtг 4.1 – Процедура, выполняющая импорт данных в формате JSON

Чтобы сохранить данные JSON документа в таблице в PostgreSQL можно использовать тип данных JSON или JSONB. JSONB хранит документ в бинарном формате и поддерживает индексирование, JSON же хранит данные в текстовом виде и не поддерживает индексирование. По скольку поля в типе данных JSON и JSONB все имеют тип данных text, их нужно явно привести к изначальным типам данных столбцов таблицы при импорте.

## 4.2 Экспорт данных таблицы в формате JSON

Для выполнения экспорта данных из формата JSON мы можем использовать различные инструменты и библиотеки, предоставляемые языками программирования и базами данных. Эти инструменты позволяют нам считывать данные из базы данных или других источников, и сохранять их в формате JSON. Пример файла в формате JSON, который будет экспортирован, представлен на рисунке 4.1.



Рисунок 4.1 – Файл JSON формата

В листинге 4.2 представлена процедура, выполняющая экспорт данных из JSON формата в таблицу Category. PostgreSQL предоставляет встроенный функционал для импорта данных в JSON через функцию COPY.

CREATE OR REPLACE PROCEDURE export\_json(path\_to\_file text)

LANGUAGE plpgsql

AS $$

BEGIN

EXECUTE format('COPY (SELECT row\_to\_json(t) FROM (SELECT \* FROM "base\_category") t) TO %L', path\_to\_file);

END;

Листинг 4.2 – Процедура, выполняющая экспорт данных в формате JSON

На рисунке 4.2 видно, что категории из файла успешно добавлены.

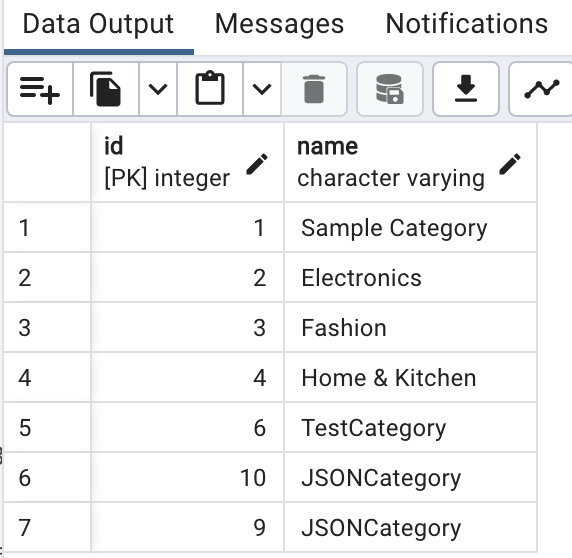


Рисунок 4.2 – Добавленные категории

Такой функционал очень полезен в рамках онлайн-маркетплейса, так как можно высылать данные аналитикам в удобном формате.

# Тестирование производительности

Перед началом тестирования была разработана функция для заполнения 100.000 строками тестовых данных в таблицу Order.

Функция представлена в листинге 5.1.

CREATE OR REPLACE FUNCTION insert\_random\_orders() RETURNS VOID AS $$

DECLARE

counter INTEGER := 0;

user\_count INTEGER;

random\_user\_id INTEGER;

random\_status VARCHAR;

statuses VARCHAR[] := ARRAY['Processing', 'Shipped', 'Delivered'];

BEGIN

SELECT COUNT(\*) INTO user\_count FROM auth\_user;

WHILE counter < 100000 LOOP

SELECT floor(random() \* user\_count) + 1 INTO random\_user\_id;

random\_status := statuses[floor(random() \* array\_length(statuses, 1)) + 1];

INSERT INTO "base\_order" (

user\_id,

"paymentMethod",

"deliveryTime",

"taxPrice",

"shippingPrice",

"totalPrice",

status,

"isPaid",

"paidAt",

"isDelivered",

"deliveredAt",

"createdAt"

)

VALUES (

(SELECT id FROM auth\_user LIMIT 1 OFFSET random\_user\_id),

'Credit Card',

NOW() + interval '1 day' \* counter,

10.00,

5.00,

100.00,

random\_status,

TRUE,

NOW(),

FALSE,

NULL,

NOW()

);

counter := counter + 1;

END LOOP;

END;

$$ LANGUAGE plpgsql;

Листинг 5.1 – Функция заполнения таблицы Order данными

## Тестирование базы данных

Для тестирования производительности базы данных необходимо заполнить таблицы достаточным количеством записей. Для данного случая мы заполнили таблицу 100.000 строк.

Для улучшения производительности запросов, было создано несколько индексов. Для таблицы Order мы создали индексы на столбцы "Status", "User\_id" и "CreatedAt".

Далее подробно рассмотрим, как добавление индекса на столбец "CreatedAt" повлияло на стоимость выполнения запросов к таблице Order. Пример создания индекс показан в листинге 5.2

|  |
| --- |
| -- Создание индекса  CREATE INDEX IF NOT EXISTS idx\_order\_created\_at ON public.base\_order (createdAt); |

Листинг 5.2 – Индекс для таблицы

Так как в онлайн-маркетплейсе часто необходимо отсортировать заказы по дате создания, сравним стоимость следующего запроса – листинг 5.3.

EXPLAIN SELECT \* FROM public.base\_order ORDER BY "createdAt" DESC;

Листинг 5.3 – Сортировка заказов по дате создания

При выполнении запроса, представленного в листинге 5.3, был использован инструмент Explain в pgAdmin 4 для анализа его стоимости и оптимизации. Результаты анализа, отображенные на Рисунке 5.1, показали, что стоимость выполнения запроса составляет приблизительно 15 227 условных единиц.

Анализ стоимости запроса с помощью инструмента Explain является важной частью процесса оптимизации базы данных. Он позволяет оценить затраты на выполнение запроса и идентифицировать узкие места или неэффективные операции, которые могут замедлять выполнение запроса.

Используя информацию, полученную из Explain, администратор базы данных может принять меры по оптимизации запроса, например, изменить структуру индексов, переписать запрос или внести другие изменения, чтобы улучшить его производительность. Это помогает снизить стоимость выполнения запроса, ускорить его выполнение и повысить общую эффективность базы данных.

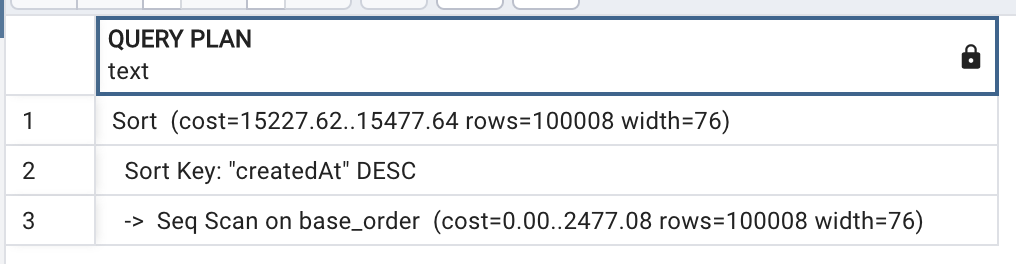


Рисунок 5.1 – Оценка запроса без использования индекса

Создание индекса на столбце "CreatedAt" позволило сократить стоимость запроса до 2086 условных единиц, что почти в 7,5 раза меньше. Результат выполнения запроса с использованием индекса отображен на рисунке 5.2.

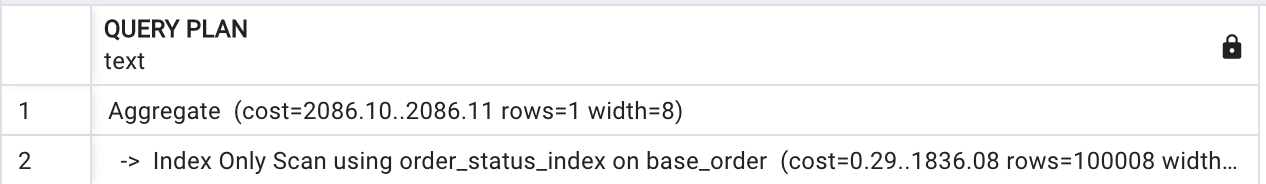


Рисунок 5.2 – Оценка запроса с использованием индексf

Таким образом, использование индексов является важным приемом оптимизации производительности базы данных и может сократить время выполнения запросов на выборку данных из таблиц.

## Тестирования приложения

После добавления 100 000 записей в таблицу "orders" возникли проблемы с производительностью приложения, связанные с большим объемом данных. В связи с этим было принято решение внедрить пагинацию для просмотра заказов, что позволило улучшить производительность системы и обеспечить более эффективное использование ресурсов.

На Рисунке 5.3 показан пример реализации пагинации в приложении. Пагинация позволяет разбить большой объем данных на небольшие части, называемые страницами, и отображать только одну страницу данных на экране пользователю. Это снижает нагрузку на систему и улучшает отзывчивость приложения при работе с большими объемами данных.

Добавление пагинации позволяет пользователям просматривать и навигировать по заказам постранично, а также сокращает время отклика приложения. Это удобное решение, которое обеспечивает более плавную и эффективную работу с большим объемом данных.

В результате внедрения пагинации в приложение удалось улучшить его производительность и обеспечить более комфортное пользовательское взаимодействие с заказами. Пагинация является одной из распространенных техник оптимизации работы с большими объемами данных и позволяет эффективно управлять и отображать данные на основе пользовательских запросов.

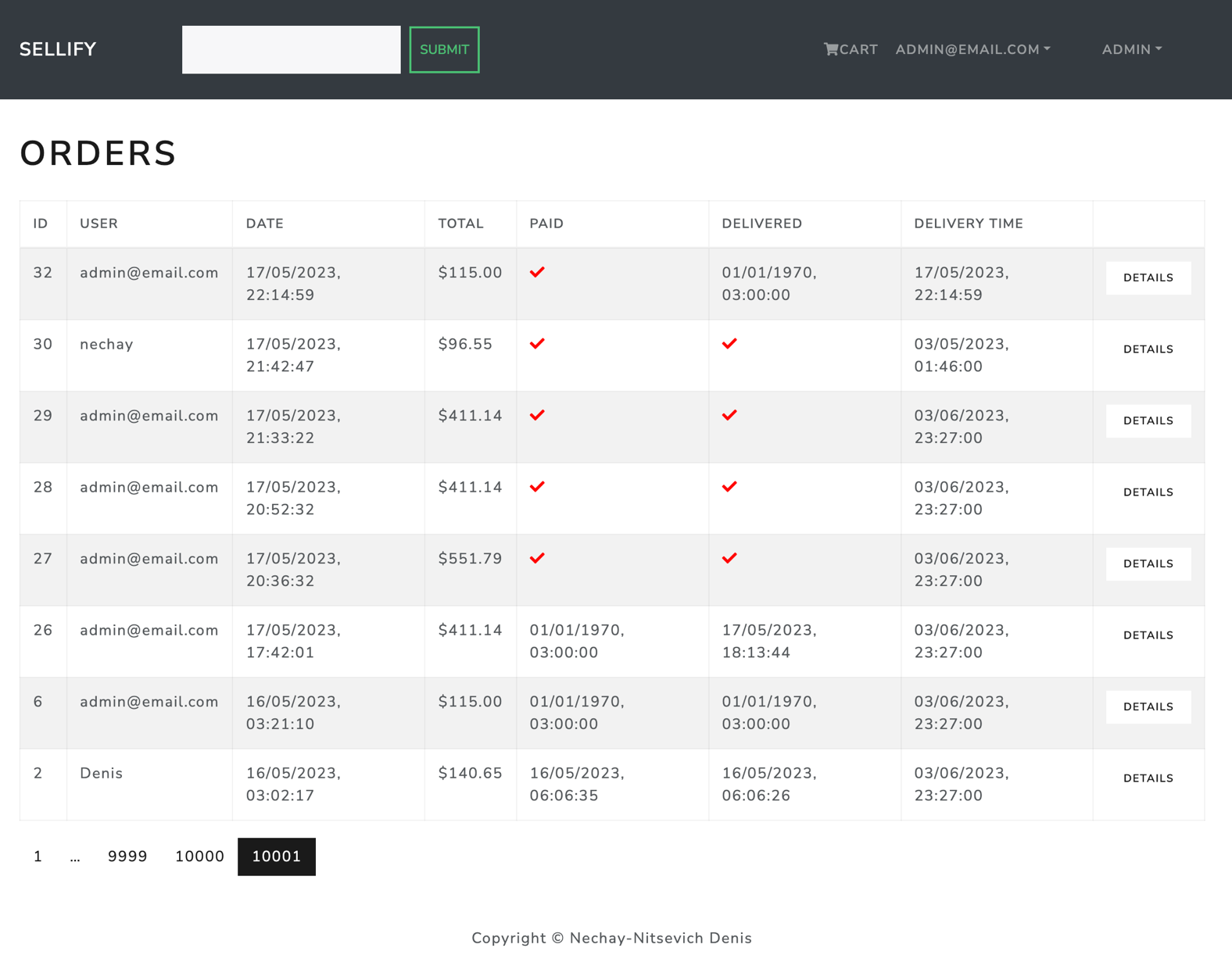


Рисунок 5.3 – Пагинатор страницы заказов

А также приложение теперь имеет возможность выводить заказы в отсортированном по дате создания виде. Это улучшение достигнуто за счет использования построенного индекса, описанного в разделе 5.1. Таким образом, производительность работы приложения была ускорена благодаря оптимизации запросов и использованию индекса на столбце "CreatedAt".

# Описание технологии и ее применения в базе данных.

## Использование pgAdmin 4 в качестве инструмента мониторинга СУБД

PgAdmin 4 представляет собой мощный интерфейс для управления и мониторинга баз данных PostgreSQL. Он предоставляет широкий спектр функциональных возможностей, позволяющих администраторам и разработчикам контролировать и анализировать работу базы данных. Одной из основных функций pgAdmin 4 является мониторинг активности базы данных, который обеспечивает ценную информацию о производительности, нагрузке и состоянии системы продемонстрированный на рисунке 6.1.

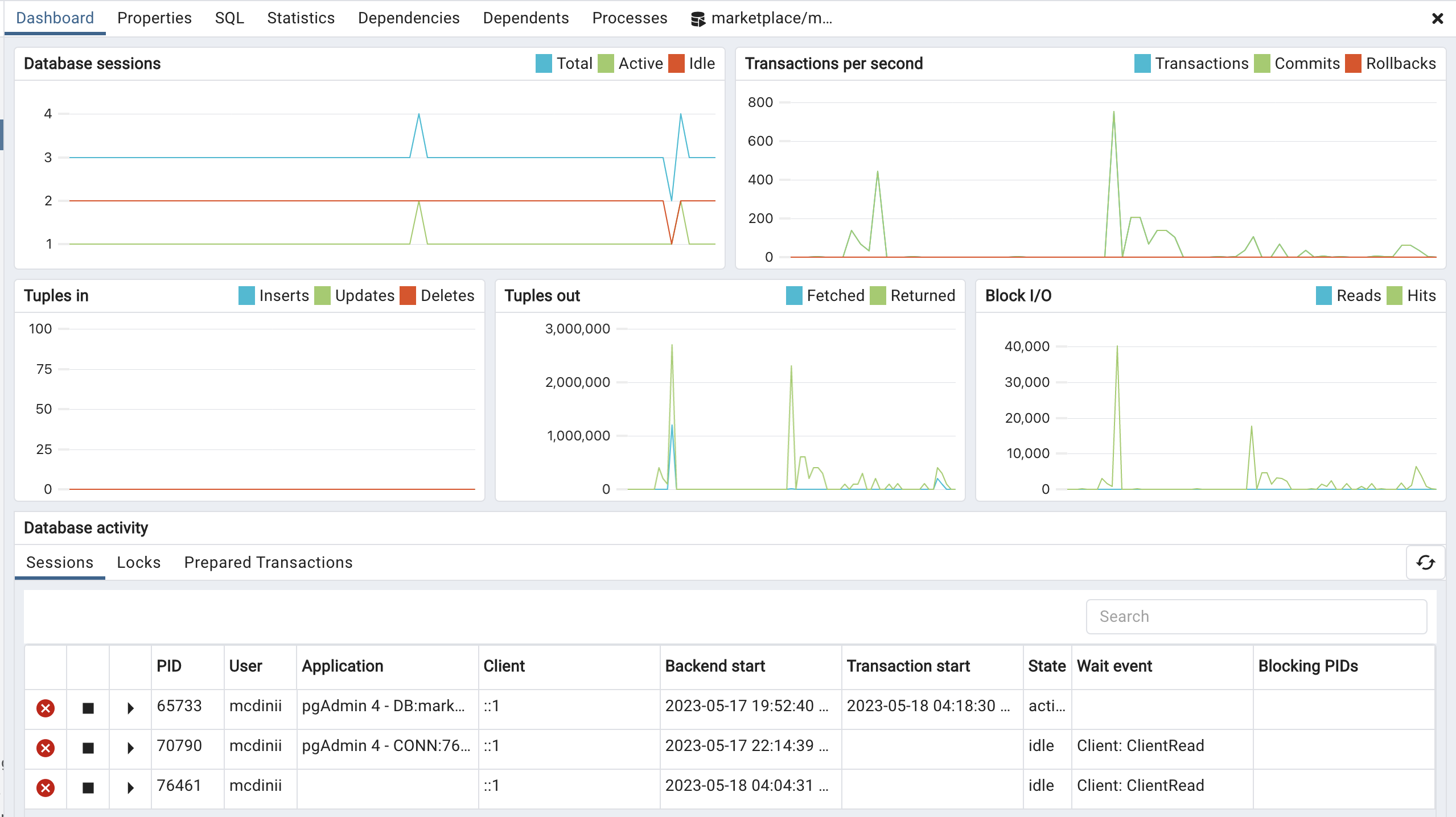


Рисунок 6.1 – Дашборд активности сервера базы данных

Такая информация будет очень полезна администратору онлайн-маркетплейса, так как он сможет контролировать поведение базы данных и следить за ее работой в пиковые моменты нагрузки.

Далее мы рассмотрим секции "Дашборд".

## Транзакции в секунду (Transactions per Second)

Секция "Транзакции в секунду" отображает количество транзакций, выполняемых в базе данных в течение определенного периода времени. Это позволяет отслеживать активность базы данных и оценивать нагрузку на систему. Высокие значения этой метрики могут указывать на интенсивную работу с базой данных или на увеличенный трафик пользователей.

На рисунке 6.2 виден график секции во время активного пользования маркетплейсом.

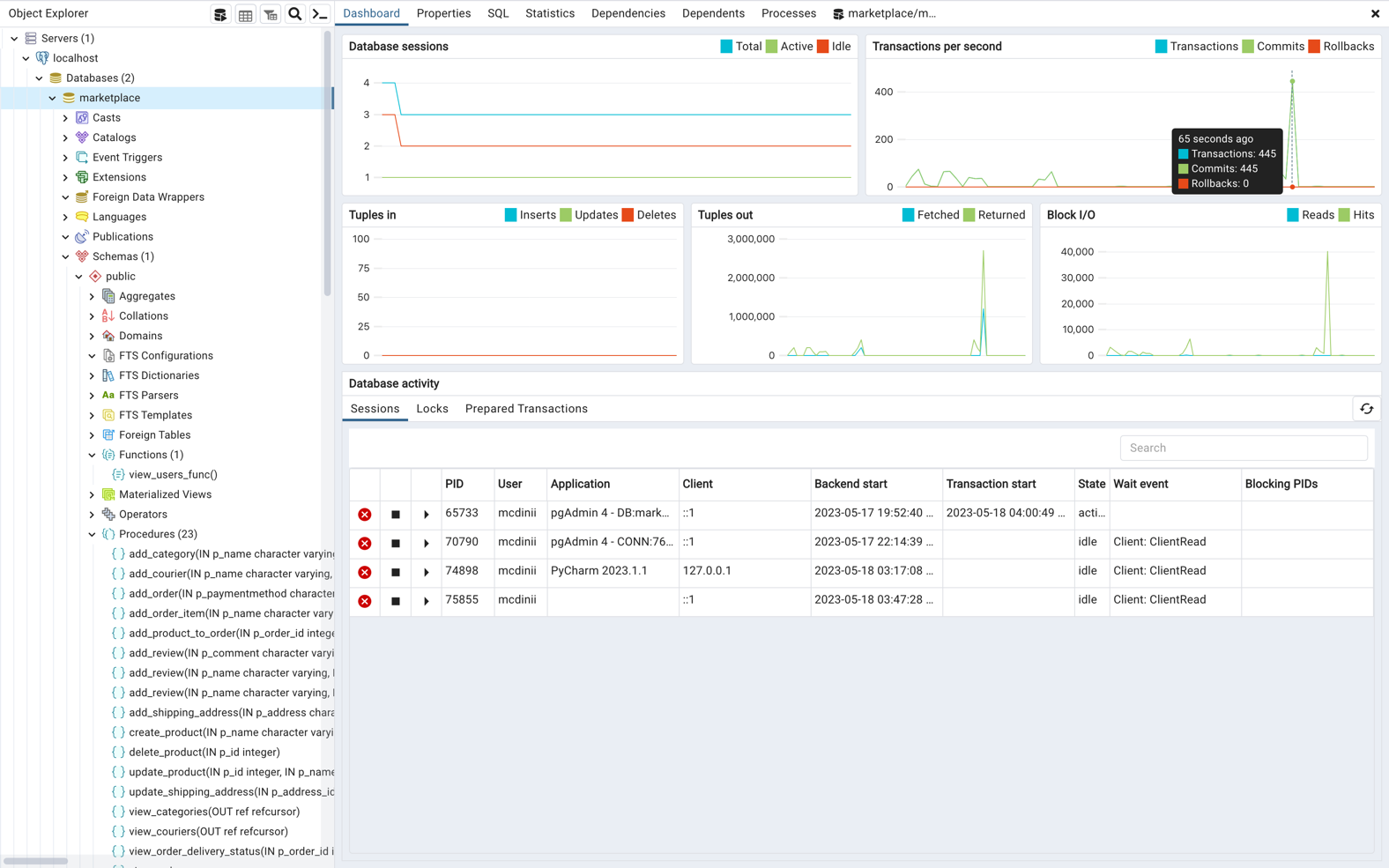


Рисунок 6.2 – Транзакции в секунду

## Блоки записаны (Blocks I/О)

Секция "Блоки записаны" отображает количество блоков данных, которые были записаны на диск в базе данных. Это позволяет оценить объем данных, которые были изменены или добавлены в базу данных. Высокие значения этой метрики могут указывать на интенсивное обновление данных или на запись больших объемов информации.

## Активность базы данных (Database Activity)

Секция "Активность базы данных" предоставляет информацию о текущей активности в базе данных. Она может включать количество активных сеансов, количество вставленных, удаленных и обновленных кортежей (строк) данных. Эти метрики помогают понять, какие операции выполняются в базе данных и насколько интенсивна их активность.

В нашем случае, на рисунке 6.3 видно, наши подключения из pgAdmin 4 в трех панелях Query Tool и подключение нашего приложения.

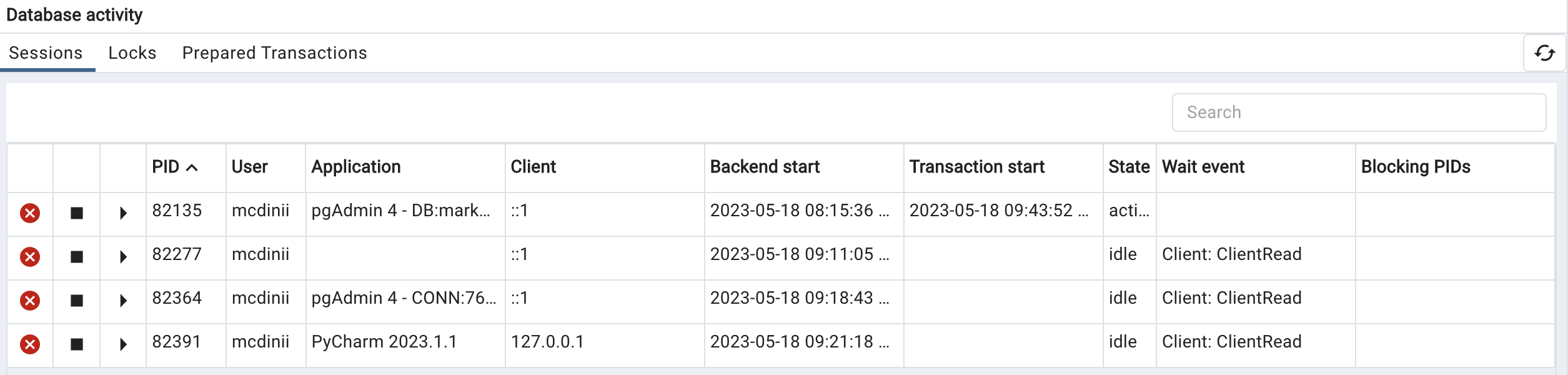


Рисунок 6.3 – Подключения к базе данных

## Кортежи входящие/исходящие (Tuples In/Out)

Секция "Кортежи входящие/исходящие" отображает количество кортежей (строк) данных, переданных в базу данных и отправленных из базы данных . Эта информация помогает оценить объем данных, с которыми работает база данных. Высокие значения этих метрик могут указывать на интенсивный обмен данными или на активную коммуникацию с внешними системами. На рисунке 6.4 видно, что примерно раз в 50 секунд база данных возвращает 1500, при обычном использование маркетплейса.

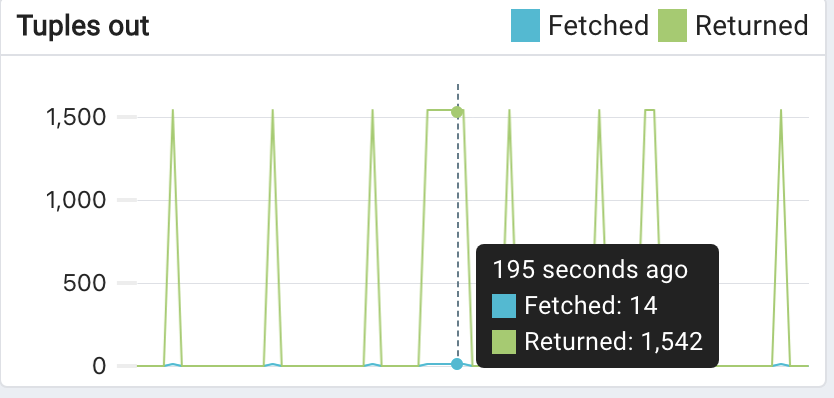


Рисунок 6.4 – Секция исходящие кортежи

Обычно, если количество "fetched" (полученных) и "returned" (возвращенных) кортежей одинаково, это означает, что все полученные данные были успешно возвращены и доступны для использования. Однако, в некоторых случаях, может возникнуть расхождение, например, если данные были отфильтрованы или ограничены при выполнении запроса, что может привести к различию между "fetched" и "returned" кортежами. У нас как раз такой случай.

## Вывод

PgAdmin 4 является мощным инструментом мониторинга для баз данных PostgreSQL. Он предоставляет администраторам баз данных полную информацию о работе системы, позволяя им отслеживать активность базы данных, производительность и нагрузку. Секция "Транзакции в секунду" позволяет администраторам увидеть количество выполняемых транзакций в единицу времени, что является важным показателем производительности базы данных. Благодаря информации, предоставляемой в секции "Блоки записаны", администраторы могут оценить нагрузку на дисковую систему и оптимизировать ее работу.

Другие важные секции включают "Активность базы данных" и "Кортежи входящие/исходящие". В секции "Активность базы данных" администраторы могут видеть текущие запросы, выполняемые на базе данных, и мониторить их выполнение. Это позволяет обнаруживать проблемные запросы или потенциальные узкие места в производительности. Секция "Кортежи входящие/исходящие" предоставляет информацию о количестве данных, передаваемых между клиентом и сервером. Это помогает администраторам оценить объем трафика и обнаружить возможные проблемы с сетевой производительностью.

В целом, pgAdmin 4 является важным инструментом для мониторинга и оптимизации баз данных PostgreSQL. Он обеспечивает администраторам ценную информацию, позволяющую повысить производительность системы, выявить и устранить проблемы, связанные с нагрузкой и эффективностью обработки данных, и обеспечить стабильную и эффективную работу базы данных.

# Краткое описание приложения для демонстрации

## Пример работы приложения от лица Администратора

Администратор имеет возможность просматривать, редактировать, удалять и добавлять товары на онлайн-маркетплейс, что видно на рисунке 7.1.

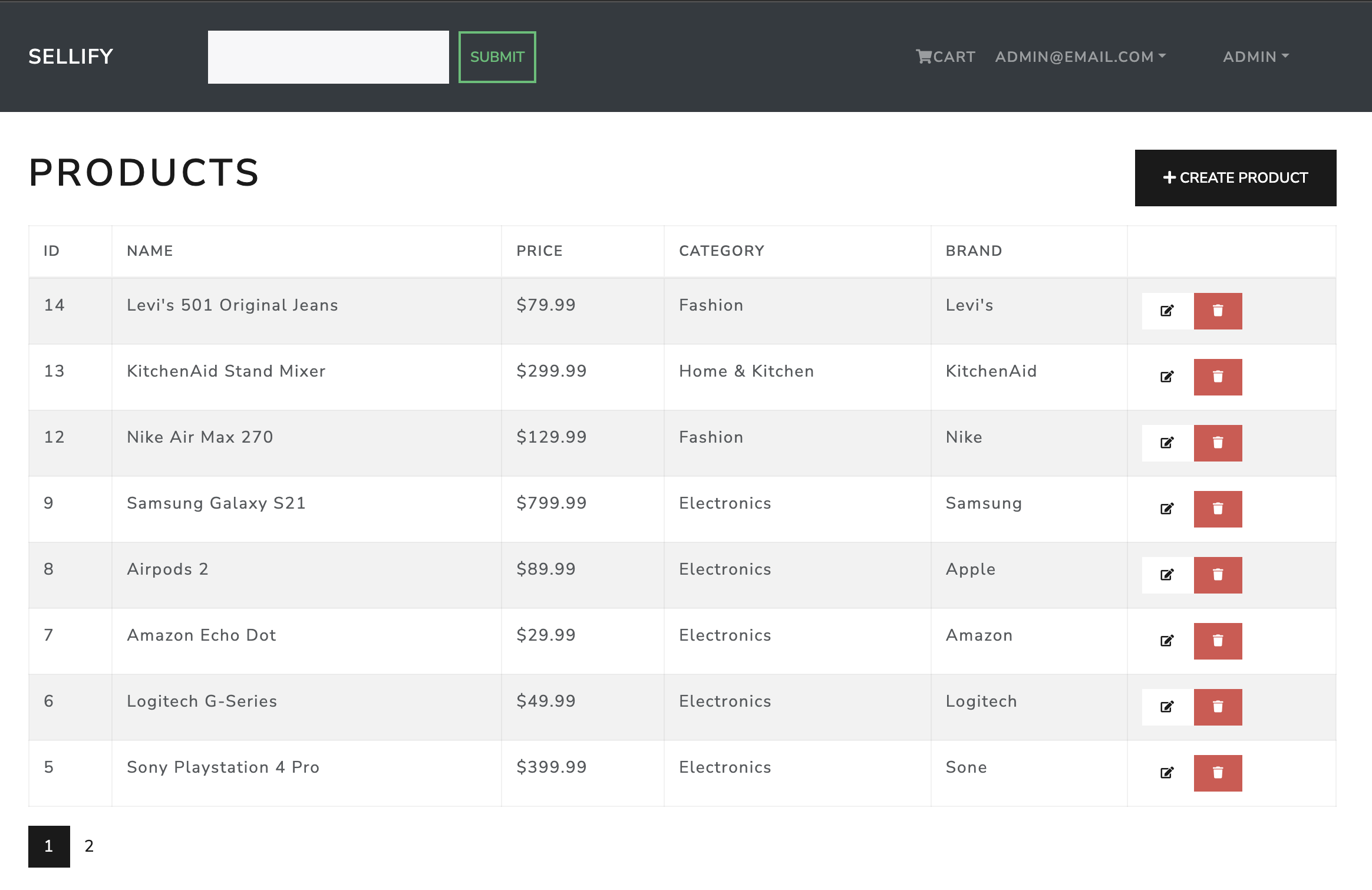


Рисунок 7.1 – Просмотр товаров Администратором

На рисунке 7.2 показан процесс добавления нового товара, после нажатия кнопки "Create Product" отображенную на рисунке 7.1.

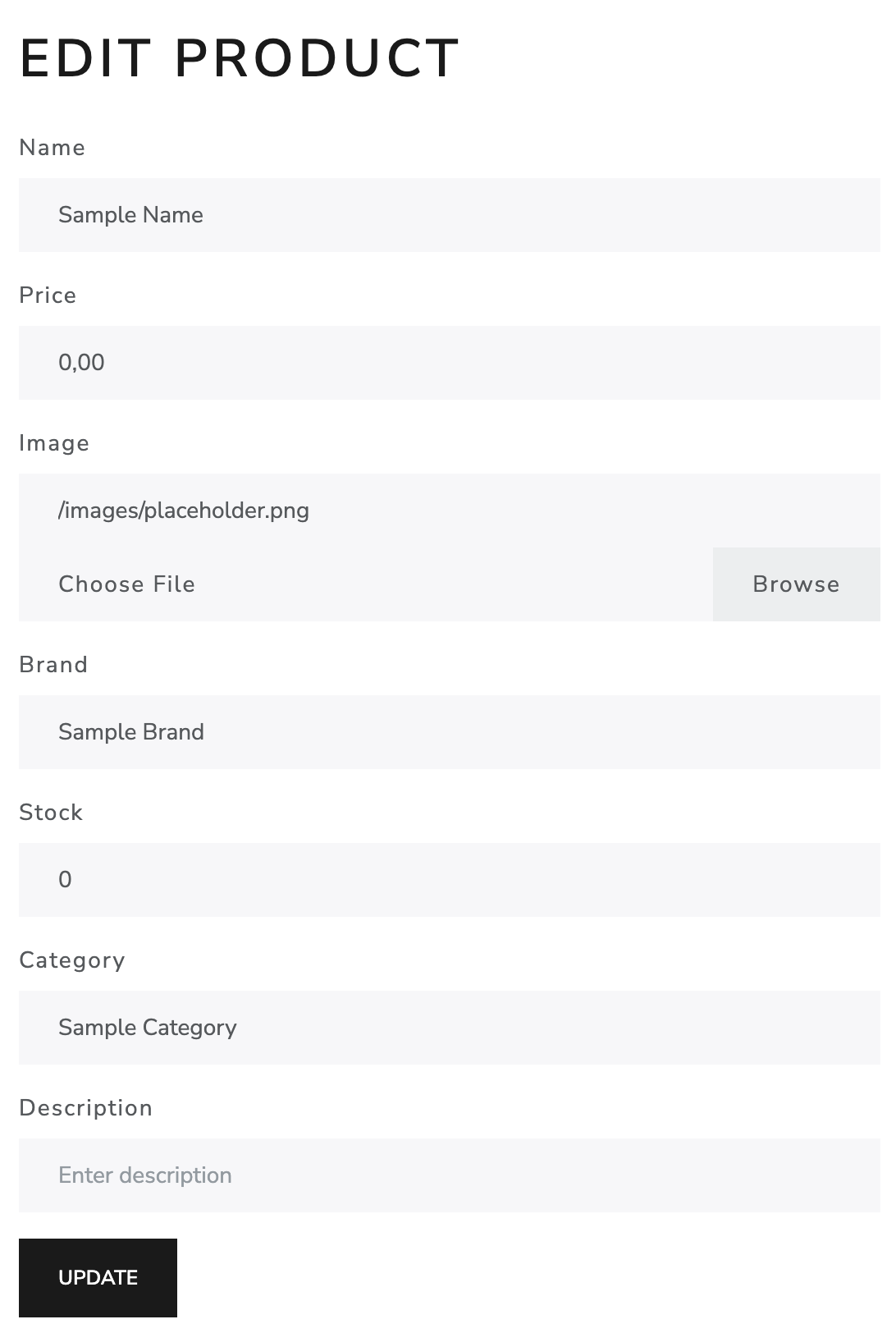


Рисунок 7.2 – Добавление товара

На рисунке 7.2 видны подсказки при заполнении.

Так же администратор может просматривать аккаунты пользователей и назначать их сотрудниками – Рисунок 7.3.

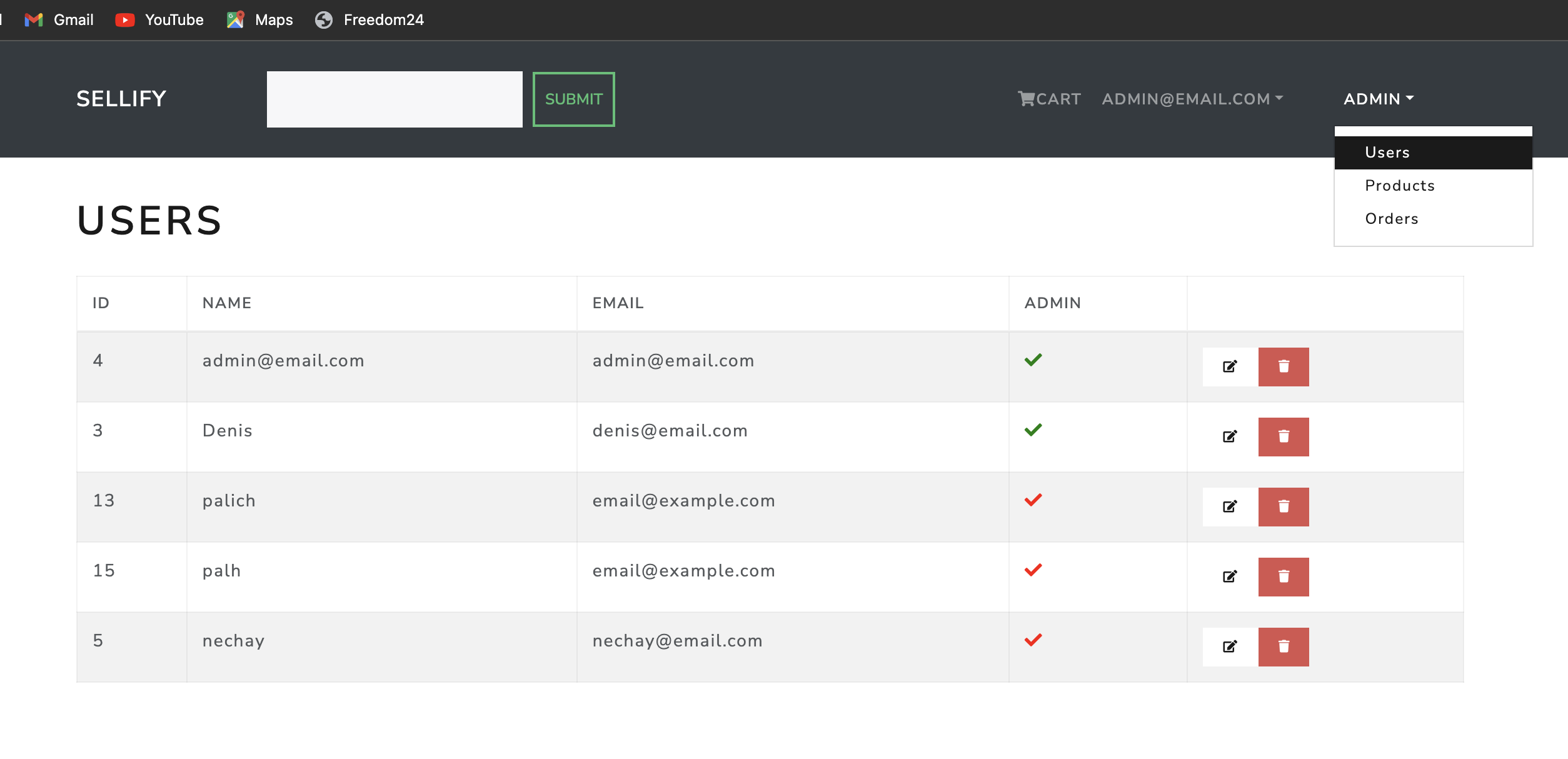


Рисунок 7.3 – Просмотр пользователей

## Пример работы приложения от лица Клиента

Любой клиент, даже не зарегистрированный может просматривать страницу с товарами, лучшими товарами за месяц, а также выбирать товары по категориям, этот функционал отображен на рисунке 7.4.

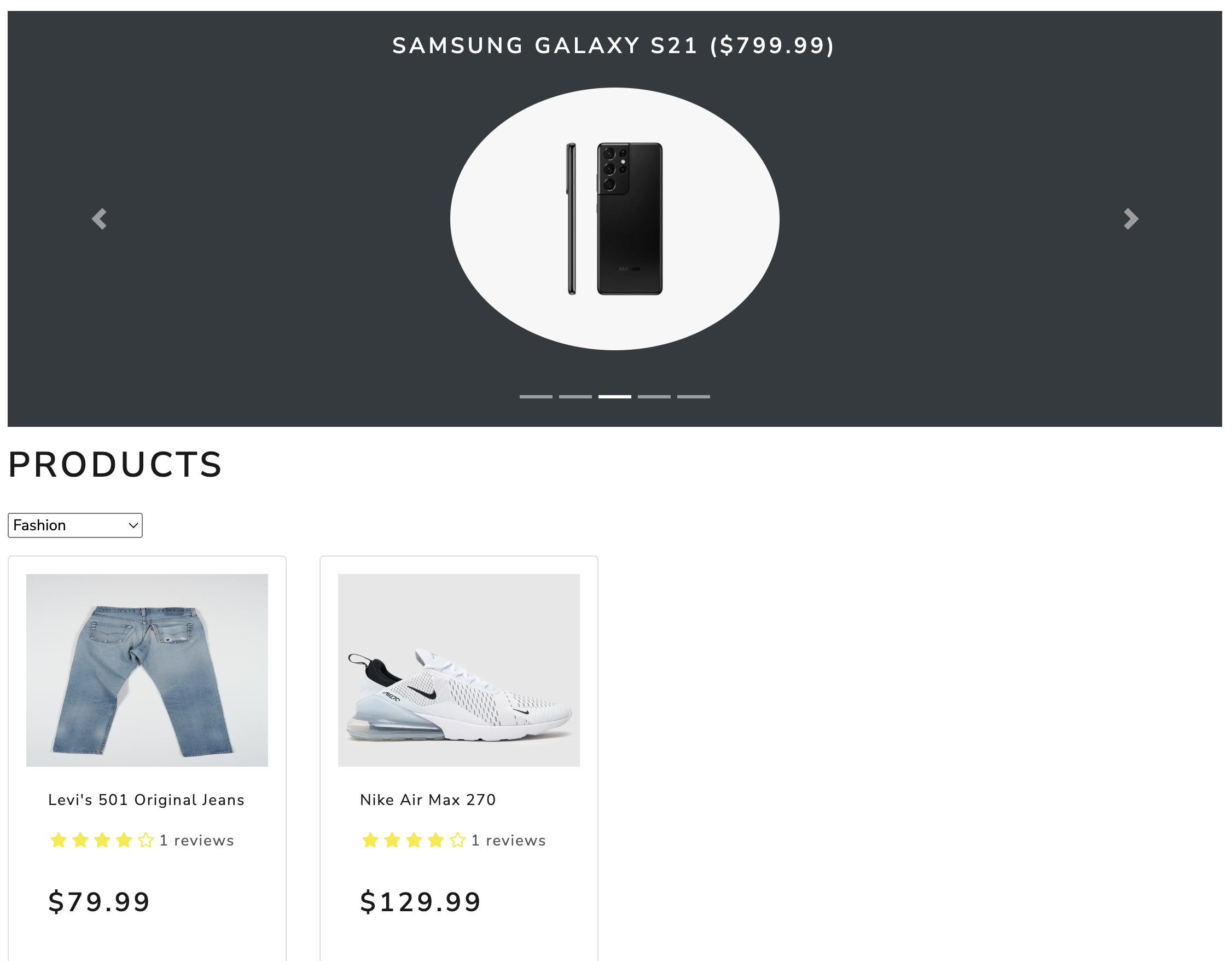


Рисунок 7.4 – Главная страница

Так же реализован поиск по каталогу.

# Руководство пользователя

## Руководство использования приложения

При первом входе на сайт отобразиться главная страница. Она приведена была приведена в разделе 7.

Далее пользователь может зарегистрироваться на сайте, чтобы иметь возможность добавлять товары в корзину, оставлять заказы и отзывы – Рисунок 8.1

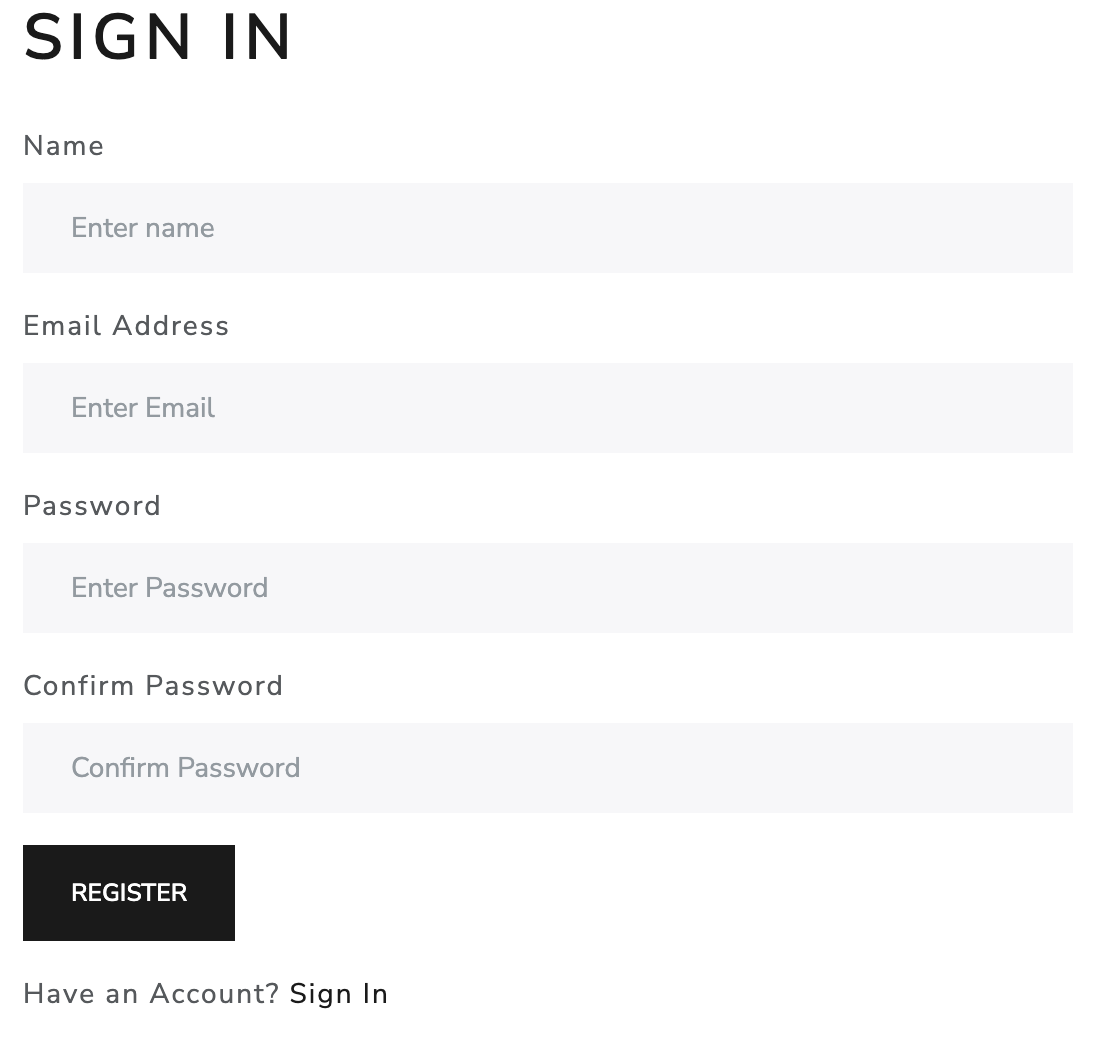


Рисунок 8.1 – Регистрация/Вход

При регистрации в системе требуется ввод пароля, который должен соответствовать определенным критериям безопасности. Это важно для обеспечения защиты аккаунта и конфиденциальности пользовательских данных. Критерии безопасности могут включать требование использования достаточно длинного пароля, содержащего различные типы символов, такие как буквы верхнего и нижнего регистра, цифры и специальные символы. Такой подход повышает стойкость пароля к взлому и предотвращает его угадывание или подбор методами перебора.

Кроме того, при регистрации пользователю обычно требуется повторно ввести пароль для подтверждения. Это служит для предотвращения ошибок при вводе пароля и подтверждает, что пользователь ввел пароль правильно. Соответствие пароля при подтверждении помогает избежать ситуаций, когда пользователь случайно вводит неправильный пароль и не может получить доступ к своему аккаунту. Это также улучшает пользовательский опыт, так как пользователь получает обратную связь о правильности введенного пароля и может внести исправления, если необходимо.

Также можно добавить товары в корзину, выбрать удобное время доставки, а так же необходимый вид оплаты и оформить заказ,. Функционал заказа товара показан на рисунках 8.2, 8.3, 8.4 и 8.5 соответственно.

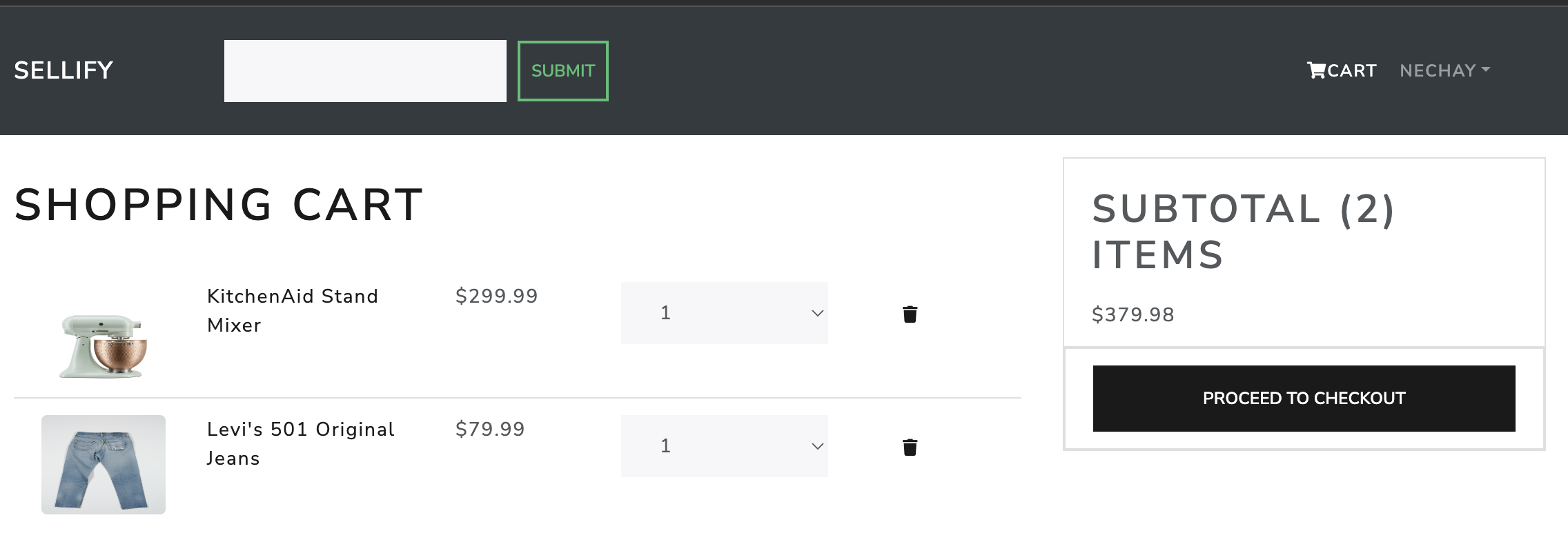


Рисунок 8.2 – Корзина

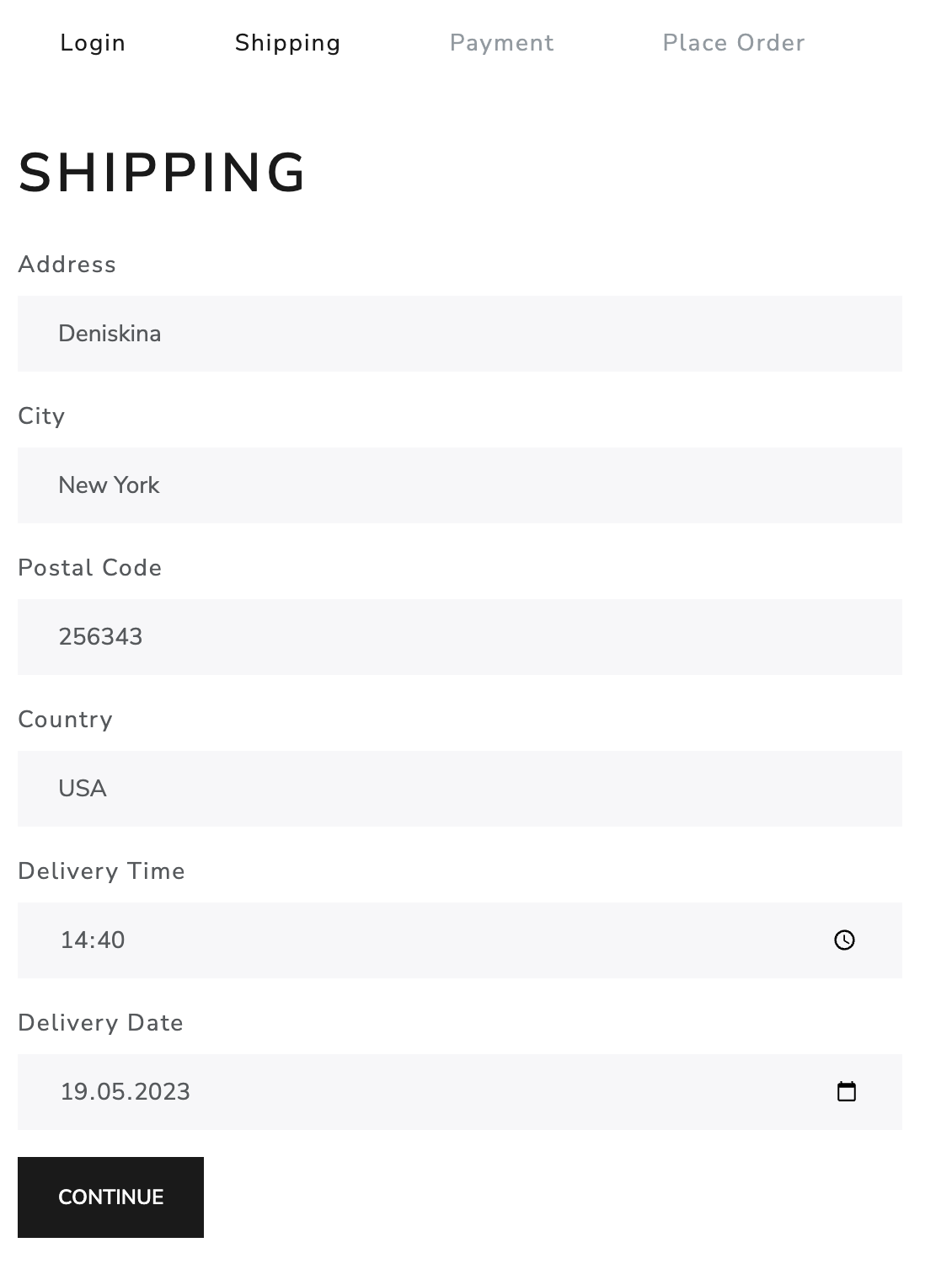


Рисунок 8.3 – Выбор времени доставки

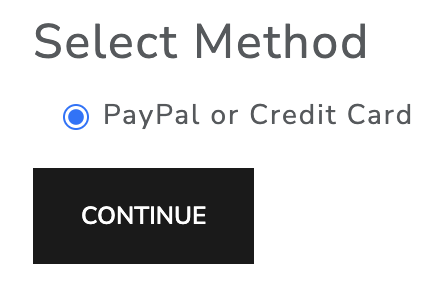


Рисунок 8.3 – Выбор оплаты

В разделе подтверждения заказа мы предоставляем пользователям полную информацию о заказе. Мы отображаем итоговую стоимость заказа, рассчитываем налоги и уточняем все детали, связанные с доставкой, выбранным методом оплаты и списком товаров.

Пользователь видит детальную информацию о каждом товаре, включая его название, цену и количество. Мы также предоставляем информацию о стоимости доставки и применяемых налогах. Это позволяет пользователям быть в курсе всех расходов, связанных с их заказом.

На этом этапе подтверждения заказа пользователь имеет возможность выбрать способ оплаты. У нас предусмотрена кнопка для оплаты картой, которая перенаправляет пользователя на защищенную платежную страницу. Также мы предлагаем опцию оплаты через PayPal, что обеспечивает удобство и безопасность при совершении платежей.

Все эти шаги позволяют пользователям окончательно уточнить и подтвердить свой заказ, ознакомиться с полной стоимостью и деталями заказа, а также выбрать удобный способ оплаты. Это создает удобную и прозрачную среду для пользователей, где они могут совершать заказы и производить оплату без лишних сложностей.

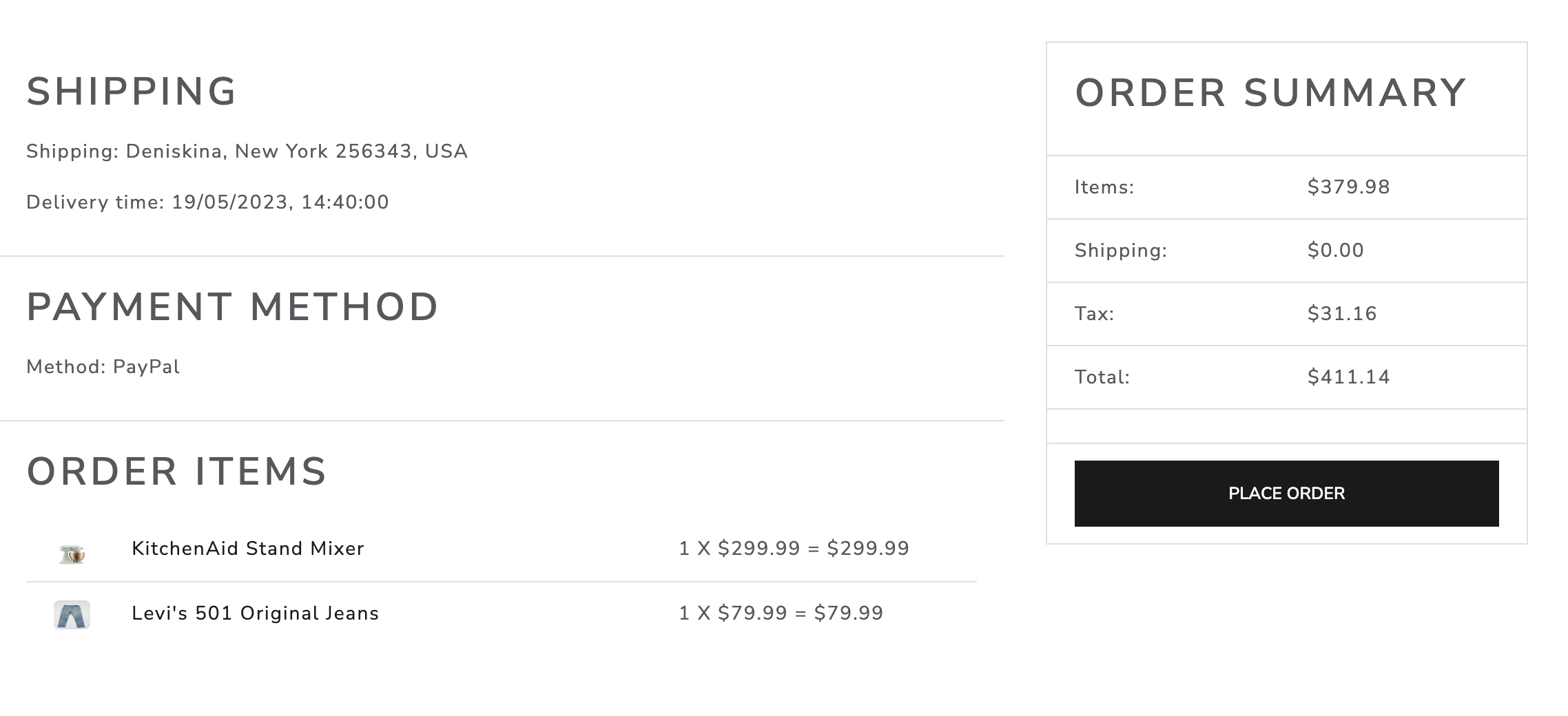


Рисунок 8.4 – Подтверждение заказа

Приложение разрабатывалось так, чтобы весь функционал был интуитивно понятен.

Заключение

В процессе выполнения курсовой работы были тщательно изучены требования к построению базы данных, осуществлена проектировка структуры базы данных, включающей таблицы, связи и ограничения. С использованием СУБД PostgreSQL была успешно реализована база данных, прошедшая последующее тестирование. Кроме того, было создано приложение онлайн-маркетплейса, демонстрирующее функциональность разработанной базы данных.

В дополнение к основной работе, в проект была интегрирована технология мониторинга состояния СУБД с применением инструмента pgAdmin 4. Данный инструмент предоставил возможность наблюдения за активностью базы данных, отслеживания производительности и нагрузки на систему. В процессе анализа транзакций в секунду, блоков записей, активности базы данных, а также информации о входящих и исходящих кортежах, была получена значимая информация для анализа работы базы данных и определения путей оптимизации ее производительности.

В результате выполнения данного проекта была создана функциональная база данных для онлайн-маркетплейса, обеспечивающая надежность и производительность системы. Проект подтверждает значимость баз данных в контексте онлайн-маркетплейсов и демонстрирует потенциал PostgreSQL и инструментов мониторинга для обеспечения высокой эффективности и надежности базы данных.

Список используемых источников

1. Блинова Е.А. Курс лекций по базам данных/ Е.А. Блинова.
2. Использование диаграммы вариантов использования UML [Электронный ресурс]. – Режим доступа: https://habr.com/ru/articles/566218/. – Дата доступа: 24.04.2023.
3. Инструмент управления PostgreSQL [Электронный ресурс]. – Режим доступа: https://www.pgadmin.org/docs. – Дата доступа: 25.04.2023.
4. PostgreSQL 14 изнутри [Электронный ресурс] – Режим доступа: https://www.postgrespro.ru/education/books/internals. – Дата доступа: 30.04.2023
5. Сравнение процедур и функций в PostgreSQL [Электронный ресурс]. – Режим доступа: https://proglib.io/p/funkcii-i-hranimye-procedury-v-sql-zachem-nuzhny-i-kak-primenyat-v-realnyh-primerah-2022-10-07. – Дата доступа: 02.05.2023.
6. Как устроенно индексирование баз данных [Электронный ресурс]. – Режим доступа: https://habr.com/ru/companies/ruvds/articles/724066. – Дата доступа: 04.05.2023.
7. Работа с данными формата JSON в PostgreSQL [Электронный ресурс]. – Режим доступа: https://konbert.com/blog/import-json-into-postgres-using-copy. – Дата доступа: 07.05.2023.
8. Основы мониторинга PostgreSQL [Электронный ресурс]. – Режим доступа: https://habr.com/ru/articles/486710/. – Дата доступа: 08.05.2023.

Приложение А

Листинг создания таблиц

CREATE TABLE IF NOT EXISTS public.base\_category

(

id integer NOT NULL DEFAULT nextval('base\_category\_id\_seq'::regclass),

name character varying(200) COLLATE pg\_catalog."default",

CONSTRAINT base\_category\_pkey PRIMARY KEY (id)

)

-- Table: public.auth\_user

-- DROP TABLE IF EXISTS public.auth\_user;

CREATE TABLE IF NOT EXISTS public.auth\_user

(

id integer NOT NULL DEFAULT nextval('auth\_user\_id\_seq'::regclass),

password character varying(128) COLLATE pg\_catalog."default" NOT NULL,

last\_login timestamp with time zone,

is\_superuser boolean NOT NULL,

username character varying(150) COLLATE pg\_catalog."default" NOT NULL,

first\_name character varying(150) COLLATE pg\_catalog."default" NOT NULL,

last\_name character varying(150) COLLATE pg\_catalog."default" NOT NULL,

email character varying(254) COLLATE pg\_catalog."default" NOT NULL,

is\_staff boolean NOT NULL,

is\_active boolean NOT NULL,

date\_joined timestamp with time zone NOT NULL,

CONSTRAINT auth\_user\_pkey PRIMARY KEY (id),

CONSTRAINT auth\_user\_username\_key UNIQUE (username)

)

-- Table: public.base\_courier

-- DROP TABLE IF EXISTS public.base\_courier;

CREATE TABLE IF NOT EXISTS public.base\_courier

(

id integer NOT NULL DEFAULT nextval('base\_courier\_id\_seq'::regclass),

name character varying(200) COLLATE pg\_catalog."default",

active boolean NOT NULL,

user\_id integer,

CONSTRAINT base\_courier\_pkey PRIMARY KEY (id),

CONSTRAINT base\_courier\_user\_id\_key UNIQUE (user\_id),

CONSTRAINT base\_courier\_user\_id\_b17ae6e5\_fk\_auth\_user\_id FOREIGN KEY (user\_id)

REFERENCES public.auth\_user (id) MATCH SIMPLE

ON UPDATE NO ACTION

ON DELETE NO ACTION

DEFERRABLE INITIALLY DEFERRED

)

-- Table: public.base\_order

-- DROP TABLE IF EXISTS public.base\_order;

CREATE TABLE IF NOT EXISTS public.base\_order

(

"paymentMethod" character varying(200) COLLATE pg\_catalog."default",

"taxPrice" numeric(7,2),

"shippingPrice" numeric(7,2),

"totalPrice" numeric(7,2),

"isPaid" boolean NOT NULL,

"paidAt" timestamp with time zone,

"isDelivered" boolean NOT NULL,

"deliveredAt" timestamp with time zone,

\_id integer NOT NULL DEFAULT nextval('base\_order\_\_id\_seq'::regclass),

user\_id integer,

"createdAt" timestamp with time zone NOT NULL,

status character varying(255) COLLATE pg\_catalog."default" NOT NULL,

"deliveryTime" timestamp with time zone,

CONSTRAINT base\_order\_pkey PRIMARY KEY (\_id),

CONSTRAINT base\_order\_user\_id\_8ad0adec\_fk\_auth\_user\_id FOREIGN KEY (user\_id)

REFERENCES public.auth\_user (id) MATCH SIMPLE

ON UPDATE NO ACTION

ON DELETE NO ACTION

DEFERRABLE INITIALLY DEFERRED

)

-- Table: public.base\_orderitem

-- DROP TABLE IF EXISTS public.base\_orderitem;

CREATE TABLE IF NOT EXISTS public.base\_orderitem

(

name character varying(200) COLLATE pg\_catalog."default",

qty integer,

price numeric(7,2),

image character varying(200) COLLATE pg\_catalog."default",

\_id integer NOT NULL DEFAULT nextval('base\_orderitem\_\_id\_seq'::regclass),

order\_id integer,

product\_id integer,

CONSTRAINT base\_orderitem\_pkey PRIMARY KEY (\_id),

CONSTRAINT base\_orderitem\_order\_id\_aaa7f08a\_fk\_base\_order\_\_id FOREIGN KEY (order\_id)

REFERENCES public.base\_order (\_id) MATCH SIMPLE

ON UPDATE NO ACTION

ON DELETE NO ACTION

DEFERRABLE INITIALLY DEFERRED,

CONSTRAINT base\_orderitem\_product\_id\_397c77c4\_fk\_base\_product\_\_id FOREIGN KEY (product\_id)

REFERENCES public.base\_product (\_id) MATCH SIMPLE

ON UPDATE NO ACTION

ON DELETE NO ACTION

DEFERRABLE INITIALLY DEFERRED

)

-- Table: public.base\_product

-- DROP TABLE IF EXISTS public.base\_product;

CREATE TABLE IF NOT EXISTS public.base\_product

(

name character varying(200) COLLATE pg\_catalog."default",

brand character varying(200) COLLATE pg\_catalog."default",

category\_id integer,

description text COLLATE pg\_catalog."default",

rating numeric(7,2),

"numReviews" integer,

price numeric(7,2),

"countInStock" integer,

"createdAt" timestamp with time zone NOT NULL,

\_id integer NOT NULL DEFAULT nextval('base\_product\_\_id\_seq'::regclass),

user\_id integer,

image character varying(100) COLLATE pg\_catalog."default",

CONSTRAINT base\_product\_pkey PRIMARY KEY (\_id),

CONSTRAINT base\_product\_category\_id\_2193a903\_fk\_base\_category\_id FOREIGN KEY (category\_id)

REFERENCES public.base\_category (id) MATCH SIMPLE

ON UPDATE NO ACTION

ON DELETE NO ACTION

DEFERRABLE INITIALLY DEFERRED,

CONSTRAINT base\_product\_user\_id\_95204f5a\_fk\_auth\_user\_id FOREIGN KEY (user\_id)

REFERENCES public.auth\_user (id) MATCH SIMPLE

ON UPDATE NO ACTION

ON DELETE NO ACTION

DEFERRABLE INITIALLY DEFERRED

)

-- Table: public.base\_review

-- DROP TABLE IF EXISTS public.base\_review;

CREATE TABLE IF NOT EXISTS public.base\_review

(

name character varying(200) COLLATE pg\_catalog."default",

rating integer,

comment text COLLATE pg\_catalog."default",

\_id integer NOT NULL DEFAULT nextval('base\_review\_\_id\_seq'::regclass),

product\_id integer,

user\_id integer,

"createdAt" timestamp with time zone NOT NULL,

CONSTRAINT base\_review\_pkey PRIMARY KEY (\_id),

CONSTRAINT base\_review\_product\_id\_9a95f2eb\_fk\_base\_product\_\_id FOREIGN KEY (product\_id)

REFERENCES public.base\_product (\_id) MATCH SIMPLE

ON UPDATE NO ACTION

ON DELETE NO ACTION

DEFERRABLE INITIALLY DEFERRED,

CONSTRAINT base\_review\_user\_id\_9a731b84\_fk\_auth\_user\_id FOREIGN KEY (user\_id)

REFERENCES public.auth\_user (id) MATCH SIMPLE

ON UPDATE NO ACTION

ON DELETE NO ACTION

DEFERRABLE INITIALLY DEFERRED

)

-- Table: public.base\_shippingaddress

-- DROP TABLE IF EXISTS public.base\_shippingaddress;

CREATE TABLE IF NOT EXISTS public.base\_shippingaddress

(

address character varying(200) COLLATE pg\_catalog."default",

city character varying(200) COLLATE pg\_catalog."default",

"postalCode" character varying(200) COLLATE pg\_catalog."default",

country character varying(200) COLLATE pg\_catalog."default",

"shippingPrice" numeric(7,2),

\_id integer NOT NULL DEFAULT nextval('base\_shippingaddress\_\_id\_seq'::regclass),

order\_id integer,

courier\_id integer,

CONSTRAINT base\_shippingaddress\_pkey PRIMARY KEY (\_id),

CONSTRAINT base\_shippingaddress\_order\_id\_key UNIQUE (order\_id),

CONSTRAINT base\_shippingaddress\_courier\_id\_4ed8b6fe\_fk\_base\_courier\_id FOREIGN KEY (courier\_id)

REFERENCES public.base\_courier (id) MATCH SIMPLE

ON UPDATE NO ACTION

ON DELETE NO ACTION

DEFERRABLE INITIALLY DEFERRED,

CONSTRAINT base\_shippingaddress\_order\_id\_d85223e7\_fk\_base\_order\_\_id FOREIGN KEY (order\_id)

REFERENCES public.base\_order (\_id) MATCH SIMPLE

ON UPDATE NO ACTION

ON DELETE NO ACTION

DEFERRABLE INITIALLY DEFERRED

)

Приложение Б

Листинг создания триггеров

-- Триггер 1: Обновление статуса оплаты

CREATE TRIGGER payment\_status\_trigger

BEFORE UPDATE ON public."base\_order"

FOR EACH ROW

WHEN (NEW.status <> OLD.status)

EXECUTE FUNCTION update\_payment\_status();

-- Триггер 2: Проверка наличия товара

CREATE TRIGGER product\_availability\_trigger

BEFORE INSERT ON public."base\_orderitem"

FOR EACH ROW

EXECUTE FUNCTION check\_product\_availability();

-- Триггер 3: Обновление количества товара

CREATE TRIGGER product\_quantity\_trigger

AFTER INSERT ON public."base\_orderitem"

FOR EACH ROW

EXECUTE FUNCTION update\_product\_quantity();

-- Триггер 4: Обновление статуса доставки

CREATE TRIGGER delivery\_status\_trigger

BEFORE UPDATE ON public."base\_order"

FOR EACH ROW

WHEN (NEW.status <> OLD.status)

EXECUTE FUNCTION update\_delivery\_status();

-- Триггер 5: Проверка доставки

CREATE TRIGGER check\_delivery\_time

BEFORE UPDATE ON "base\_order"

FOR EACH ROW

EXECUTE FUNCTION update\_order\_status();

Приложение В

Листинг создания основных процедур

-- Процедура добавления категории

CREATE OR REPLACE PROCEDURE public.add\_category(p\_name VARCHAR)

LANGUAGE plpgsql

AS $$

BEGIN

INSERT INTO public.base\_category(name) VALUES(p\_name);

END;

$$;

REVOKE ALL ON PROCEDURE public.add\_category(VARCHAR) FROM PUBLIC;

GRANT EXECUTE ON PROCEDURE public.add\_category(VARCHAR) TO admin\_role;

-- Вызов процедуры создание тестовой категории

CALL public.add\_category('TestCategory');

-- Процедура добавления продукта

CREATE OR REPLACE PROCEDURE public.add\_product(p\_name VARCHAR, p\_brand VARCHAR, p\_category\_id INTEGER, p\_description TEXT, p\_rating NUMERIC, p\_numReviews INTEGER, p\_price NUMERIC, p\_countInStock INTEGER, p\_created\_at DATA, p\_user\_id INTEGER, p\_image VARCHAR)

LANGUAGE plpgsql

AS $$

BEGIN

INSERT INTO public.base\_product(name, brand, category\_id, description, rating, "numReviews", price, "countInStock",created\_at user\_id, image)

VALUES(p\_name, p\_brand, p\_category\_id, p\_description, p\_rating, p\_numReviews, p\_price, p\_countInStock,p\_created\_at p\_user\_id, p\_image);

END;

$$;

REVOKE ALL ON PROCEDURE public.add\_product(VARCHAR, VARCHAR, INTEGER, TEXT, NUMERIC, INTEGER, NUMERIC, INTEGER, INTEGER, VARCHAR) FROM PUBLIC;

GRANT EXECUTE ON PROCEDURE public.add\_product(VARCHAR, VARCHAR, INTEGER, TEXT, NUMERIC, INTEGER, NUMERIC, INTEGER, INTEGER, VARCHAR) TO admin\_role;

--CALL public.add\_product('TestProduct', 'TestBrand', 1, 'TestDescription', 4.5, 10, 100.00, 50, "10.05.2023", 1, 'TestImage');

-- Процедура добавления отзыва

CREATE OR REPLACE PROCEDURE public.add\_review(p\_name VARCHAR, p\_rating INTEGER, p\_comment TEXT, p\_product\_id INTEGER, p\_user\_id INTEGER)

LANGUAGE plpgsql

AS $$

BEGIN

INSERT INTO public.base\_review(name, rating, comment, product\_id, user\_id)

VALUES(p\_name, p\_rating, p\_comment, p\_product\_id, p\_user\_id);

END;

$$;

REVOKE ALL ON PROCEDURE public.add\_review(VARCHAR, INTEGER, TEXT, INTEGER, INTEGER) FROM PUBLIC;

GRANT EXECUTE ON PROCEDURE public.add\_review(VARCHAR, INTEGER, TEXT, INTEGER, INTEGER) TO client\_role, guest\_role;

CALL public.add\_review('TestReview', 5, 'Great product!', 1, 1);

CREATE OR REPLACE PROCEDURE public.add\_shipping\_address(p\_address VARCHAR, p\_city VARCHAR, p\_postalCode VARCHAR, p\_country VARCHAR, p\_shippingPrice NUMERIC, p\_order\_id INTEGER, p\_courier\_id INTEGER)

LANGUAGE plpgsql

AS $$

BEGIN

INSERT INTO public.base\_shippingaddress(address, city, postalCode, country, shippingPrice, order\_id, courier\_id)

VALUES(p\_address, p\_city, p\_postalCode, p\_country, p\_shippingPrice, p\_order\_id, p\_courier\_id);

END;

$$;

REVOKE ALL ON PROCEDURE public.add\_shipping\_address(VARCHAR, VARCHAR, VARCHAR, VARCHAR, NUMERIC, INTEGER, INTEGER) FROM PUBLIC;

GRANT EXECUTE ON PROCEDURE public.add\_shipping\_address(VARCHAR, VARCHAR, VARCHAR, VARCHAR, NUMERIC, INTEGER, INTEGER) TO admin, courier;

CALL public.add\_shipping\_address('TestAddress', 'TestCity', 'TestPostalCode', 'TestCountry', 5.00, 1, 1);

set role mcdinii;

CREATE OR REPLACE PROCEDURE public.add\_order(

p\_paymentMethod VARCHAR,

p\_taxPrice NUMERIC,

p\_shippingPrice NUMERIC,

p\_totalPrice NUMERIC,

p\_user\_id INTEGER

)

LANGUAGE plpgsql

AS $$

BEGIN

INSERT INTO public.base\_order(

"paymentMethod",

"taxPrice",

"shippingPrice",

"totalPrice",

"isPaid",

"isDelivered",

"status",

"createdAt",

"user\_id",

"deliveryTime"

)

VALUES(

p\_paymentMethod,

p\_taxPrice,

p\_shippingPrice,

p\_totalPrice,

FALSE,

FALSE,

'Processing',

NOW(),

p\_user\_id,

NOW()

);

END;

$$;

REVOKE ALL ON PROCEDURE public.add\_order(VARCHAR, NUMERIC, NUMERIC, NUMERIC, INTEGER) FROM PUBLIC;

GRANT EXECUTE ON PROCEDURE public.add\_order(VARCHAR, NUMERIC, NUMERIC, NUMERIC, INTEGER) TO admin\_role;

REVOKE ALL ON PROCEDURE public.add\_order(VARCHAR, NUMERIC, NUMERIC, NUMERIC, INTEGER) FROM PUBLIC;

GRANT EXECUTE ON PROCEDURE public.add\_order(VARCHAR, NUMERIC, NUMERIC, NUMERIC, INTEGER) TO admin\_role;

REVOKE ALL ON PROCEDURE public.add\_order(VARCHAR, NUMERIC, NUMERIC, NUMERIC, INTEGER) FROM PUBLIC;

GRANT EXECUTE ON PROCEDURE public.add\_order(VARCHAR, NUMERIC, NUMERIC, NUMERIC, INTEGER) TO client;

CALL public.add\_order('TestPaymentMethod', 10.00, 5.00, 115.00, 1);

-- Процедура добавления элемента заказа

CREATE OR REPLACE PROCEDURE public.add\_order\_item(p\_name VARCHAR, p\_qty INTEGER, p\_price NUMERIC, p\_image VARCHAR, p\_order\_id INTEGER, p\_product\_id INTEGER)

LANGUAGE plpgsql

AS $$

BEGIN

INSERT INTO public.base\_orderitem(name, qty, price, image, order\_id, product\_id)

VALUES(p\_name, p\_qty, p\_price, p\_image, p\_order\_id, p\_product\_id);

END;

$$;

REVOKE ALL ON PROCEDURE public.add\_order\_item(VARCHAR, INTEGER, NUMERIC, VARCHAR, INTEGER, INTEGER) FROM PUBLIC;

GRANT EXECUTE ON PROCEDURE public.add\_order\_item(VARCHAR, INTEGER, NUMERIC, VARCHAR, INTEGER, INTEGER) TO client;

CALL public.add\_order\_item('TestOrderItem', 1, 100.00, 'TestImage', 1, 1);

-- Процедура добавления курьера

CREATE OR REPLACE PROCEDURE public.add\_courier(p\_name VARCHAR, p\_active BOOLEAN, p\_user\_id INTEGER)

LANGUAGE plpgsql

AS $$

BEGIN

INSERT INTO public.base\_courier(name, active, user\_id)

VALUES(p\_name, p\_active, p\_user\_id);

END;

$$;

REVOKE ALL ON PROCEDURE public.add\_courier(VARCHAR, BOOLEAN, INTEGER) FROM PUBLIC;

GRANT EXECUTE ON PROCEDURE public.add\_courier(VARCHAR, BOOLEAN, INTEGER) TO admin;

CALL public.add\_courier('TestCourier', TRUE, 1);

-- Процедура просмотра пользователей

CREATE OR REPLACE PROCEDURE public.view\_users(out ref REFCURSOR)

LANGUAGE plpgsql

AS $$

BEGIN

OPEN ref FOR SELECT id, username, first\_name, last\_name, email FROM public."auth\_user";

END;

$$;

REVOKE ALL ON PROCEDURE public.view\_users(REFCURSOR) FROM PUBLIC;

GRANT EXECUTE ON PROCEDURE public.view\_users(REFCURSOR) TO admin\_role;

CREATE OR REPLACE FUNCTION public.view\_users\_func()

RETURNS SETOF auth\_user

LANGUAGE plpgsql

AS $$

BEGIN

RETURN QUERY SELECT \* FROM public."auth\_user";

END;

$$;

select username, email, password,last\_login from public.view\_users\_func();

-- Процедура просмотра товаров

CREATE OR REPLACE PROCEDURE public.view\_products()

LANGUAGE plpgsql

AS $$

BEGIN

CREATE TEMPORARY TABLE temp\_products AS SELECT \* FROM public.base\_product;

SELECT \* FROM temp\_products;

DROP TABLE temp\_products;

END;

$$;

REVOKE ALL ON PROCEDURE public.view\_products() FROM PUBLIC;

GRANT EXECUTE ON PROCEDURE public.view\_products() TO admin\_role;

-- Процедура просмотра всех заказов

CREATE OR REPLACE PROCEDURE public.view\_orders()

LANGUAGE plpgsql

AS $$

BEGIN

CREATE TEMPORARY TABLE temp\_orders AS SELECT \* FROM public.base\_order;

SELECT \* FROM temp\_orders;

DROP TABLE temp\_orders;

END;

$$;

REVOKE ALL ON PROCEDURE public.view\_orders() FROM PUBLIC;

GRANT EXECUTE ON PROCEDURE public.view\_orders() TO admin\_role;

-- Процедура удаления продукта

CREATE OR REPLACE PROCEDURE public.delete\_product(p\_id INTEGER)

LANGUAGE plpgsql

AS $$

BEGIN

DELETE FROM public.base\_product WHERE \_id = p\_id;

END;

$$;

REVOKE ALL ON PROCEDURE public.delete\_product(INTEGER) FROM PUBLIC;

GRANT EXECUTE ON PROCEDURE public.delete\_product(INTEGER) TO admin;

CALL public.delete\_product(1);

--Просмотр категорий

CREATE OR REPLACE PROCEDURE public.view\_categories(out ref REFCURSOR)

LANGUAGE plpgsql

AS $$

BEGIN

OPEN ref FOR SELECT \* FROM public."base\_category";

END;

$$;

-- Просмотр товаров в категории

CREATE OR REPLACE PROCEDURE public.view\_products\_in\_category(out ref REFCURSOR, p\_category\_id INTEGER)

LANGUAGE plpgsql

AS $$

BEGIN

OPEN ref FOR SELECT \* FROM public."base\_product" WHERE "category\_id" = p\_category\_id;

END;

$$;

-- Просмотр детальной информации о товаре

CREATE OR REPLACE PROCEDURE public.view\_product\_details(out ref REFCURSOR, p\_product\_id INTEGER)

LANGUAGE plpgsql

AS $$

BEGIN

OPEN ref FOR SELECT \* FROM public."base\_product" WHERE "id" = p\_product\_id;

END;

$$;

-- Просмотр отзывов о товаре

CREATE OR REPLACE PROCEDURE public.view\_product\_reviews(out ref REFCURSOR, p\_product\_id INTEGER)

LANGUAGE plpgsql

AS $$

BEGIN

OPEN ref FOR SELECT \* FROM public."base\_review" WHERE "product\_id" = p\_product\_id;

END;

$$;

-- Процедура добавления товара в заказ

CREATE OR REPLACE PROCEDURE public.add\_product\_to\_order(

p\_order\_id INTEGER,

p\_product\_id INTEGER,

p\_quantity INTEGER

)

LANGUAGE plpgsql

AS $$

BEGIN

INSERT INTO public.base\_order\_item(order\_id, product\_id, quantity)

VALUES(p\_order\_id, p\_product\_id, p\_quantity);

END;

$$;

-- Процедура обновления информации об адресе доставки

CREATE OR REPLACE PROCEDURE public.update\_shipping\_address(

p\_address\_id INTEGER,

p\_new\_address VARCHAR

)

LANGUAGE plpgsql

AS $$

BEGIN

UPDATE public.base\_shippingadress

SET address = p\_new\_address

WHERE id = p\_address\_id;

END;

$$;

--Процедура для просмотра отзывов

CREATE OR REPLACE PROCEDURE public.view\_product\_reviews(p\_product\_id INTEGER, OUT ref REFCURSOR)

LANGUAGE plpgsql

AS $$

BEGIN

OPEN ref FOR SELECT \* FROM public."base\_review" WHERE "product\_id" = p\_product\_id;

END;

$$;

-- Процедура просмотра товаров для гостя

CREATE OR REPLACE PROCEDURE public.view\_products\_for\_guest()

LANGUAGE plpgsql

AS $$

DECLARE

rec public."base\_product"%rowtype; -- Переменная-запись для хранения строки

BEGIN

FOR rec IN SELECT \* FROM public."base\_product" LOOP

-- Выводим данные из таблицы

RAISE NOTICE 'Product: %', rec;

END LOOP;

END;

$$;

Приложение Г

Листинг создания функций

CREATE OR REPLACE FUNCTION check\_product\_availability()

RETURNS TRIGGER AS $$

DECLARE

product\_quantity INTEGER;

BEGIN

SELECT quantity INTO product\_quantity FROM base\_product WHERE id = NEW.product\_id;

IF product\_quantity < 1 THEN

RAISE EXCEPTION 'Product is out of stock';

END IF;

RETURN NEW;

END;

$$ LANGUAGE plpgsql;

CREATE OR REPLACE FUNCTION update\_payment\_status()

RETURNS TRIGGER AS $$

BEGIN

IF NEW.status = 'Paid' THEN

NEW."isPaid" := TRUE;

END IF;

RETURN NEW;

END;

$$ LANGUAGE plpgsql;

CREATE OR REPLACE FUNCTION update\_product\_quantity()

RETURNS TRIGGER AS $$

BEGIN

UPDATE base\_product SET quantity = quantity - 1 WHERE id = NEW.product\_id;

RETURN NEW;

END;

$$ LANGUAGE plpgsql;

CREATE OR REPLACE FUNCTION update\_delivery\_status()

RETURNS TRIGGER AS $$

BEGIN

IF NEW.status = 'Delivered' THEN

NEW."isDelivered" := TRUE;

END IF;

RETURN NEW;

END;

$$ LANGUAGE plpgsql;

CREATE OR REPLACE FUNCTION update\_order\_status() RETURNS TRIGGER AS $$

BEGIN

IF NEW."deliveryTime" <= current\_timestamp AND NEW."status" != 'Delivered' THEN

NEW."status" := 'Delivered';

NEW."isDelivered" := TRUE;

NEW."deliveredAt" := current\_timestamp;

RAISE NOTICE 'Order % is now Delivered', NEW."\_id";

ELSIF NEW."deliveryTime" > current\_timestamp AT TIME ZONE 'UTC' AND NEW."status" = 'Delivered' THEN

RAISE NOTICE 'Order % should not be Delivered yet', NEW."\_id";

END IF;

RETURN NEW;

END;

$$ LANGUAGE plpgsql;

CREATE OR REPLACE FUNCTION register\_courier(

courier\_name VARCHAR,

user\_name VARCHAR,

first\_name VARCHAR,

last\_name VARCHAR,

user\_password VARCHAR,

user\_email VARCHAR

) RETURNS VOID AS $$

BEGIN

INSERT INTO auth\_user (username,first\_name,last\_name, password, email, is\_staff, is\_active, is\_superuser, date\_joined)

VALUES (user\_name,first\_name,last\_name, user\_password, user\_email, FALSE, TRUE, FALSE, NOW());

INSERT INTO base\_courier (user\_id, name, active)

VALUES ((SELECT id FROM auth\_user WHERE username = user\_name), courier\_name, TRUE);

END;

$$ LANGUAGE plpgsql;

CREATE OR REPLACE FUNCTION check\_table\_size(in\_table\_name TEXT)

RETURNS TABLE(table\_name TEXT, size\_bytes BIGINT) AS $$

BEGIN

RETURN QUERY

SELECT CAST(t.table\_name AS TEXT), pg\_total\_relation\_size(t.table\_name::regclass)

FROM information\_schema.tables AS t

WHERE t.table\_schema = 'public'

AND t.table\_name = in\_table\_name

ORDER BY pg\_total\_relation\_size(t.table\_name::regclass) DESC;

END;

$$ LANGUAGE plpgsql;

CREATE OR REPLACE FUNCTION check\_table\_indexes\_size(in\_table\_name TEXT)

RETURNS TABLE(index\_name TEXT, size\_bytes BIGINT) AS $$

BEGIN

RETURN QUERY

SELECT

CAST(i.relname AS TEXT) AS index\_name,

pg\_total\_relation\_size(i.oid) AS size\_bytes

FROM

pg\_class t,

pg\_class i,

pg\_index ix

WHERE

t.oid = ix.indrelid

AND i.oid = ix.indexrelid

AND t.relkind = 'r'

AND t.relname = in\_table\_name;

END;

$$ LANGUAGE plpgsql;

Приложение Д

Листинг создания индексов

-- Индекс "Auth\_index" на таблице "auth\_user" по столбцу "username"

CREATE INDEX Auth\_index ON auth\_user (username);

-- Индекс "Order\_status\_index" на таблице "Order" по столбцу "status"

CREATE INDEX Order\_status\_index ON "base\_order" ("status");

-- Индекс "Courier\_active\_index" на таблице "Courier" по столбцу "active"

CREATE INDEX Courier\_active\_index ON base\_courier (active);

-- Индекс "Order\_user\_index" на таблице "Order" по столбцу "user\_id"

CREATE INDEX IF NOT EXISTS base\_order\_user\_id\_8ad0adec

ON public.base\_order USING btree

(user\_id ASC NULLS LAST)

TABLESPACE pg\_default;