СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 5](#_Toc169186970)

[1. ИЗУЧЕНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ 6](#_Toc169186971)

[1.1. Изучение аналогов разрабатываемого программного продукта 6](#_Toc169186972)

[1.2. Достоинства и недостатки существующих систем 7](#_Toc169186973)

[1.3. Архитектура и реализация программного средства на базе анализа и выбранных технологий 8](#_Toc169186974)

[1.4. Вывод о конечных пользователях программы 9](#_Toc169186975)

[2. ПОСТАНОВКА ЗАДАЧИ И ПРОЕКТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА 10](#_Toc169186976)

[2.1. Постановка задачи 10](#_Toc169186977)

[2.2. Проектирование архитектуры данных 11](#_Toc169186978)

[2.3. Выбор и обоснование средств разработки 13](#_Toc169186979)

[2.4. Создание диаграмм 14](#_Toc169186980)

[3. РЕАЛИЗАЦИЯ. ОПИСАНИЕ АЛГОРИТМОВ РАБОТЫ ПРОГРАММНОГО СРЕДСТВА 17](#_Toc169186981)

[3.1. Проектирование программного средства 17](#_Toc169186982)

[3.2. Аутентификация и авторизация пользователей 20](#_Toc169186983)

[3.3. Хранение данных в PostgreSQL 20](#_Toc169186984)

[3.4. Обработка заказа 21](#_Toc169186985)

[4. ТЕСТИРОВАНИЕ 22](#_Toc169186986)

[4.1. Общая информация 22](#_Toc169186987)

[4.2. Чек-листы 25](#_Toc169187003)

[4.3. Тест-кейсы программного средства 26](#_Toc169187004)

[4.4. Вывод о результатах тестирования 29](#_Toc169187005)

[5. ЭКОНОМИЧЕСКИЙ РАЗДЕЛ 30](#_Toc169187006)

[5.1. Исходная информация 30](#_Toc169187007)

[5.2. Определение себестоимости программного обеспечения (ПО), как базы для формирования цены 30](#_Toc169187008)

[5.3. Выбор схемы монетизации программного продукта 37](#_Toc169187014)

[ЗАКЛЮЧЕНИЕ 40](#_Toc169187015)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 41](#_Toc169187016)

[ПРИЛОЖЕНИЕ 1 ИСХОДНЫЙ ТЕКСТ ПРОГРАММНОГО СРЕДСТВА 42](#_Toc169187017)

[ПРИЛОЖЕНИЕ 2 БЛОК-СХЕМА АЛГОРИТМА РАБОТЫ ПРОГРАММНОГО СРЕДСТВА 50](#_Toc169187018)

[ПРИЛОЖЕНИЕ 3 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ 51](#_Toc169187019)

[ПРИЛОЖЕНИЕ 4 ПРОТОКОЛ ИСПЫТАНИЙ ПРОГРАММНОГО СРЕДСТВА 52](#_Toc169187020)

# ВВЕДЕНИЕ

Современные веб-приложения все больше полагаются на надежные и эффективные backend системы для обеспечения функциональности, безопасности и масштабируемости. Разработка backend компонентов является ключевым аспектом создания успешных онлайн-сервисов, включая веб-приложения для зоомагазинов.

Тема дипломного проекта – «Разработка backend сайта зоомагазина». Этот проект актуален, так как позволит создать надежную и функциональную серверную часть сайта, обеспечивающую эффективное взаимодействие с клиентами и управление данными, что будет способствовать развитию и конкурентоспособности зоомагазина.

Целью дипломного проектирования является реализация современной и масштабируемой backend системы сайта для зоомагазина.

Чтобы достичь поставленной цели, необходимо решить следующие задачи:

­­­ — изучить основные принципы разработки backend систем, включая выбор технологий, архитектурных подходов и методов обработки запросов;

— провести анализ требований к функциональности и безопасности backend системы зоомагазина, учитывая специфику операций с товарами для животных и потребности клиентов;

— спроектировать архитектуру backend системы, определить структуру базы данных и выбрать подходящие технологии для реализации функциональности;

— реализовать backend компоненты с использованием современных технологий веб-разработки, таких, как языки программирования (например, Python, Java, или Node.js) и фреймворки (например, Django, Spring или Express.js);

— провести тестирование и оптимизацию работы backend системы с целью обеспечения высокой производительности, масштабируемости, безопасности и надежности;

— обеспечить интеграцию frontend и backend компонентов для создания полноценного веб-приложения зоомагазина;

Плюсом от использования программного средства, разработанного в рамках дипломного проекта, является обеспечение безопасности данных, эффективной обработки запросов, масштабируемости и интеграции с другими системами.

1. ИЗУЧЕНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ
   1. Изучение аналогов разрабатываемого программного продукта

С целью анализа предметной области разрабатываемого программного продукта были рассмотрены нижеперечисленные веб-ресурсы и выявлены их сильные и слабые стороны.

Зоокомпас (zoocompass.ru).

Зоокомпас – это интернет-зоомагазин, специализирующийся на товарах для домашних животных. Сайт предлагает широкий ассортимент товаров, включая корма, лакомства, игрушки, аксессуары, средства гигиены и многое другое для различных видов домашних питомцев, таких как собаки, кошки, грызуны, птицы и рыбы.

Функции Зоокомпаса включают следующее:

* удобный поиск и навигация (сайт предоставляет удобный интерфейс для поиска товаров, где вы можете фильтровать по виду животного, бренду, цене и другим параметрам. Это помогает быстро найти нужные товары);
* детальная информация о товарах (каждый товар сопровождается подробным описанием, характеристиками, фотографиями и инструкциями по использованию. Вы также можете прочитать отзывы покупателей, чтобы получить дополнительную информацию);
* онлайн-заказ и доставка (вы можете легко разместить заказ на выбранные товары через сайт. Зоокомпас предлагает различные способы доставки, и вы можете выбрать удобную для вас опцию. Заказы доставляются непосредственно к вам домой или в указанный адрес);
* клиентская поддержка (зоокомпас предлагает различные каналы связи, такие как электронная почта и телефон, чтобы помочь вам с вопросами, консультациями или решением проблем, связанных с заказами или товарами.

Питомец (pitomets.ru).

Питомец – это ещё один интернет-зоомагазин, предлагающий широкий выбор товаров для домашних животных. Сайт специализируется на товарах для собак и кошек, включая корма, лакомства, игрушки, лежаки, аксессуары, средства гигиены и многое другое.

Функции Питомца включают следующее:

* автоматическая доставка (питомец предлагает услугу автоматической доставки, которая позволяет настроить регулярные поставки корма и других товаров для вашего питомца. Вы можете установить удобное расписание доставки и быть уверенным, что у вас всегда есть необходимые товары);
* частые акции и скидки (сайт предлагает регулярные акции и скидки на различные товары, что позволяет сэкономить деньги при покупке товаров для вашего питомца);
* советы и статьи (питомец предоставляет полезные советы и информацию о заботе, воспитании и здоровье собак и кошек. Вы можете найти статьи о правильном питании, тренировке, уходе за шерстью и многом другом);
* отзывы и оценки (вы можете прочитать отзывы и оценки других покупателей о товарах на сайте, чтобы получить представление о качестве и рекомендациях).

Оба этих интернет-зоомагазина предлагают удобные способы покупки товаров для домашних животных, широкий выбор продуктов и возможность получить дополнительную информацию о товарах перед покупкой. Важно отметить, что функции и особенности могут изменяться со временем, поэтому рекомендуется посетить сайты Зоокомпаса (zoocompass.ru) и Питомца (pitomets.ru) для получения актуальной информации о них.

* 1. Достоинства и недостатки существующих систем

При проведении предметного анализа и изучении существующих программных решений можно выделить ряд их достоинств и недостатков.

Достоинства существующих программных решений:

* Функциональность (существующие программы обладают разнообразным набором функций, позволяющих эффективно управлять бизнес-процессами. Они предоставляют возможности управления каталогом товаров, оформления заказов, учета и отчетности, а также интеграции с другими системами);
* удобство использования (большинство программных решений разработаны с учетом удобства использования. Интуитивный интерфейс и понятные рабочие процессы делают их доступными для пользователей разного уровня технической подготовки);
* гибкость настройки (некоторые программы предоставляют возможность настраивать пользовательский интерфейс и рабочие процессы в соответствии с индивидуальными потребностями и предпочтениями пользователей. Это позволяет адаптировать программное решение под конкретные бизнес-процессы и повышает его эффективность);
* возможность интеграции (некоторые программы предлагают интеграцию с другими системами, такими как системы учета, электронные платежи или системы управления клиентскими отношениями (CRM). Это обеспечивает единый поток данных и повышает автоматизацию бизнес-процессов).

Несмотря на достоинства, существуют и некоторые недостатки существующих программных решений:

* ограниченный функционал (некоторые программы предлагают ограниченный набор функций, что может ограничивать способность бизнеса адаптироваться к изменяющимся требованиям и рыночным условиям);
* несовместимость с другими системами (некоторые программы могут быть несовместимы с существующими системами, что создает сложности в интеграции и обмене данных между различными приложениями);
* ограниченная настраиваемость (некоторые программные решения не обладают достаточной гибкостью настройки пользовательского интерфейса и рабочих процессов. Это может приводить к необходимости компромиссов и неэффективному использованию программного решения);
* отсутствие поддержки и обновлений (в случае, если программное решение не получает достаточной поддержки со стороны разработчиков, возникают проблемы с исправлением ошибок, обновлением функционала и обеспечением безопасности).

Важно учитывать и анализировать как достоинства, так и недостатки существующих программных решений при выборе или разработке нового программного решения. Это позволяет создать эффективное и инновационное решение, которое лучше соответствует потребностям бизнеса и клиентов.

* 1. Архитектура и реализация программного средства на базе анализа и выбранных технологий

На основе проведенного анализа можно сделать вывод о необходимости разработки программного средства, которое будет обладать следующими особенностями:

* расширенный функционал (учитывая ограниченный функционал некоторых существующих программных решений, целесообразно разработать программное средство с широким набором функций. Это позволит бизнесу адаптироваться к изменяющимся требованиям и эффективно управлять бизнес-процессами);
* интеграция с другими системами (учитывая значимость взаимодействия с другими системами, новое программное средство должно предоставлять возможность интеграции с различными системами, такими как системы учета, электронные платежи или системы управления клиентскими отношениями (CRM). Это обеспечит единый поток данных и повысит автоматизацию бизнес-процессов);
* гибкость настройки (важно разработать программное средство, которое обладает гибкостью настройки пользовательского интерфейса и рабочих процессов. Это позволит пользователям адаптировать программу под свои потребности и повысит эффективность ее использования);
* поддержка и обновления (разработчики нового программного средства должны обеспечивать регулярную поддержку и обновления, чтобы исправлять ошибки, добавлять новый функционал и обеспечивать безопасность системы).

На основе указанных требований можно рассмотреть разработку веб-приложения с использованием Gin Gonic фреймворка в качестве основы для языка Go, PostgreSQL в качестве базы данных. Этот стек технологий обладает хорошей производительностью, надежностью и позволяет разработать гибкое и масштабируемое программное средство.

* 1. Вывод о конечных пользователях программы

На основе проведенного анализа и разработки программного средства, следует сделать вывод о конечных пользователях программы.

В результате разработки программного средства можно сделать вывод о следующих конечных пользователях.

Руководители и владельцы бизнеса. Программное средство предоставляет руководителям и владельцам бизнеса инструменты для эффективного управления и контроля над бизнес-процессами. Оно позволяет им получать актуальную информацию о продажах, складских запасах, финансовых показателях и других ключевых аспектах деятельности компании. Это помогает принимать информированные решения и оптимизировать работу бизнеса.

Сотрудники отделов продаж, логистики и учета. Программное средство облегчает работу сотрудников отделов продаж, логистики и учета. Оно предоставляет им удобные инструменты для управления каталогом товаров, оформления заказов, отслеживания поставок и учета финансовых операций. Это повышает эффективность работы сотрудников и сокращает время на выполнение рутинных задач.

Клиенты. Программное средство может иметь также внешний интерфейс для клиентов, что позволяет им удобно оформлять заказы, получать информацию о товарах и услугах, а также отслеживать статус выполнения заказа. Это улучшает пользовательский опыт клиентов и способствует укреплению взаимоотношений с ними.

1. ПОСТАНОВКА ЗАДАЧИ И ПРОЕКТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА

* 1. Постановка задачи

Серверная часть интернет-магазина зоотоваров представляет собой систему, для обеспечения безопасности и удобства пользования. При проектировании такой системы следует учитывать сам масштаб системы и возможность ее масштабирования.

В данном контексте, требуется разработать следующие компоненты:

* управление товарами и каталогом: сервер интернет-магазина зоотоваров должен обеспечивать механизмы для управления товарами и каталогом. Это включает функциональность, позволяющую добавлять, редактировать и удалять товары, а также организовывать их в категории и подкатегории. Кроме того, система может включать функциональность для управления характеристиками товаров, такими как размер, цвет, вес и так далее;
* обработка заказов: сервер должен предоставлять механизмы для обработки заказов, включая функциональность оформления заказа, расчета стоимости, учета наличия товаров на складе, обработки платежей и управления статусами заказов (например, "в обработке", "отправлен", "доставлен");
* управление пользователями: сервер интернет-магазина должен обеспечивать возможность регистрации и аутентификации пользователей. Это включает функциональность для создания учетных записей пользователей, хранения их персональных данных, управления правами доступа и обеспечения безопасности данных пользователей;
* работа с корзиной покупок: сервер должен поддерживать функциональность работы с корзиной покупок, включая добавление товаров в корзину и обеспечение сохранения корзины между сеансами пользователя;

Обработка поисковых запросов: сервер должен обеспечивать поиск по товарам и каталогу, чтобы пользователи могли легко находить нужные товары. Это включает поиск, по ключевым словам, фильтрацию по категориям, а также сортировку результатов по различным параметрам, таким как цена, рейтинг и так далее. Так же одной из задач сервера является валидация данных и обеспечение безопасности их хранения. Это чрезвычайно важно выполнять на стороне сервера т.к. обычный пользователь не имеет доступа для подключения к нему и зачастую подключение к серверу выполняется на основе специальных ключей. При добавлении записей, требующих даты, будет записываться дата с сервера.

* 1. Проектирование архитектуры данных

Данные на уровне хранения будут представлены в виде реляционной модели таблиц и связей. Структура данных представлена ниже.

Таблица 2.1 – Пользователи (Users)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| user\_id | username | email | password\_hash | role\_id |
| (PK) int | varchar | varchar | varchar | (FK) int |

Таблица 2.2 – "Роли" (Roles)

|  |  |
| --- | --- |
| role\_id | role\_name |
| (PK) int | varchar |

Таблица 2.3 – "Товары" (Products)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| product\_id | product\_name | description | price | availability |
| (PK) int | varchar | varchar | float | int |

Таблица 2.4 – "Категории" (Categories)

|  |  |
| --- | --- |
| category\_id | category\_name |
| (PK) int | varchar |

Таблица 2.5 – "Заказы" (Orders)

|  |  |  |  |
| --- | --- | --- | --- |
| order\_id | user\_id | order\_date | status |
| (PK) int | (FK) int | datetime | varchar |

Таблица 2.6 – "Детали заказа" (OrderDetails)

|  |  |  |  |
| --- | --- | --- | --- |
| order\_detail\_id | order\_id | product\_id | quantity |
| (PK) int | (FK) int | (FK) int | int |

В качестве входных данных используются таблицы БД в JSON-формате.

Листинг 1 – Таблица "Пользователи" (Users) в json

{

"type": "object",

"properties": {

"user\_id": {"type": "integer"},

"username": {"type": "string"},

"email": {"type": "string"},

"password\_hash": {"type": "string"},

"role\_id": {"type": "integer"}

}

}

Листинг 2 – Таблица "Роли" (Roles) в json

{

"type": "object",

"properties": {

"role\_id": {"type": "integer"},

"role\_name": {"type": "string"}

}

}

Листинг 3 – Таблица "Товары" (Products) в json

{

"type": "object",

"properties": {

"product\_id": {"type": "integer"},

"product\_name": {"type": "string"},

"description": {"type": "string"},

"price": {"type": "number"},

"availability": {"type": "integer"}

}

}

Листинг 4 – Таблица "Категории" (Categories) в json

{

"type": "object",

"properties": {

"category\_id": {"type": "integer"},

"category\_name": {"type": "string"}

}

}

Листинг 5 – Таблица "Заказы" (Orders) в json

{

"type": "object",

"properties": {

"order\_id": {"type": "integer"},

"user\_id": {"type": "integer"},

"order\_date": {"type": "string"},

"status": {"type": "string"}

}

}

Листинг 6 – Таблица "Детали заказа" (OrderDetails) в json

{

"type": "object",

"properties": {

"order\_detail\_id": {"type": "integer"},

"order\_id": {"type": "integer"},

"product\_id": {"type": "integer"},

"quantity": {"type": "integer"}

}

}

Схема выходных данных представляет собой схему входных данных в теле ответа и так же код ответа от сервера.

* 1. Выбор и обоснование средств разработки

Для реализации программного средства были выбраны следующие инструменты.

Gin Gonic — это быстрый и минималистичный веб-фреймворк для языка программирования Go. Он предоставляет простой и эффективный интерфейс для построения RESTful API и обработки веб-запросов.

Вот основные характеристики и преимущества использования Gin Gonic:

* простота и производительность: Gin Gonic отличается лаконичным и эффективным синтаксисом, что позволяет быстро разрабатывать надежные веб-приложения с минимальным количеством кода;
* маршрутизация и промежуточное программное обеспечение: Gin Gonic предоставляет мощную систему маршрутизации, позволяющую легко определять и обрабатывать различные HTTP-методы и маршруты. Он также поддерживает использование промежуточного программного обеспечения для расширенной обработки запросов;
* middleware и расширяемость: Gin Gonic позволяет легко добавлять и настраивать промежуточное программное обеспечение, что делает его гибким и расширяемым. Есть большое сообщество, которое предоставляет множество готовых middleware-компонентов;
* высокая производительность: Gin Gonic основан на высокопроизводительной библиотеке httprouter, что обеспечивает быстрое обслуживание запросов.
* интеграция с другими инструментами: Gin Gonic хорошо интегрируется с другими популярными Go-инструментами, такими как ORM-библиотеки, средства логирования, системы мониторинга и тестирования;
* кроссплатформенность и безопасность: как и сам язык Go, Gin Gonic является кроссплатформенным решением, работающим на различных операционных системах. Он также включает в себя механизмы защиты от распространенных веб-уязвимостей.

В качестве СУБД была выбрана PostgreSQL.

PostgreSQL – это мощная объектно-реляционная система управления базами данных (СУБД) с открытым исходным кодом. Он предоставляет расширенные возможности для хранения, организации и манипулирования данными. PostgreSQL широко используется в веб-разработке благодаря своей надежности, производительности и поддержке продвинутых функций, таких как транзакции, индексы, схемы и многое другое.

* 1. Создание диаграмм

Диаграмма классов UML (Unified Modeling Language) является графическим представлением структуры и взаимодействия классов в системе. Она используется для моделирования объектно-ориентированных программных систем и позволяет лучше понять и визуализировать связи и взаимодействия между классами.

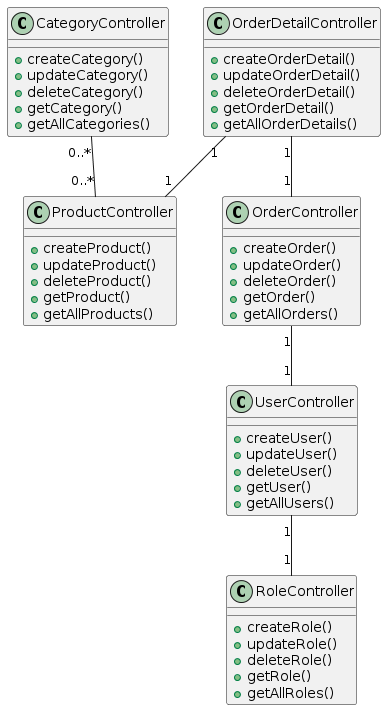


Рисунок 2.1 — Диаграмма классов

Диаграмма сущностей, также известная как диаграмма сущность-связь (Entity-Relationship Diagram, ER-диаграмма), является графическим инструментом моделирования, используемым для описания структуры данных в базе данных.

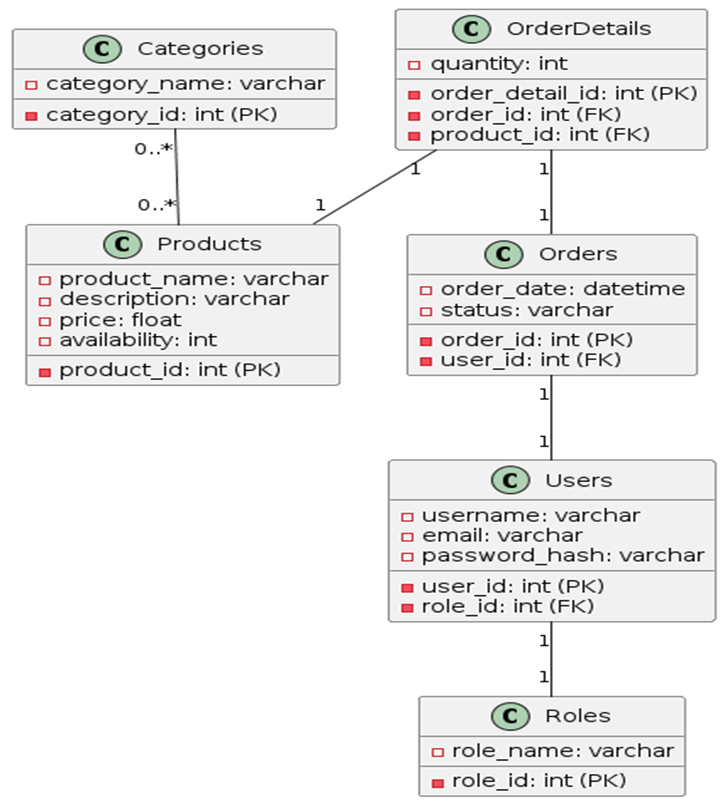


Рисунок 2.2 — Диаграмма сущностей

Диаграмма вариантов использования, также известная как диаграмма прецедентов (Use Case Diagram), является графическим инструментом моделирования, который позволяет описать функциональные возможности или сценарии использования системы с точки зрения взаимодействия акторов (пользователей) и системы.

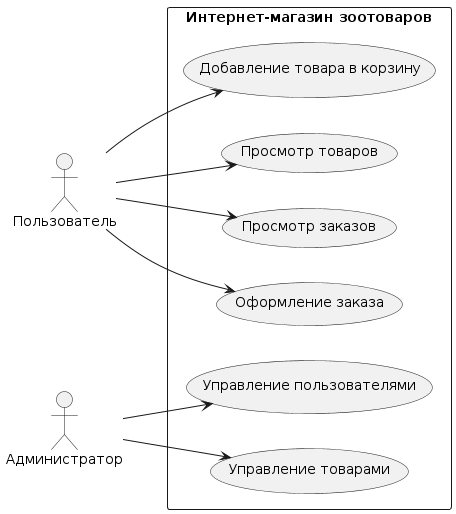


Рисунок 2.3 — Диаграммы вариантов использования

Диаграмма состояний является графическим инструментом моделирования, который позволяет описать различные состояния, переходы и события, в которых может находиться объект или система в течение своей жизненного цикла.

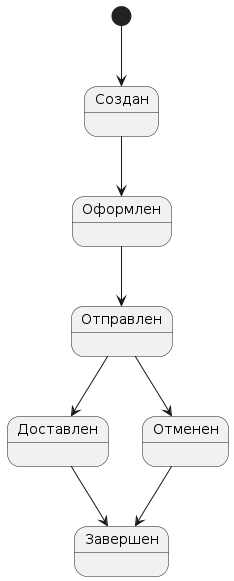


Рисунок 2.4 — Диаграмма состояний

1. РЕАЛИЗАЦИЯ. ОПИСАНИЕ АЛГОРИТМОВ РАБОТЫ ПРОГРАММНОГО СРЕДСТВА
   1. Проектирование программного средства

Для разработки серверной части приложения использовался язык golang с библиотекой gin-gonic, который предоставляет объекты и методы для взаимодействия клиентской части с серверной. Одним из преимуществ данной библиотеки принято считать скорость разработки и скорость работы api. Сам по себе язык изначально создавался для разработки приложений на базе микросервисной архитектуры, с чем golang и справляется идеально. Для реализации данной концепции в golang есть ключевое слово package, которая создает общую область видимости для всех файлов, с таким же именем пакета. Это, в первую очередь, удобство и избегание многих проблем, связанных с модульным программированием, таких как циркулярные импорты.

Так же gin-gonic предоставляет отличную систему логирования. Это так же дает ощутимые преимущества в разработке серверного приложения. ВыглядиАприт это следующим образом:

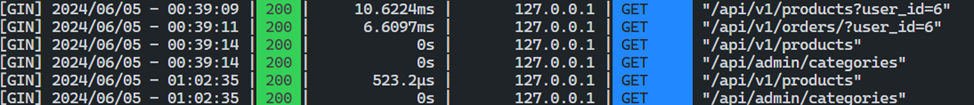


Рисунок 3.1 – Система логирования

Структура "лога" представляет собой:

* дата и время;
* статус-код ответа сервера;
* время, затраченное на выполнение запроса;
* домен;
* http метод запроса;
* endpoint.

Самая важная часть любого приложения – архитектура. Самый удобный и популярный вариант на сегодняшний день – MVC, но удовлетворяющий современным реалиям. В целом, остается все то же, но добавляется и много нового. В стандартном MVC всю бизнес логику содержат модели, а контроллеры служат лишь как связующее звено между представлением и моделью. Но сейчас все немного иначе. Самым популярным решением в современном мире является "луковичная архитектура Роберта Мартина, которая, в основе своей и использует MVC, но немного в другом виде. Данная архитектура и ляжет в основу приложения. Она, как и сказано в названии, представляет собой некую луковицу, которая, по мере развития приложения, обрастает новыми слоями.

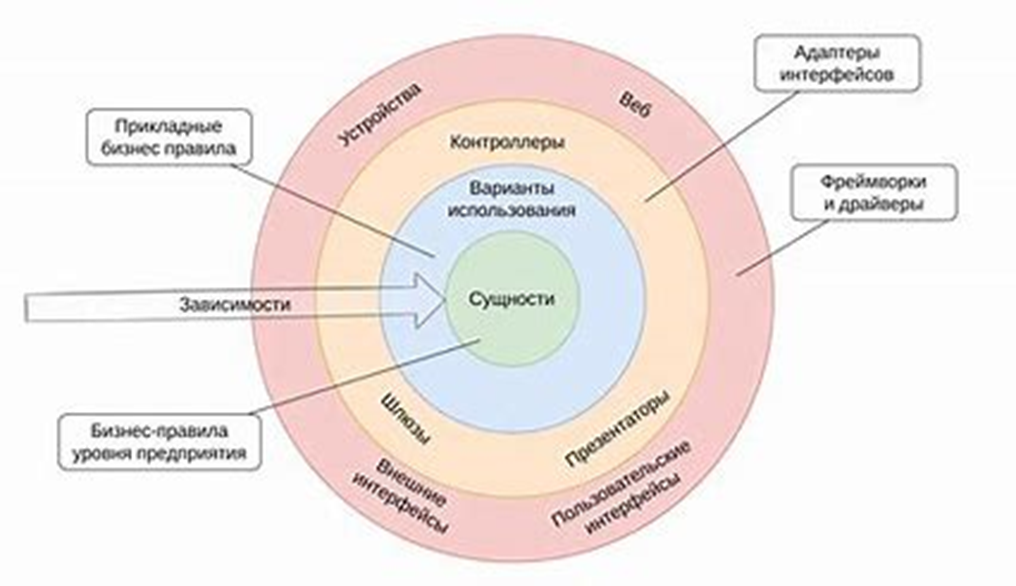


Рисунок 3.2 – Луковичная архитектура

Это представление отображает суть того, как будет проводиться внедрение зависимостей внутри приложения.

Рассмотрим на примере пользовательского приложения:

Листинг 3.1 – «структура App»

func NewApp(db \*db.DataBase) \*App {

// Repositories

productRepo := repositories.NewProductRepository(db)

authRepo := repositories.NewAuthRepository(db)

orderRepository := repositories.NewOrderRepository(db)

// Services

productService := services.NewProductService(productRepo)

authService := services.NewAuthService(authRepo)

orderService := services.NewOrderService(orderRepository)

//Controllers

productController:=controllers.NewProductController(productService)

authController := controllers.NewAuthController(authService)

orderController := controllers.NewOrderController(orderService)

//Routes

productRoutes := routes.NewProductRoutes(productController)

authRoutes := routes.NewAuthRoutes(authController)

orderRoutes := routes.NewOrderRoutes(orderController)

return &App{

ProductRoutes: productRoutes,

AuthRoutes: authRoutes,

OrderRoutes: orderRoutes,

}

}

type App struct {

ProductRoutes \*routes.ProductRoutes

AuthRoutes \*routes.AuthRoutes

OrderRoutes \*routes.OrderRoutes

}

На практике это будет работать следующим образом: запрос попадает в контроллер, там из него, если это необходимо, достается тело запроса и параметры запроса. В зависимости от входных параметров вызывается тот или иной сервис, который отвечает за обработку информации, валидацию и вызов репозитория, репозиторий, в свою очередь отвечает за запись данных в базу данных.

Пример запроса на сервер:

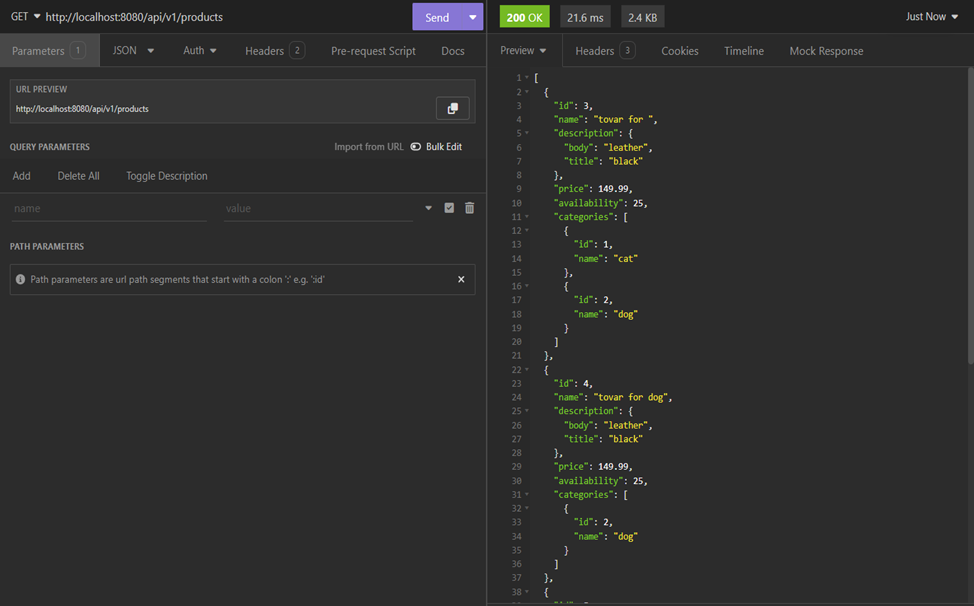


Рисунок 3.3 – Пример запроса

В данном примере рассматривается пример GET запроса на сервер для получения всех продуктов. Ответ пришел в виде JSON структуры.

* 1. Аутентификация и авторизация пользователей

Для обеспечения безопасности и контроля доступа к функционалу сайта, в backend-части реализована система аутентификации и авторизации пользователей. Эта система играет ключевую роль в защите данных и предотвращении несанкционированного доступа к критически важным частям приложения.

* регистрация пользователей. При регистрации нового пользователя, его личные данные (логин, пароль, email) сохраняются в базе данных. Для безопасного хранения паролей используется криптографическая библиотека bcrypt. Она позволяет хэшировать пароли с использованием соли, что делает их устойчивыми к взлому даже в случае компрометации базы данных;
* аутентификация. Когда пользователь пытается войти в систему, backend-часть реализует алгоритм проверки логина и пароля. Она сравнивает введенные данные с хранящимися в базе данных хэшированными паролями. Если введенные данные соответствуют записи в базе, аутентификация считается успешной.

Таким образом, реализованная в backend-части система аутентификации и авторизации пользователей обеспечивает надежный контроль доступа к функциональности сайта зоомагазина и защищает критически важные данные от несанкционированного использования.

* 1. Хранение данных в PostgreSQL

Для хранения всей необходимой информации о товарах и их категориях используется база данных PostgreSQL. PostgreSQL является популярной СУБД, которая хорошо подходит для этих целей.

В PostgreSQL данные хранятся в виде таблиц, объединенных в коллекции. Для сайта зоомагазина используются две основные коллекции:

* товары: хранит информацию о каждом товаре, включая название, описание, цену, категорию, изображения и другие атрибуты;
* категории: хранит информацию о различных категориях товаров, таких как корма, игрушки, аксессуары и прочее.

Для управления данными в этих коллекциях реализованы стандартные CRUD-операции (Create, Read, Update, Delete):

* create: добавление новых товаров или категорий в базу данных;
* read: получение информации о существующих товарах и категориях;
* update: изменение данных об уже существующих товарах или категориях;
* delete: удаление товаров или категорий из базы данных.

Эти операции позволяют администраторам сайта легко управлять каталогом товаров и структурой категорий.

Алгоритм поиска товаров использует фильтрацию по различным критериям, таким как название, категория, цена. Пользователи могут вводить ключевые слова для поиска, а система будет находить соответствующие товары в базе данных.

Таким образом, использование PostgreSQL в качестве базы данных для хранения информации о товарах и категориях обеспечивает гибкость, масштабируемость и эффективное управление данными в сайте зоомагазина.

* 1. Обработка заказа

Когда клиент оформляет заказ, в первую очередь backend-система проверяет, что пользователь авторизован на сайте. Это необходимо для идентификации клиента и привязки заказа к его учетной записи.

Когда клиент оформляет заказ, backend-система также обновляет информацию о наличии соответствующих товаров на складе в базе данных. Это необходимо для поддержания актуальных данных об остатках.

Система производит расчет общей стоимости заказа с учетом цен на товары, количества, применения скидок и промокодов. Это позволяет предоставить клиенту точную информацию о сумме, которую ему необходимо оплатить.

По мере выполнения заказа, backend-система обновляет его статус в базе данных (например, "новый", "в обработке", "доставлен", "отменен"). Это отражает текущее состояние заказа.

Каждый раз, когда статус заказа меняется, backend-система отправляет уведомление клиенту. Это может быть email или push-уведомление. Таким образом, клиент всегда в курсе текущего состояния его заказа.

Таким образом, backend-часть сайта играет ключевую роль в обработке заказов клиентов, обновлении информации о наличии товаров, расчете стоимости, управлении статусами заказов и взаимодействии с платежной системой.

1. ТЕСТИРОВАНИЕ
   1. Общая информация

Web-приложение для зоомагазина – это ключевое решение, которое оптимизирует все аспекты бизнеса. Он позволяет вести единую базу данных по товарам, услугам, клиентам и питомцам, обеспечивая полный учет и контроль. Сервер дает возможность эффективно управлять закупками, складскими остатками и продажами, избегая ошибок и оперативно реагируя на изменение спроса. Кроме того, он служит платформой для организации онлайн-продаж и удобного взаимодействия с клиентами через веб-портал или мобильное приложение. Сервер также помогает собирать аналитику по предпочтениям покупателей и другим важным метрикам, что позволяет принимать обоснованные стратегические решения. В целом, грамотное использование серверного решения является ключом к оптимизации бизнес-процессов и повышению эффективности зоомагазина.

Цель проведения тестирования разработанного дипломного проекта является подтверждение качественных характеристик, которые включают:

Модель качества программного продукта включает:

* функциональная пригодность;
* уровень производительности;
* надёжность;
* сопровождаемость;
* переносимость (мобильность).

В соответствии с темой дипломного проекта необходимо разработать план тестирования (п. 4.1.1.).

### Тест-план

Тестирование сервера для зоомагазина, версия 1.0.0.

### Введение

Цель данного документа – создание тест плана Web-приложения. Представленный тест-план оценивает подходы к тестированию данного продукта. Основное назначение данного тест-плана является усвоение навыков проверки качества созданного программного продукта.

### Объекты тестирования

Объектами тестирования являются:

1. функциональные;
2. нефункциональные:

* уровень производительности;
* надежность;
* переносимость (мобильность);
* сопровождаемость

### Что будет тестироваться

По методике чек-листа тестируем уровень производительности, надежность, переносимость (мобильность) и сопровождаемость. Функциональная пригодность будет протестирована по методике тест-кейсов.

### Что не будет тестироваться

Тестирование не будет затрагивать справочный материал, так как в данной версии программы он находится в постоянной разработке, а также:

* совместимость;
* удобство использования (юзабилити);
* защищенность.

### Подход к тестированию

По доступу к исходному коду – White Box.

По запуску кода – динамическое тестирование.

По степени автоматизации – ручное.

По объекту тестирования:

* функциональное;
* нефункциональное.

По требованиям

* позитивное.

По степени подготовленности - интуитивное тестирование.

### Критерии прохождения тестов

Все тест-кейсы, имеющие высокий приоритет, закрыты с результатом «пройдено».

Тестовое покрытие проверено и являются достаточным, где критерий достаточности составляет не менее 99% покрытия требований тестами.

### Результаты проведения тестирования

Тест-план, чек-лист, тест-кейсы, отчёт о дефектах.

### Задачи для проведения тестирования

Таблица 4.1 – Задачи для проведения тестирования

|  |  |
| --- | --- |
| Задача | Расположение |
| Написание тест-плана | Создание тест плана, обязанности |
| Написание чек-листа | Объекты тестирования, обязанности |
| Написание тест-кейсов | Объекты тестирования, обязанности |
| Проведение тестирования и оценка результатов | Подход к тестированию, обязанности |
| Создание отчетов о результатах тестирования | Результаты проведения тестирования |

### Технические требования

Тестирование Web-API будет происходить на следующей операционной системе – Windows 10.

### Обязанности

Обязанности по проведению тестирования возложены на разработчика ПП Титкова Романа Николаевича.

### Необходимые компетенции и тренинги

Для выполнения поставленных у задач необходимо обладать следующими знаниями и умениями:

* знания и умения применения на практике стандарта IEEE-829;
* знания и умения применить на практике основных техник тест дизайна;
* знание различных типов тестирования, в том числе функционального и нефункционального.

### Расписание / срок сдачи

Срок окончания всех работ и сдачи проекта – 10/06/2024 09:00:00

### Риски

Возможные риски во время тестирования:

* недостаточное количество знаний для тестирования приложения в установленные сроки;
* задержки в сроках выполнения работ, вызванные другими обстоятельствами.

### Утверждение

|  |  |
| --- | --- |
| Тестирование выполнил учащаяся группы 9ПЗ-55 Титков Роман Николаевич | Дата «\_\_\_» \_\_\_\_\_\_\_\_\_\_ 2024  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись) |

## Чек-листы

Тестирование уровня производительности. Рассмотрим следующие сценарии:

* API возвращает нужные данные в пределах 100 мс при нагрузке в 10 запросов в секунду;
* API обрабатывает до 10 запросов в секунду без значительного ухудшения производительности;
* Среднее время ответа API составляет 50 мс при нагрузке в 10 запросов в секунду.

Тестирование надёжности. Рассмотрим следующие сценарии:

* API корректно обрабатывает ввод некорректных данных или возникновении ошибок на стороне сервера;
* API демонстрирует стабильную работу при длительной нагрузке, без сбоев и ошибок;
* API может быстро восстановиться после сбоев или ошибок, без потери данных или функциональности.

Тестирование сопровождаемости. Рассмотрим следующие сценарии:

* API имеет модульную структуру, что облегчает внесение изменений и исправление ошибок;
* API позволяет повторно использовать существующий код, что уменьшает затраты и увеличивает производительность;
* API может быть легко изменен без ухудшения его качества и функциональности;
* API имеет простой и понятный код, что облегчает поиск и исправление ошибок.

Тестирование переносимости. Рассмотрим следующие сценарии:

* API работает корректно в различных браузерах различных версий;
* API работает корректно на различных операционных системах и устройствах;
* API быстро отвечает на запросы, даже при медленном интернет-соединении.

## 4.3. Тест-кейсы программного средства

Таблица 4.2 – Тест-кейсы приложения

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ID | Описание | Шаги, входные данные | Ожидаемый результат | Фактический результат | Статус |
| 1 | 2 | 3 | 4 | 5 | 6 |
| 4.2.1 | Запуск программы | Запустить исполняемый файл программы | Проект запустился и доступен по ссылке http://0.0.0.0:8080/api/ | Ожидаемый результат подтвержден | Пройден успешно |
| 4.2.2 | Получения списка всех товаров | На странице браузера в адресной строке ввести http://0.0.0.0:8080/api/v1/products | Сервер вернет ответ в виде  массива товаров, шаблон товара - листинг 8 | Ожидаемый результат подтвержден | Пройден успешно |
| 4.2.3 | Получение товара по ID | На странице браузера в адресной строке ввести http://0.0.0.0:8080/api/v1/products/6 | Возврат товара с заданным id. Шаблон – листинг 8 | Ожидаемый результат подтвержден | Пройден успешно |
| 4.2.4 | Получение товара с несуществующим id | На странице браузера в адресной строке ввести http://0.0.0.0:8080/api/v1/products/99 | Возврат ошибки {«error»: «not found»} и статус код 404 | Ожидаемый результат подтвержден | Пройден успешно |
| 4.2.6 | Создание нового товара на части ад-министратора | Открыть приложение для тестирования запросов. Перейти по ссылке http://0.0.0.0: 8080/api/admin/products/, с указанием того, что этот запрос - POST, ввести данные как в структуре листинга 8 и нажать кнопку «Send» | Сервер вернет статус код 200 и добавит товар в базу данных | Ожидаемый результат подтвержден | Пройден успешно |

Продолжение таблицы 4.2

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 5 | 6 |
| 4.2.7 | Получение всех категорий | На странице браузера в адресной строке ввести <http://0.0.0.0:8080/api/v1/categories> | Сервер вернет ответ в виде JSON, отобра-жаемом на странице браузера в виде массива структуры, приведенной в листинге 9 | Ожидаемый результат подтвержден | Пройден успешно |
| 4.2.8 | Создание заказа | Открыть приложение для тестирования запросов. В адресной строке ввести [http:// 0.0.0.0](http://0.0.0.0):8080/api/v1/orders/ с указанием, что этот запрос - POST, ввести данные структуры, приведен-ной в листинге 10 и нажать кнопку «Send» | Добавление товара в базу данных и возврат клиенту ответа с статус кодом 200 | Ожидаемый результат подтвержден | Пройден успешно |
| 4.2.9 | Регистрация пользователя | Открыть приложение для тестирования запросов. Перейти по ссылке http://0.0.0.0:8080/api/register  С указанием того, что этот запрос - POST, ввести данные, которые соответствую листингу 8 и нажать кнопку «Send» | Добавление пользователя в базу данных нового пользователя | Ожидаемый результат подтвержден | Пройден успешно |

Продолжение таблицы 4.2

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 5 | 6 |
| 4.2.10 | Авторизация пользователя | Открыть приложение для тестирования запросов. В адресной строке ввести http://0.0.0.0:8080/api/auth  С указанием того, что этот запрос - POST, ввести данные, которые соответствуют листингу 8 и нажать кнопку «Send» | Возврат клиенту статус кода 200 и текущего пользователя | Ожидаемый результат подтвержден | Пройден успешно |
| 4.2.11 | Добавление товара в избранное | Открыть приложение для тестирования запросов. адресной строке ввести http://0.0.0.0:8080/api/v1/products  С указанием того, что этот запрос - POST, ввести желаемые данные и нажать кнопку «Send» | Добавление товара в запись об избранном и возврат пользователю статус код 200 | Ожидаемый результат подтвержден | Пройден успешно |
| 4.2.12 | Создание нового товара на части администратора | Открыть приложение для тестирования запросов. Перейти по ссылке http://0.0.0.0:8080/api/admin/products/, с указанием того, что этот запрос - POST, ввести данные, которые не соответствуют структуре листинга 8 и нажать кнопку «Send» | Сервер вернет {«error»: «»} и статус код 400 | Ожидаемый результат подтвержден | Пройден успешно |

Листинг 7 – Таблица "Пользователи" (Users) в json

{

"type": "object",

"properties": {

"user\_id": {"type": "integer"},

"username": {"type": "string"},

"email": {"type": "string"},

"password\_hash": {"type": "string"},

"role\_id": {"type": "integer"}

}

}

Листинг 8 – Таблица "Товары" (Products) в json

{

"type": "object",

"properties": {

"product\_id": {"type": "integer"},

"product\_name": {"type": "string"},

"description": {"type": "string"},

"price": {"type": "number"},

"availability": {"type": "integer"}

}

}

Листинг 9 – Таблица "Категории" (Categories) в json

{

"type": "object",

"properties": {

"category\_id": {"type": "integer"},

"category\_name": {"type": "string"}

}

}

Листинг 10 – Таблица "Заказы" (Orders) в json

{

"type": "object",

"properties": {

"order\_id": {"type": "integer"},

"user\_id": {"type": "integer"},

"order\_date": {"type": "string"},

"status": {"type": "string"}

}

}

## 

## 4.4. Вывод о результатах тестирования

При проведении тестирования Web-API на тему «Разработка серверной части сайта зоомагазина» дефектов не обнаружено, программный продукт готов к эксплуатации.

1. ЭКОНОМИЧЕСКИЙ РАЗДЕЛ

## Исходная информация

Качество и глубина проработки экономических вопросов в дипломном проекте определяется тем, насколько полно собраны данные, необходимые для экономического анализа и расчетов. Эту работу необходимо выполнять в период прохождения преддипломной практики в соответствии с методическими указаниями, содержащимися в программе преддипломной практики.

Исходная информация для расчета экономического раздела получена в ОАО "InTechHub" указана в таблице 5.1.

Таблица 5.1 - Исходная информация

|  |  |  |  |
| --- | --- | --- | --- |
| Показатель | Условное обозначение | Единицы измерения | Значение, рублей |
| Стоимость компьютера | К | Рублей | 1500 |
| Мощность, потребляемая компьютером | МК | КВт/час | 0,4 |
| Полезный фонд времени работы компьютера за год | FK | Час | 2000 |
| Стоимость принтера | П | Рублей | 450 |
| Мощность, потребляемая принтером | МП | КВт/час | 0,2 |
| Полезный фонд времени работы принтера за год | FП | Час | 1500 |
| Стоимость 1 кВт/час (с учетом НДС) | С | Рублей | 0,17 |
| Норма амортизации для оборудования годовая | На | Проценты | 15 |
| Часовая тарифная ставка 13 разряда | ЧТС13 | Рублей | 10,5 |

Руководитель дипломного проекта \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись)

* 1. Определение себестоимости программного обеспечения (ПО), как базы для формирования цены

В современных рыночных экономических условиях ПО выступает преимущественно в виде продукции научно-технических организаций, представляющей собой функционально завершенное и имеющее товарный вид ПО, реализуемое покупателям по рыночным отпускным ценам, установленным в результате анализа рыночных условий и переговоров с заказчиком и возможностей тиражирования. Все завершенные разработки ПО являются научно-технической продукцией.

Широкое применение вычислительной техники требует постоянного обновления и совершенствования ПО. Выбор эффективных проектов ПО связан с их экономической оценкой и расчетом экономического эффекта, который может определяться как у разработчика, так и у пользователя.

У разработчика экономический эффект выступает в виде чистой прибыли, остающейся в распоряжении предприятия от реализации ПО, а у пользователя – в виде экономии трудовых, материальных и финансовых ресурсов, получаемой за счет:

* снижения трудоемкости расчетов и алгоритмизации программирования и отладки программ, за счет использования ПО в процессе разработки автоматизированных систем обработки данных;
* сокращения расходов на оплату машинного времени и других ресурсов на отладку программ;
* снижения расходов на материалы;
* ускорение ввода в эксплуатацию новых систем;
* улучшения показателей основной деятельности предприятий в результате использования ПО.

Стоимостная оценка ПО у разработчиков предполагает составление сметы затрат, которая включает следующие статьи:

* затраты на материалы;
* стоимость машинного времени;
* основная заработная плата;
* дополнительная заработная плата;
* отчисление в фонд социальной защиты населения;
* обязательное страхование от несчастных случаев на производстве и профессиональных заболеваний;
* прочие затраты;
* расходы на сопровождение и адаптацию.

На основании сметы затрат рассчитывается себестоимость ПО. Расчет ведется в следующей последовательности:

* + 1. Расчет затрат на материалы

По статье «Материалы» отражаются расходы на магнитные носители, бумагу, красящие ленты и другие материалы, необходимые для разработки ПО. Расчет затрат на материалы (М), необходимые для разработки программного обеспечения (ПО) осуществляется по нормативу (Нр) на 100 команд (принимаем равным 1 руб.), с учетом общего объема команд (VкПО ) (берется учащимся по факту, исходя из полученного им количества команд при разработке ПО).

Определение затрат на материалы осуществляется по формуле, в рублях:

М= (5.1)

М=

* + 1. Обоснование трудоемкости разработки ПО

Обоснование трудоемкости разработки ПО ведется следующим образом: на основании опытно-статистического (суммарного) метода, в целом на все разработку, исходя из того, что ее выполняет один программист 2-ой категории. Данные о трудоемкости разработки, определенные опытно-статистическим путём, подтверждаю, трудоемкость – 171 час.

Общий объём программного обеспечения – количество команд (VkПО) – 146.

Руководитель дипломного проекта \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись)

* + 1. Расчет стоимости машинного времени

Определение стоимости потребляемой электроэнергии осуществляется по формуле, в рублях:

Сэл= (МК×FK + МП ×FП) × С × Квр×Кс, (5.2)

где Квр - коэффициент, учитывающий использование, повремени (Квр=0,8)

Кс - коэффициент, учитывающий потери в сети (Kc=1,05)

Сэл= (0,4×2000 + 0,2 ×1500**)** × 0,17 × 0,8 ×1,05= 157,08 (руб.)

Определение суммы амортизационных отчислений осуществляется по формуле, в рублях:

Сао= (К+П) ×На/100, (5.3)

Сао= (1500+450) ×15/100=292,5 (руб.)

Результаты расчетов сводим в таблицу 2.

Таблица 5.2 - Определение стоимости электроэнергии и амортизационных отчислений

|  |  |  |  |
| --- | --- | --- | --- |
|  | Условное | Значение, рублей | |
| Показатель | обозначение | На год  (ЗМг) | На 1  машино-час (СМч) |
| Стоимость потребляемой электроэнергии | Сэл | 157,08 | 0,08 |
| Сумма ежегодных амортизационных отчислений | Сао | 292,5 | 0,15 |
| Итого: | ЗМг | 449,58 | 0,23 |

Расчет затрат на 1 машино-час ведется исходя из стоимости машинного времени на год (графа 3 таблицы 1) деленный на полезный фонд времени работы компьютера (FK.) (из таблицы «Исходная информация»).

Определение затрат на 1 машино-час осуществляется по формуле, в рублях:

СМч=ЗМг/FK, (5.4)

СМч=449,58 / 2000 = 0,23 (руб.)

Определение стоимости машинного времени осуществляется по формуле, в рублях:

* + 1. СМвр=Тм×СМч, (5.5)

где Тм - машинное время работы в расчёте на программу, час. (70% от трудоемкости разработки программного обеспечения)

Смги - стоимость одного машино-часа, руб. (итог графы 4 таблицы 2)

СМвр= 91 × 0,23 = 20,93 (руб.)

* + 1. Расчет затрат на оплату труда

Расчёт основной заработной платы ведётся исходя из трудоемкости программного обеспечения.

Определение прямой заработной платы осуществляется по формуле, в рублях:

Зпр=Тi×ЧТС13, (5.6)

где Тi – трудоемкость программного обеспечения, ч.

ЧТС13 – часовая тарифная ставка 13 разряда, руб.

Зпр=130×10,5 = 1365 (руб.)

Премия определяется как 30 % от основной заработной платы.

Определение премий осуществляется по формуле, в рублях:

П = Зпр×30/100, (5.7)

П = 1365×30/100 =409,5 (руб.)

Дополнительная заработная плата (Зд) включает выплаты, предусмотренные законодательством о труде (оплата отпусков, льготных часов, времени выполнения государственных обязанностей и других выплат, не связанных с основной деятельностью исполнителей) и определяется по нормативу в процентах к основной заработной плате.

Определение дополнительной заработной платы осуществляется по формуле, в рублях:

, (5.8)

где Зос – основная заработная плата;

ЗД – дополнительная заработная плата на конкретное ПО в руб.

НД – норматив дополнительной заработной платы принимается 10% от основной заработной платы.

= 177,45 (рублей)

Отчисления в фонд социальной защиты населения (3C3) определяются в соответствии с действующими законодательными актами по нормативу в процентном отношении к фонду основной и дополнительной зарплаты (Зд).

Определение отчислений в фонд социальной защиты населения осуществляется по формуле, в рублях:



(5.9)

где Нсз – норматив отчислений в фонд социальной защиты населения (процент действующий норматив на момент расчета).

Обязательное страхование от несчастных случаев на производстве и профессиональных заболеваний (Зстр) берётся по нормативу, установленному в целом по организации (можно принять Нстр– 0,6 %).

= 663,66 (рублей)

Определение отчислений на обязательное страхование от несчастных случаев на производстве и профессиональных заболеваний осуществляется по формуле, в рублях:



(5.10)

= 11,71 рубля

* + 1. Расчет полной себестоимости программного обеспечения

Прочие затраты (Пз) включают расходы, связанные с необходимостью содержания аппарата управления, организационные расходы и расходы на общехозяйственные нужды. Принимаются (Нпз) в процентном отношении к основной заработной плате исполнителей. Норматив устанавливается в целом по организации (Нпз принимаем 99%).

Определение прочих затрат осуществляется по формуле, в рублях:

, (5.11)

= 1756,76

Полная себестоимость без учета расходов на сопровождение и адаптацию (Сп) рассчитывается как сумма затрат на материалы, стоимости машинного времени, затрат на основную, дополнительную заработную плату, стоимости отчислений в фонд социальной защиты населения и на обязательное страхование от несчастных случаев на производстве и профессиональных заболеваний, прочих затрат.

Определение полной себестоимости осуществляется по формуле, в рублях:

Сп=М+СМвр+Зос+Зд+Зсз+Зстр+ПЗ (5.12)

Сп=+20,93++177,45 +663,66+11,71 +1756,76 =4406,73 (руб)

Кроме того, организация-разработчик осуществляет затраты на сопровождение и адаптацию ПО (РСА), которые определяются по нормативу НРСА (принимаем 4%), от полной себестоимости без учёта расходов на адаптацию (Сп).

Определение затрат на сопровождение и адаптацию ПО осуществляется по формуле, в рублях:



(5.13)

где Сп – полная себестоимость ПО без учёта расходов на сопровождение и адаптацию, руб.

Определение общей суммы расходов на разработку (с затратами на сопровождение и адаптацию) (Спл) осуществляется по формуле, в рублях:



, (5.14)

Спл = 4406,73 + = 4582,99 (рублей)

Результаты всех расчётов сводим в таблицу 5.3.

Таблица 5.3 – Определение полной себестоимости программного обеспечения

|  |  |  |
| --- | --- | --- |
| Наименование статей затрат | Сумма, руб. | Обоснование расчета |
| 1. Затраты на материалы | 1,72 | пункт 1.1. |
| 2. Стоимость машинного времени | 20,93 | пункт 1.3. |
| 3. Основная заработная плата | 1774,5 | пункт 1.4 |
| 4.Дополнительная заработная плата | 177,45 | 10% от ст.3 (пункт 1.4) |
| 5. Итого, фонд оплаты труда (ФОТ) | 1951,95 | (ст.3 + ст.4) (пункт 1.4) |
| 6. Отчисление в фонд социальной защиты населения | 663,66 | 34% от ст.5 (пункт 1.4) |
| 7. Обязательное страхование от несчастных случаев на производстве и профессиональных заболеваний | 11,71 | 0,6 % от ст.5 (пункт 1.4) |
| 8. Итого: отчисления на социальные нужды | 675,37 | (ст. 6 + ст.7) (пункт 1.4) |
| 9. Прочие затраты | 1756,76 | 99% от ст.3 (пункт 1.5) |
| 10. Полная себестоимость без учёта расходов на сопровождение и адаптацию | 4406,75 | (ст.1+ст.2+ст.5+ст.8+ст.9) таблицы 3 (пункт 1.5) |
| 11. Расходы на сопровождение и адаптацию. | 176,26 | 4% от ст.10 таблицы 3 (пункт 1.5) |
| 12. Полная себестоимость с учётом расходов на сопровождение и адаптацию | 4603,3 | (ст.10 + ст.11) таблицы 3 (пункт 1.5) |

* 1. Выбор схемы монетизации программного продукта

Рассмотрим основных конкурентов с указанием их функций, стоимости, достоинств и недостатков:

1. Magento 2 Commerce:

* функции: полнофункциональная eCommerce платформа с широкими возможностями для зоомагазинов;
* стоимость: лицензия - от 75 000 BYN/год, дополнительные расходы на разработку и интеграцию;
* достоинства: масштабируемость, гибкость, обширная экосистема дополнений;
* недостатки: высокая начальная стоимость, сложность внедрения, требует опытной команды разработчиков.

1. WooCommerce:

* функции: плагин для WordPress, предоставляющий базовые eCommerce функции для зоомагазинов;
* стоимость: бесплатный плагин, дополнительные расходы на хостинг, домен и разработку;
* достоинства: низкая начальная стоимость, простота интеграции с WordPress;
* недостатки: ограниченные возможности по сравнению с профессиональными eCommerce решениями, зависимость от WordPress.

1. Shopify:

* функции: облачная eCommerce платформа с широким набором инструментов для зоомагазинов;
* стоимость: от 100 BYN/мес за базовый тариф, дополнительные расходы на домен, дизайн и интеграции;
* достоинства: простота использования, широкий выбор шаблонов и приложений, надежность облачной инфраструктуры;
* недостатки: ограниченные возможности кастомизации, постоянные ежемесячные расходы.

Основными конкурентными преимуществами разрабатываемого backend-сайта зоомагазина являются:

* подробные описания товаров с характеристиками и фотографиями.
* возможность интеграции с другими системами и сервисами по мере роста и развития проекта.
* интеграция с системами учета и автоматизация складских операций.
* надежная защита от киберугроз и утечки данных.
* отказоустойчивость и высокая доступность системы для бесперебойной работы.

Для монетизации backend-сайта зоомагазина могут быть использованы следующие схемы:

* комиссия за транзакции: взимание процентной комиссии с каждой продажи, совершенной через сайт.
* подписка: предложение платной подписки для доступа к расширенным функциям.

Реклама и спонсорство: размещение рекламы продуктов и услуг партнеров на сайте.

Комиссия за транзакции может быть одним из наиболее оптимальных вариантов монетизации backend-сайта зоомагазина. Комиссионная модель широко распространена среди онлайн-площадок, маркетплейсов и других подобных бизнес-моделей. Комиссия за транзакции является сильным выбором для монетизации backend-сайта зоомагазина.

Преимущества комиссионной модели:

* комиссионная модель напрямую связывает доход с выручкой от продаж, что обеспечивает стабильный рост доходов по мере роста бизнеса.
* комиссионная модель легко масштабируется, так как затраты на обслуживание дополнительных транзакций относительно невелики, позволяя расти и обслуживать все больший объем продаж без значительного увеличения издержек.
* комиссионная схема не требует сложного ценообразования и управления различными тарифными планами, что упрощает администрирование и делает модель более прозрачной для клиентов.

Недостатки комиссионной модели:

* доход будет напрямую зависеть от объема продаж, что создает риски в периоды экономической нестабильности или сезонных колебаний спроса.
* для получения значимого дохода от комиссий потребуется обеспечить высокую конверсию посетителей в покупателей, что может потребовать дополнительных инвестиций в маркетинг и оптимизацию пользовательского опыта.
* некоторые клиенты могут негативно воспринимать дополнительные комиссии, что может повлиять на их лояльность и готовность совершать покупки.
* если конкуренты предлагают более привлекательные условия без комиссий, это может стать препятствием для привлечения клиентов.

Такой подход к монетизации позволит максимально эффективно монетизировать backend-сайта зоомагазина, обеспечивая гибкость, устойчивость доходов и высокую ценность для клиентов.

# ЗАКЛЮЧЕНИЕ

В ходе работы над дипломным проектом на тему "Разработка backend-сайта зоомагазина" были реализованы следующие основные результаты:

* проведен тщательный анализ предметной области, включая изучение аналогов разрабатываемого программного средства, определение достоинств и недостатков уже существующих, определение необходимого функционала для backend-части сайта;
* разработана полная проектная документация;
* выбраны и применены современные технологии и инструменты backend-разработки, такие как Gin Golang, система управления базами данных PostgreSQL, система контроля версий Git. Это позволило обеспечить высокую производительность, масштабируемость и надежность создаваемого веб-ресурса;
* реализован функционал, включая модули управления каталогом товаров, заказами клиентов, пользовательскими профилями, системой скидок и акций, администрирования сайта и другие важные компоненты;
* проведено тестирование backend-части;
* разработана стратегия интеграции backend- и frontend-частей веб-сайта, обеспечивающая эффективное взаимодействие между ними;

Таким образом, в результате выполнения дипломного проекта был создан высокопроизводительный, масштабируемый и надежный backend-сайт зоомагазина, полностью отвечающий всем требованиям и ожиданиям заказчика. Реализованное решение позволит значительно повысить эффективность управления бизнес-процессами зоомагазина и улучшить взаимодействие с клиентами.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Go [Электронный ресурс]. Режим доступа: <https://go.dev/>
2. Учебник SQL [Электронный ресурс]. Режим доступа:

<https://sqlzoo.net/wiki/SQL_Tutorial>

1. Справочник SQL-команд [Электронный ресурс]. Режим доступа:

<https://www.postgresql.org/docs/current/sql-commands.html>.

1. Gin-Gonic [Электронный ресурс]. Режим доступа: https://gin-gonic.com/
2. Архитектура сервера [Электронный ресурс]. Режим доступа: https://www.techopedia.com/definition/30262/server-architecture
3. Обработка ошибок [Электронный ресурс]. Режим доступа: https://boldlygo.tech/posts/2024-01-08-error-handling/

# ПРИЛОЖЕНИЕ 1

ИСХОДНЫЙ ТЕКСТ ПРОГРАММНОГО СРЕДСТВА

package repositories

import (

"admin/web-server/admin/models"

"admin/web-server/db"

"encoding/json"

)

type ProductRepository struct {

db \*db.DataBase

}

func NewProductRepository(database \*db.DataBase) \*ProductRepository {

return &ProductRepository{db: database}

}

func (pr \*ProductRepository) GetAllProducts() ([]models.Product, error) {

query := `

SELECT

p.id, p.name, p.description, p.price, p.availability,

c.id AS category\_id, c.name AS category\_name

FROM product p

LEFT JOIN product\_categories pc ON p.id = pc.product\_id

LEFT JOIN categories c ON pc.category\_id = c.id

`

rows, err := pr.db.Query(query)

if err != nil {

return nil, err

}

defer rows.Close()

var productsAsJSON = make(map[int]models.Product)

for rows.Next() {

var product models.ProductFromDB

var category models.Category

if err := rows.Scan(

&product.ID,

&product.Name,

&product.Description,

&product.Price,

&product.Availability,

&category.ID,

&category.Name,

); err != nil {

return nil, err

}

if \_, ok := productsAsJSON[int(product.ID)]; !ok {

productsAsJSON[int(product.ID)] = models.Product{

ID: product.ID,

Name: product.Name,

Description: models.JSONDescription{},

Price: product.Price,

Availability: product.Availability,

Categories: &[]models.Category{},

}

}

entry := productsAsJSON[int(product.ID)]

json.Unmarshal(product.Description, &entry.Description)

if category.ID != nil && \*category.ID != 0 {

\*entry.Categories = append(\*entry.Categories, category)

}

productsAsJSON[int(product.ID)] = entry

}

if err := rows.Err(); err != nil {

return nil, err

}

products := make([]models.Product, 0, len(productsAsJSON))

for \_, product := range productsAsJSON {

products = append(products, product)

}

return products, nil

}

func (pr \*ProductRepository) CreateProduct(product models.CreateProduct) error {

description, err := json.Marshal(product.Description)

if err != nil {

return err

}

stmt, err := pr.db.Prepare("INSERT INTO product(name, description, price, availability, image) VALUES ($1, $2, $3, $4, $5) RETURNING id")

if err != nil {

return err

}

defer stmt.Close()

var productID int64

err = stmt.QueryRow(product.Name, description, product.Price, product.Availability, product.Image.Filename).Scan(&productID)

if err != nil {

return err

}

for \_, cat := range product.Categories {

stmt, err = pr.db.Prepare("INSERT INTO product\_categories(product\_id, category\_id) VALUES ($1, $2)")

if err != nil {

return err

}

defer stmt.Close()

\_, err = stmt.Exec(productID, cat.ID)

if err != nil {

return err

}

}

return nil

}

func (pr \*ProductRepository) GetProductById(id string) (models.Product, error) {

var productFromDB models.ProductFromDB

var product models.Product

err := pr.db.QueryRow("SELECT id, name, description, price, availability FROM product WHERE id = $1", id).

Scan(&product.ID, &product.Name, &productFromDB.Description, &product.Price, &product.Availability)

if err != nil {

return models.Product{}, err

}

json.Unmarshal(productFromDB.Description, &product.Description)

categories, err := pr.GetCategoriesOfProduct(product.ID)

if err != nil {

return models.Product{}, err

}

if product.Categories == nil {

product.Categories = &categories

} else {

\*product.Categories = categories

}

return product, nil

}

func (pr \*ProductRepository) GetProductsByCategory(category models.Category) ([]models.Product, error) {

tx, err := pr.db.Begin()

if err != nil {

return nil, err

}

stmt, err := tx.Prepare(`

SELECT p.id, p.name, p.description, p.price, p.availability

FROM product p

JOIN product\_categories pc ON p.id = pc.product\_id

JOIN categories c ON pc.categories\_id = c.id

WHERE c.name = $1

`)

if err != nil {

tx.Rollback()

return nil, err

}

defer stmt.Close()

rows, err := stmt.Query(category.Name)

if err != nil {

tx.Rollback()

return nil, err

}

defer rows.Close()

var products []models.Product

for rows.Next() {

var product models.Product

err = rows.Scan(

&product.ID,

&product.Name,

&product.Description,

&product.Price,

&product.Availability,

)

if err != nil {

tx.Rollback()

return nil, err

}

products = append(products, product)

}

err = tx.Commit()

if err != nil {

tx.Rollback()

return nil, err

}

return products, nil

}

func (pr \*ProductRepository) GetCategoriesOfProduct(productID uint) ([]models.Category, error) {

var categories []models.Category

rows, err := pr.db.Query("SELECT c.id, c.name FROM categories c INNER JOIN product\_categories pc ON c.id = pc.category\_id WHERE pc.product\_id = $1", productID)

if err != nil {

return nil, err

}

defer rows.Close()

for rows.Next() {

var category models.Category

if err := rows.Scan(&category.ID, &category.Name); err != nil {

return nil, err

}

categories = append(categories, category)

}

if err := rows.Err(); err != nil {

return nil, err

}

return categories, nil

}

func (pr \*ProductRepository) DeleteProduct(id int) error {

deleteRelationQuery := `delete from user\_product where product\_id=$1;`

if \_, err := pr.db.Exec(deleteRelationQuery, id); err != nil {

return err

}

query := `

delete from product where id=$1;

`

if \_, err := pr.db.Exec(query, id); err != nil {

return err

}

return nil

}

func (pr \*ProductRepository) UpdateProduct(newProduct models.UpdateProduct) error {

query := `UPDATE product

SET name = $1,

description = $2,

price = $3,

availability = $4

WHERE id = $5`

description, err := json.Marshal(newProduct.Description)

if err != nil {

return err

}

\_, err = pr.db.Exec(query, newProduct.Name, description, newProduct.Price, newProduct.Availability, newProduct.ID)

if err != nil {

return err

}

err = pr.updateProductCategories(newProduct.ID, newProduct.Categories)

if err != nil {

return err

}

return nil

}

func (pr \*ProductRepository) updateProductCategories(productID int, categories []models.CreateCategory) error {

tx, err := pr.db.Begin()

if err != nil {

return err

}

defer tx.Rollback()

\_, err = tx.Exec("DELETE FROM product\_categories WHERE product\_id = $1", productID)

if err != nil {

return err

}

for \_, category := range categories {

\_, err = tx.Exec("INSERT INTO product\_categories (product\_id, category\_id) VALUES ($1, $2)", productID, category.ID)

if err != nil {

return err

}

}

return tx.Commit()

}

package repositories

import (

"admin/web-server/admin/models"

"admin/web-server/db"

)

type OrderRepository struct {

db \*db.DataBase

}

func NewOrderRepository(database \*db.DataBase) \*OrderRepository {

return &OrderRepository{db: database}

}

func (or \*OrderRepository) GetAllOrders() ([]models.AdminOrder, error) {

query := `SELECT

o.id,

o.order\_date,

os.name,

p.name,

u.username

FROM "order" o

LEFT JOIN public.order\_status os ON o.order\_status = os.id

LEFT JOIN public.custom\_user u ON o.user\_id = u.id

LEFT JOIN public.product p ON o.product = p.id`

rows, err := or.db.Query(query)

if err != nil {

return nil, err

}

var orders []models.AdminOrder

for rows.Next() {

var order models.AdminOrder

if err := rows.Scan(

&order.ID,

&order.OrderDate,

&order.OrderStatus,

&order.ProductName,

&order.Username,

); err != nil {

return nil, err

}

orders = append(orders, order)

}

return orders, nil

}

func (or \*OrderRepository) GetUserOrders(userID int) ([]models.Order, error) {

query := `

select \* from public.order where user\_id=$1

`

var userOrders []models.Order

rows, err := or.db.Query(query, userID)

if err != nil {

return nil, err

}

for rows.Next() {

var order models.Order

rows.Scan(

&order.ID,

&order.OrderDate,

&order.OrderStatus,

&order.ProductID,

)

userOrders = append(userOrders, order)

}

return userOrders, nil

}

func (or \*OrderRepository) DeleteOrder(id int) error {

query := `delete from "order" where id=$1`

if \_, err := or.db.Exec(query, id); err != nil {

return err

}

return nil

}

package controllers

import (

"admin/web-server/admin/models"

"admin/web-server/admin/services"

"fmt"

"net/http"

"strconv"

"github.com/gin-gonic/gin"

)

type ProductController struct {

service \*services.ProductService

}

func NewProductController(producService \*services.ProductService) \*ProductController {

return &ProductController{

service: producService,

}

}

func (pc \*ProductController) GetAllProducts(c \*gin.Context) {

products, err := pc.service.GetAllProducts()

if err != nil {

c.JSON(http.StatusInternalServerError, gin.H{

"error": err.Error(),

})

return

}

c.JSON(http.StatusOK, products)

}

func (pc \*ProductController) GetProductById(c \*gin.Context) {

id := c.Param("id")

product, err := pc.service.GetProductById(id)

if err != nil {

c.JSON(http.StatusNotFound, gin.H{

"error": err.Error(),

})

return

}

c.JSON(http.StatusOK, product)

}

func (pc \*ProductController) CreateProduct(c \*gin.Context) {

var product models.CreateProduct

if err := c.ShouldBind(&product); err != nil {

c.JSON(http.StatusBadRequest, gin.H{

"error": err.Error(),

})

return

}

if len(product.Categories) == 0 {

c.JSON(400, gin.H{

"error": "no categories",

"product": product,

})

return

}

dst := fmt.Sprintf("uploads/%s", product.Image.Filename)

if err := c.SaveUploadedFile(product.Image, dst); err != nil {

c.JSON(http.StatusInternalServerError, gin.H{

"error": "Не удалось сохранить файл",

})

return

}

if err := pc.service.CreateProduct(product); err != nil {

c.JSON(http.StatusBadRequest, gin.H{

"error": err.Error(),

})

return

}

c.JSON(http.StatusCreated, product)

}

func (pc \*ProductController) DeleteProduct(c \*gin.Context) {

paramId := c.Param("id")

id, \_ := strconv.Atoi(paramId)

if err := pc.service.DeleteProduct(id); err != nil {

c.JSON(http.StatusBadRequest, gin.H{"error": err})

}

}

func (pc \*ProductController) UpdateProduct(c \*gin.Context) {

var product models.UpdateProduct

if err := c.ShouldBind(&product); err != nil {

c.JSON(http.StatusBadRequest, gin.H{"error": err.Error()})

return

}

if len(product.Categories) == 0 {

c.JSON(400, gin.H{

"error": "no categories",

"product": product,

})

return

}

err := pc.service.UpdateProduct(product);

if err != nil {

c.JSON(http.StatusBadRequest, gin.H{

"error": err.Error(),

})

return

}

}

# ПРИЛОЖЕНИЕ 2

БЛОК-СХЕМА АЛГОРИТМА РАБОТЫ ПРОГРАММНОГО СРЕДСТВА

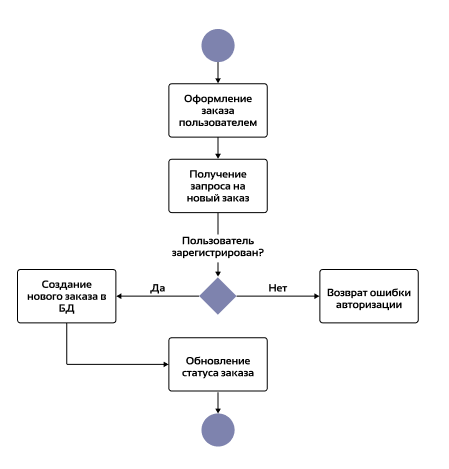


Рисунок П2.1 – Блок-схема работы алгоритма

# ПРИЛОЖЕНИЕ 3

РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

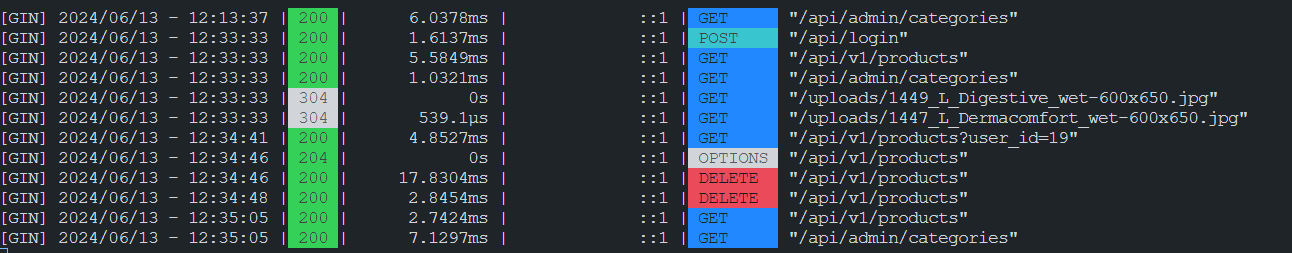
Открыть терминал с директорией проекта. В строке ввода терминала написал go build main.go, что запустит процесс компиляции в исходный .exe файл. После успешной компиляции запустить main.exe файл, находящийся в текущей директории проекта на уровне в main.go файлом. После запуска откроется окно терминала со всеми логами программы. В логах содержится информация о статус кодах, времени выполнения и дате запроса, а так же с информацией, какой конкретно обработчик выполнял запрос.

Рисунок П3.1 – Пример логов

# ПРИЛОЖЕНИЕ 4

ПРОТОКОЛ ИСПЫТАНИЙ ПРОГРАММНОГО СРЕДСТВА

Комиссия в составе

* руководитель дипломного проекта – С.Г. Сутович
* председатель цикловой комиссии – С.Г. Тыбербай
* преподаватель – А.А. Брылева

произвела проверку работоспособности программного средства, разработанного в порядке выполнения дипломного проекта.

1. Информация о программном средстве:
   1. Название программного средства: Backend-сайта зоомагазина.
   2. Номер версии программного средства: 1.0
   3. Дата разработки версии программного средства: 11.06.2024
   4. Назначение программного средства: Программное средство предназначено для управления онлайн-продажами товаров для домашних животных, включая администрирование каталога, управление клиентской базой, обработку заказов и платежей, а также интеграцию с другими системами.
   5. Функции программного средства: Программное средство включает в себя функции управления каталогом товаров, клиентской базой, обработки заказов и платежей, а также интеграции с внешними системами, обеспечивая эффективное управление онлайн-продажами и оптимизацию бизнес-процессов.
   6. Язык программирования: SQL, Go.
   7. Наличие программы инсталляции: Нет.
   8. Тип компьютера(ов): ноутбук.
   9. Требования к компьютеру:
      1. Оперативная память: не менее 128 Mb.
      2. Ёмкость жесткого диска(ов) 500 Mb.
      3. Монитор(ы) с разрешением 800х600, с поддержкой 256 цветов.
      4. Дополнительное периферийное оборудование: манипулятор типа «мышь», принтер струйный или лазерный.
   10. Тип операционной системы (систем): Windows 10 и выше.
2. Информация об организации-заказчике:
   1. Полное официальное наименование: "Витебский государственный колледж электротехники".
   2. Место нахождения (почтовый адрес): 210015, г. Витебск, пр-т Черняховского, 14.
   3. Номер контактного телефона(ов): (8-0212)64-89-16.
   4. Номер факса: (8-0212)64-89-16.
   5. Адрес электронной почты: priem@[vgke.by](mailto:vgpt@vitebsk.by).
3. Информация о разработчике программного средства:
   1. Полное официальное наименование: "Витебский государственный колледж электротехники".
   2. Место нахождения (почтовый адрес): 210015, г. Витебск, пр-т Черняховского, 14.
   3. Номер контактного телефона(ов): +375-29-219-40-69.
   4. Номер факса:
   5. Адрес электронной почты:
   6. Фамилия, имя, отчество учащегося, разработавшего программное средство: Титков Роман Николаевич.

Работоспособность программного средства проверена на контрольных примерах, предусматривающих все возможные варианты исходных и промежуточных данных.

Программное средство признано работоспособным, соответствующим техническому заданию и позволяет решать поставленные перед ним производственные задачи.

Руководитель дипломного проекта С.Г. Сутович

Председатель цикловой комиссии С.Г. Тыбербай

Преподаватель А.А. Брылева