Реферат

Пояснительная записка содержит 41 страницу, 10 рисунков, 0 таблиц.

Количество приложений – 3.

Количество использованных источников – 12.

Ключевые слова: тестирование API (Application Programming Interface), Postman, Docker, OpenCart.

Целью работы является анализ подхода к тестированию веб-сервисов, при этом специфическая задача заключается в проектировании ручных тестовых сценариев для частичного анализа API сервиса OpenCart, представляющего собой открытую платформу для реализации решений в области электронной коммерции.

В аналитической части проведен аналитический обзор принципов тестирования программного обеспечения, архитектуры REST и методик тестирования API. Также включен критический анализ современных техник тест-дизайна и оборудования для тестирования API.

Теоретическая часть работы включает в себя сформулированный план тестирования на основе предварительно установленных требований к сервису. Осуществляется выбор оптимальных техник тест-дизайна и проектируются соответствующие чек-листы.

В инженерной представлены результаты создания коллекции тестов, основанных на моделировании сценариев взаимодействия конечного пользователя с системой.

Завершающая, технологическая и практическая части, охватывают процесс развертывания серверной части приложения в локальной виртуальной среде, демонстрируют применение разработанной коллекции тестов и включает детализированный отчет о проведенном тестировании.

Содержание

Введение

В последние десятилетия, параллельно с развитием интернет-технологий и глобализацией цифровых ресурсов, внимание специалистов в сфере ИТ все больше уделяется вопросам качества и безопасности взаимодействия между различными программными приложениями. Именно здесь вступает в игру API – интерфейс программирования приложений, который служит связующим звеном между различными программными решениями [1].

Если вспомнить исторический контекст, то тестирование API не всегда было столь акцентированным направлением. В начальные годы развития ИТ-индустрии в 1980-1990-х гг., основное внимание уделялось функциональному тестированию и работе с пользовательским интерфейсом. Однако с развитием веб-технологий и появлением облачных сервисов в 2000-х, ситуация кардинально изменилась. Переход к модели веб-сервисов подразумевал постоянное общение различных систем друг с другом, и в этой связи корректное и стабильное функционирование API стало ключевым.

При этом современные сложные архитектуры программных продуктов, интеграция многих веб-сервисов и необходимость в быстром времени отклика требуют особого внимания к тестированию API. Неправильно построенный или не оттестированный API может стать причиной не только технических, но и финансовых потерь, а также ущерба репутации компании.

Таким образом, актуальность тестирования API веб-сервисов неоспорима. В условиях цифровой трансформации, когда каждая деталь взаимодействия между системами может стать критической, качественное и всестороннее тестирование API является необходимым условием успешной и безопасной работы любого веб-приложения или сервиса.

Целью данной работы является проектирование ручных тестов для API сервиса OpenCart с целью оптимизации его функционала и обеспечения высокого уровня надежности. Специфические задачи включают в себя: анализ требований к API, разработку тестовых сценариев, валидацию функциональных и нефункциональных требований, а также исследование возможных уязвимостей [2].

Теоретическая значимость работы заключается в систематизации знаний в области тестирования API, а также в разработке методических рекомендаций для тестирования веб-сервисов, основанных на практическом опыте работы с OpenCart.

С практической точки зрения, работа способствует повышению качества и надежности веб-сервиса OpenCart, что в свою очередь обеспечивает лучший пользовательский опыт и безопасность сделок в сфере электронной коммерции. Разработанные ручные тесты могут

быть использованы как в процессе первичной разработки, так и при последующих обновлениях и модификациях API сервиса.

В первом разделе проводится глубокий анализ концепций тестирования программного обеспечения, специфик REST архитектуры, а также различных методик тестирования API. Осуществляется критический разбор современных подходов тест-дизайна и аппаратного обеспечения, применяемого для тестирования API.

Второй раздел посвящен формированию плана тестирования данного сервиса, основанного на ранее сформулированных требованиях к сервису [3]. Производится отбор оптимальных техник тест-дизайна [4] и создание чек-листов, соответствующих установленным требованиям.

Третий раздел представляет собой результаты разработки коллекции тестовых заданий, основанных на моделировании сценариев, имитирующих взаимодействие конечного пользователя с интерфейсом рассматриваемой системы.

Четвертый раздел затрагивает процесс развертывания серверного компонента приложения в условиях локальной виртуальной среды. В нем демонстрируются этапы применения разработанной коллекции тестов и представляется детализированный отчет о проведенном тестировании сервиса.

Раздел 1. Анализ методологии тестирования API

Введение

В современном мире информационных технологий API (Application Programming Interface) играют ключевую роль в обеспечении взаимодействия между различными программными системами. Особенно актуально это становится при работе с такими платформами, как OpenCart, которая предоставляет широкие возможности для создания интернет-магазинов. Тестирование API становится все более важным в современной разработке программного обеспечения [5]. В частности, для интернет-магазинов на базе OpenCart, где API используется для множества операций, таких как управление товарами, заказами и клиентами, качественное тестирование API является критическим фактором. Этот раздел посвящен анализу методов и инструментов для тестирования REST API в OpenCart.

REST (Representational State Transfer) - это архитектурный стиль, который используется в веб-сервисах. Он предполагает, что каждый ресурс имеет свой уникальный URL, и взаимодействие с этим ресурсом осуществляется через стандартные HTTP-методы. Тестирование REST API включает в себя проверку ответов на запросы, структуры данных и соответствие ожидаемым результатам.

Тестирование REST API является сложной и многоаспектной задачей, требующей внимания к деталям на различных уровнях. Одним из ключевых этапов является проверка ответов на запросы. Это включает в себя анализ HTTP статус-кодов, которые должны соответствовать ожидаемым — например, код 200 для успешного GET-запроса или 201 при создании нового ресурса через POST. Кроме того, необходимо тщательно изучить тело ответа и его заголовки. Тело должно содержать ожидаемую информацию в правильном формате, чаще всего это JSON или XML, а заголовки могут предоставить дополнительную информацию, такую как тип контента или дату последней модификации ресурса.

Что касается запросов, их структура и параметры также играют важную роль в тестировании. Запросы в REST обычно состоят из метода (GET, POST, PUT, DELETE и т.д.), URL-адреса ресурса и, возможно, тела запроса. Параметры могут передаваться в URL, в заголовках или в теле запроса в формате JSON или XML.

Методы запросов HTTP

HTTP (HyperText Transfer Protocol) — это протокол передачи гипертекста, который является основой любого взаимодействия данных в Интернете. HTTP функционирует как протокол запрос-ответ между клиентом и сервером. Клиент, обычно веб-браузер, отправляет запрос на сервер, а сервер возвращает ответ. Этот ответ может быть в виде HTML-страницы, изображения, файла или любого другого ресурса, доступного через веб.

Методы запросов HTTP определяют действие, которое должно быть выполнено на ресурсе. Они являются одним из ключевых элементов в HTTP протоколе и используются для выполнения различных операций с ресурсами на веб-сервере. Вот основные методы HTTP:

1. GET

- Описание: Метод «GET» используется для запроса данных с сервера.

- Особенности: Этот метод только получает данные; он не изменяет ничего на сервере.

- Пример: Запрос каталога товаров в интернет-магазине.

2. POST

- Описание: Метод «POST» используется для отправки данных на сервер для создания нового ресурса.

- Особенности: Данные отправляются в теле запроса. Этот метод может изменять состояние сервера.

- Пример: Отправка формы регистрации на веб-сайте.

3. PUT

- Описание: Метод «PUT» используется для обновления существующего ресурса на сервере или создания нового, если ресурс не существует.

- Особенности: Как и в случае с POST, данные отправляются в теле запроса.

- Пример: Обновление профиля пользователя.

4. DELETE

- Описание: Метод удаляет указанный ресурс.

- Особенности: Этот метод изменяет состояние сервера, удаляя ресурс.

- Пример: Удаление электронного письма из почтового ящика.

5. HEAD

- Описание: Метод «HEAD» аналогичен методу GET, но сервер возвращает только заголовки ответа, без тела.

- Особенности: Используется для получения метаинформации о ресурсе без загрузки самого ресурса.

- Пример: Проверка наличия веб-страницы.

6. PATCH

- Описание: Метод «PATCH» применяется для частичного изменения ресурса.

- Особенности: В отличие от PUT, который обновляет ресурс полностью, PATCH изменяет только те атрибуты, которые были указаны.

- Пример: Обновление отдельного поля в профиле пользователя.

7. OPTIONS

- Описание: Метод «OPTIONS» используется для описания параметров соединения с ресурсом.

- Особенности: Он может быть использован для определения методов, поддерживаемых для целевого ресурса.

- Пример: Определение методов, доступных для URL.

Каждый из этих методов имеет свои особенности и применяется в зависимости от того, какое действие необходимо выполнить с ресурсом на сервере. Они являются фундаментальной частью архитектуры RESTful веб-сервисов и веб-разработки в целом.

Методы GET и POST являются наиболее часто используемыми, поскольку они обеспечивают базовые операции чтения и записи, которые являются фундаментальными для большинства веб-приложений и API. Эти методы достаточно полно покрывают тестируемый в контексте решаемой задачи функционал.

Передача параметров в HTTP запросах

В REST запросах параметры могут передаваться различными способами, в зависимости от конкретного случая и требований к функциональности. Вот некоторые из наиболее распространенных методов передачи параметров:

1. Параметры в URL (')

Пример с GET-запросом:

‘‘‘

GET /users?age=30&gender=male

‘‘‘

В этом примере параметры «age» и «gender» передаются прямо в URL.

2. Параметры в пути (Path Parameters)

Пример с GET-запросом:

‘‘‘

GET /users/123

‘‘‘

Здесь «123» является параметром, который указывает на конкретного пользователя.

3. Параметры в заголовках (Header Parameters)

Пример с GET-запросом:

‘‘‘

GET /users

Authorization: Bearer <token>

‘‘‘

В этом примере параметр «Authorization» передается в заголовке HTTP-запроса.

4. Параметры в теле запроса (Body Parameters)

Пример с POST-запросом:

```
POST /users
Content-Type: application/json
{
  "name": "John",
  "age": 30
}
```

Здесь параметры «name» и «age» передаются в теле запроса в формате JSON.

5. Параметры в форме (Form Parameters)

Пример с POST-запросом:

```
POST /users
Content-Type: application/x-www-form-urlencoded
name=John&age=30
```

В этом случае данные формы передаются в теле запроса с типом «application/x-www-form-urlencoded».

6. Комбинированный пример

```
GET /users/123/orders?status=active
Authorization: Bearer <token>
```

В этом примере используются параметры в пути («123»), параметры в URL («status=active») и параметры в заголовке («Authorization»).

В контексте данной задачи было решено использовать такие методы передачи параметров как «Query Parameters», «Path Parameters» и «Body Parameters», так как они являются наиболее простыми и часто используемыми, а также полностью отвечают нашим потребностям.

Статус-коды HTTP

Статус-коды HTTP — это трехзначные числовые коды, которые сервер возвращает в ответ на запросы от клиента. Эти коды предназначены для того, чтобы дать клиенту

представление о результате его запроса. Статус-коды разделены на пять классов, определенных первой цифрой кода:

1. 1xx (Информационные): Эти коды указывают, что сервер принял запрос и требует дальнейших действий для его завершения.
    - 100 Continue: Сервер принял начальную часть запроса и ждет продолжения.
2. 2xx (Успешно): Коды этой категории указывают, что запрос был успешно принят и обработан.
    - 200 OK: Запрос успешно выполнен. Ответ зависит от метода запроса.
    - 201 Created: Запрос был успешно выполнен и в результате был создан новый ресурс.
    - 204 No Content: Запрос успешно обработан, но в ответе нет содержимого.
3. 3xx (Перенаправление): Эти коды указывают, что для успешного завершения запроса необходимо выполнить дополнительные действия.
    - 301 Moved Permanently: Ресурс был перемещен на другой URL.
    - 304 Not Modified: Ресурс не был изменен с момента последнего запроса.
4. 4xx (Ошибки клиента): Эти коды указывают на ошибки со стороны клиента, например, неправильный запрос или отсутствие авторизации.
    - 400 Bad Request: Сервер не понимает запрос из-за неверного синтаксиса.
    - 401 Unauthorized: Для доступа к ресурсу требуется аутентификация.
    - 404 Not Found: Сервер не может найти запрашиваемый ресурс.
5. 5xx (Ошибки сервера): Эти коды указывают, что сервер не смог выполнить валидный запрос.
    - 500 Internal Server Error: Сервер столкнулся с ошибкой и не может выполнить запрос.
    - 503 Service Unavailable: Сервер временно не готов обработать запрос.

Техники тест-дизайна

Техники тест-дизайна являются неотъемлемой частью современного процесса разработки программного обеспечения, выполняя критически важную роль в обеспечении качества и надежности программных продуктов. Они служат для систематизации и оптимизации процесса тестирования, позволяя тем самым сократить затраты времени и ресурсов, при этом увеличивая эффективность выявления дефектов и улучшая общую надежность системы.

Динамические техники тест-дизайна

Динамические техники тест-дизайна фокусируются на анализе поведения системы во время её выполнения. Эти техники можно классифицировать на основе различных критериев, включая уровень доступа к исходному коду и типы тестовых сценариев.

1. Функциональное Тестирование ("Черного Ящика")

В рамках функционального тестирования тестировщики не имеют доступа к исходному коду и фокусируются на внешнем поведении системы. Техники, такие как "Эквивалентное разбиение" и "Анализ граничных значений", позволяют эффективно охватить различные варианты входных данных и условий.

2. Структурное Тестирование ("Белого Ящика")

Структурное тестирование предполагает доступ к исходному коду и фокусируется на его внутренней структуре. "Тестирование путей" и "Тестирование условий" позволяют обеспечить высокий уровень покрытия кода.

3. Комбинированные Техники ("Серого Ящика")

Комбинированные техники, такие как "Тестирование на основе моделей" и "Тестирование на основе рисков", совмещают элементы функционального и структурного тестирования, предоставляя более глубокий и комплексный анализ системы.

В отличие от статических техник, которые фокусируются на анализе кода и документации без его выполнения, динамические техники требуют активного взаимодействия с работающей системой [4]. Исследовательские техники, хотя и могут быть динамическими, часто менее структурированы и зависят от интуиции и опыта тестировщика.

Тестирование "черного ящика" является одним из наиболее распространенных подходов к тестированию API, и его можно успешно применить для тестирования API платформы OpenCart. Этот метод фокусируется на функциональности системы, игнорируя внутреннюю структуру кода. В контексте API платформы OpenCart, тестирование "черного ящика" может включать в себя следующие аспекты:

1. Эквивалентное Разбиение

Цель: Проверить, как API реагирует на различные категории входных данных.

Пример: При тестировании метода для создания нового продукта можно разделить входные данные на эквивалентные классы, такие как валидные и невалидные идентификаторы продукта, и проверить, как система реагирует на каждый из них.

2. Анализ Граничных Значений

Цель: Проверить, как система реагирует на крайние значения входных данных.

Пример: При тестировании метода для добавления товара в корзину можно использовать максимально и минимально допустимые значения количества товара.

3. Тестирование Сценариев

Цель: Проверить, как API справляется с различными бизнес-сценариями.

Пример: При тестировании процесса оформления заказа можно создать сценарии, которые включают в себя добавление товаров в корзину, применение скидок и выбор способа доставки.

4. Тестирование Обработки Ошибок

Цель: Убедиться, что API корректно обрабатывает ошибочные запросы.

Пример: При тестировании можно отправить запросы с недопустимыми параметрами и проверить, возвращает ли API соответствующие коды ошибок и сообщения.

5. Тестирование Безопасности

Цель: Проверить, как API соблюдает стандарты безопасности.

Пример: Можно провести тестирование на наличие уязвимостей, таких как SQL-инъекции или перебор паролей.

В заключение тестирование "черного ящика" предоставляет эффективный способ проверки функциональности API платформы OpenCart, не требуя доступа к исходному коду. Этот подход позволяет обеспечить высокий уровень качества и надежности системы [6], что является критически важным для успешного функционирования любой коммерческой платформы.

Современные инструменты для тестирования API

В современном мире разработки программного обеспечения существует множество инструментов для тестирования API, каждый из которых имеет свои преимущества и недостатки. Некоторые из наиболее популярных инструментов включают:

- SoapUI: Один из старейших и наиболее надежных инструментов для тестирования как REST, так и SOAP API.
- JMeter: Отлично подходит для проведения нагрузочного тестирования, но имеет более крутую кривую обучения.
- Rest-Assured: Библиотека для автоматизированного тестирования REST API на Java.
- Insomnia: Интерфейс пользователя дружелюбен, но функциональность несколько ограничена по сравнению с другими инструментами.
- Postman: Один из наиболее популярных и функциональных инструментов для тестирования API.

После тщательного анализа выбор был сделан в пользу Postman по следующим причинам:

- Простота Использования

Интуитивный интерфейс: Postman имеет очень понятный и легко настраиваемый пользовательский интерфейс.

- Многофункциональность

Поддержка различных типов API: Postman поддерживает тестирование REST, SOAP и GraphQL API.

Автоматизация тестов: С помощью встроенного функционала можно легко создавать автоматизированные тестовые сценарии.

- Гибкость и Масштабируемость

Переменные окружения: Postman позволяет устанавливать переменные окружения, что упрощает тестирование в различных средах.

- Коллаборация

Облачное хранение: Postman предлагает облачное хранение, что упрощает совместную работу над проектами.

Публичные и приватные коллекции: Можно легко делиться коллекциями запросов с командой или сообществом.

- Документация и Поддержка

Обширная документация [7]: Postman предлагает обширные руководства и примеры, что ускоряет процесс обучения.

В контексте современных требований к тестированию API, Postman выделяется своей многофункциональностью, гибкостью и простотой использования. Эти факторы делают его отличным выбором для тестирования API на всех этапах разработки программного обеспечения.

Выводы

1. В ходе анализа было установлено, что методы GET и POST являются доминирующими в практике разработки веб-приложений и API. Эти методы обеспечивают основные операции чтения и записи, которые конституируют фундаментальную базу для функционирования большинства современных веб-систем. В рамках решаемой задачи эти методы предоставляют адекватное покрытие функциональных требований, что подтверждает их эффективность и целесообразность применения.
2. Для решения поставленной задачи были выбраны методы передачи параметров «Query Parameters», «Path Parameters» и «Body Parameters». Эти методы не только являются наиболее распространенными в сфере разработки API, но и

оптимально соответствуют функциональным и техническим требованиям для решения поставленной задачи.

3. Тестирование с применением методологии "черного ящика" демонстрирует высокую эффективность в контексте проверки функциональности API платформы OpenCart. Отсутствие необходимости в доступе к исходному коду делает этот подход универсальным и позволяет достичь высокого уровня качества и надежности системы, что является критически важным для успешной эксплуатации коммерческих платформ.
4. В соответствии с современными требованиями к тестированию API, инструмент Postman проявляет себя как выдающийся выбор, обладая рядом ключевых преимуществ. Эти преимущества включают в себя многофункциональность, гибкость и простоту использования, что делает Postman идеально подходящим для тестирования API на различных этапах жизненного цикла разработки программного обеспечения.

Цели и задачи УИР

Основной целью данной работы является комплексный анализ и тестирование API платформы OpenCart с применением современных методологических подходов и инструментальных средств в целях обеспечения высокого уровня качества системы.

Задачи:

- Изучение Теоретических Основ: Первоначальный этап исследования предполагает глубокий анализ существующих принципов тестирования программного обеспечения и архитектуры REST. В рамках данной задачи необходимо провести сравнительный анализ различных интерфейсов, реализуемых через API, с целью выявления оптимальных методов и подходов.
- Анализ Современных Техник и Инструментов: Следующим этапом является изучение современных методик тест-дизайна и инструментов для тестирования API. В данном контексте акцент делается на оценке функциональных возможностей и применимости различных инструментов, в частности, таких как Postman.
- Разработка Тестовых Сценариев и Чек-листов: На основе полученных теоретических данных и методологических рекомендаций, следует разработать комплекс тестовых сценариев и чек-листов, которые будут служить основой для систематической проверки функциональных и нефункциональных аспектов API.

- Техническая Реализация и Практическое Тестирование: Этот этап включает в себя разработку технических требований к сервису, проектирование коллекции тестов и последующее развертывание серверной части приложения в виртуальной среде для проведения практических испытаний.
- Анализ Результатов и Оптимизация Процесса: Завершающим этапом исследования является анализ результатов тестирования с целью выявления и документации обнаруженных дефектов и несоответствий. На основе проведенного анализа предполагается разработка рекомендаций по оптимизации процесса тестирования.

Таким образом, представленные цели и задачи обеспечивают комплексный подход к выполнению работы, начиная с теоретического анализа и заканчивая практической реализацией.

Раздел 2. Моделирование предметной области тестирования OpenCart API

Введение

Моделирование предметной области тестирования OpenCart API предполагает выделение ключевых компонентов и взаимосвязей между ними, чтобы обеспечить эффективный и систематический подход к тестированию.

Ключевые компоненты:

1. Пользовательский интерфейс (UI): Включает в себя все элементы, с которыми взаимодействует конечный пользователь.

2. Бэкенд-сервисы: Основные серверные компоненты, которые обрабатывают бизнес-логику.

3. База данных: Хранение всех пользовательских данных, каталогов товаров, заказов и т. д.

4. REST API: Интерфейс для взаимодействия между клиентом и сервером, включая методы для CRUD-операций (Создание, Чтение, Обновление, Удаление).

5. Сторонние сервисы и интеграции: Платежные системы, системы управления складом и т.д.

Взаимосвязи:

1. UI <-> REST API: Пользовательский интерфейс взаимодействует с бэкендом через REST API.

2. REST API <-> Бэкенд-сервисы: API вызывает соответствующие методы бэкенда для выполнения бизнес-логики.

3. Бэкенд-сервисы <-> База данных: Бэкенд-сервисы читают и записывают данные в базу данных.

4. Бэкенд-сервисы <-> Сторонние сервисы: Интеграция с платежными системами и другими внешними сервисами.

Тестовые сценарии:

1. Функциональное тестирование API: Проверка корректности выполнения CRUD-операций, валидации данных и обработки ошибок.

2. Интеграционное тестирование: Проверка взаимодействия между API, бэкенд-сервисами и базой данных.

3. Нагрузочное тестирование: Проверка производительности и отказоустойчивости системы.

4. Безопасность: Проверка на наличие уязвимостей, таких как SQL-инъекции, CSRF-атаки и т. д.

В нашем случае тестирование не будет проводиться по следующим направлениям: интеграция с платежными системами, интеграция со сторонними сервисами, масштабируемость, производительность, защита от SQL-инъекций и общая безопасность.

Эта модель предметной области обеспечивает комплексный подход к тестированию OpenCart API, позволяя выявить и устранить потенциальные проблемы на различных уровнях системы.

Разработка требований к сервису

В рамках УИР была проведена систематизация требований к API интернет-торговой платформы OpenCart. Эти требования, охватывающие функциональные и нефункциональные аспекты, являются ключевыми для обеспечения качества сервиса [8].

В частности, API должно предоставлять надежные механизмы аутентификации и авторизации с возможностью восстановления учетных данных. Основной акцент делается на управлении ассортиментом, что включает в себя добавление, редактирование и удаление товарных позиций, а также категоризацию и тегирование [9]. Кроме того, сервис должен обеспечивать эффективное управление заказами, начиная от их создания и до завершения транзакции.

С точки зрения нефункциональных требований, основное внимание уделяется предоставлению полной и актуальной документации по API и качественной технической поддержке.

Однако стоит отметить, что в нашем исследовании не учитываются аспекты, связанные с интеграцией с платежными системами, интеграцией со сторонними сервисами, масштабируемостью, производительностью, защитой от SQL-инъекций и общей безопасностью. Эти области выходят за рамки текущего анализа.

План тестирования

Введение

План тестирования предназначен для определения объема, подхода, ресурсов и графика всех мероприятий по тестированию API платформы OpenCart.

В плане определяются элементы, подлежащие тестированию, тестируемые функции, типы выполняемого тестирования, персонал, ответственный за тестирование, а также риски, связанные с планом.

Объект тестирования: REST API платформы OpenCart.

Основные Задачи

1. Проверка корректности обработки запросов различных типов.
2. Проверка валидации входных данных.
3. Проверка корректности ответов от сервера.

4. Проверка обработки ошибочных сценариев.

Предполагается провести:

1. Позитивное тестирование приложения (корректные шаги, корректные данные)
2. Негативное тестирование (подразумевает введение некорректных данных)

Функции системы, которые не включены в спецификации требований тестирования программного обеспечения:

1. Интеграция с платежными системами
2. Интеграция со сторонними сервисами
3. Масштабируемость
4. Производительность
5. Защита от SQL-инъекций
6. Безопасность

Инструменты и оборудование

1. Postman для формирования и отправки HTTP-запросов
2. Локальный сервер для развертывания OpenCart

Методология и техники тестирования

1. Тестирование "черного ящика"
2. Эквивалентное разделение
3. Граничные значения

План работ

1. Подготовка Тестовой Среды: Установка и настройка всех необходимых инструментов.
2. Разработка Тестовых Сценариев: Создание сценариев на основе методик "черного ящика".
3. Выполнение Тестов: Проведение тестирования согласно разработанным сценариям.
4. Анализ Результатов: Сбор и анализ данных, полученных в ходе тестирования.
5. Подготовка Отчета: Составление подробного отчета о проведенном тестировании, включая обнаруженные дефекты и рекомендации.

Модули тестирования

Были выделены следующие модули для тестирования и соответствующие им чек-листы:

1. Модуль: Корзина

Добавление продукта в корзину:

- Проверить, что продукт успешно добавляется в корзину при отправке POST-запроса.
- Проверить код ответа (должен быть 200).
- Проверить время ответа (не более 200 мс).

Удаление продукта из корзины:

- Проверить, что продукт удаляется из корзины при отправке POST-запроса на удаление.
- Проверить код ответа (должен быть 200).

Редактирование количества продукта:

- Проверить, что количество продукта в корзине изменяется при отправке POST-запроса на редактирование.
- Проверить код ответа (должен быть 200).

Проверка выбора способа доставки

- Проверить, что способ доставки корректно сохраняется при отправке POST-запроса.
- Проверить код ответа (должен быть 200).

Использование купона:

- Проверить, что купон успешно применяется при отправке POST-запроса.
- Проверить код ответа (должен быть 200).

Использование ваучера:

- Проверить, что ваучер успешно применяется при отправке POST-запроса.
- Проверить код ответа (должен быть 200).

2. Модуль: Заказ товара

Создание заказа:

- Проверить, что заказ успешно создается при отправке POST-запроса.
- Проверить код ответа (должен быть 200).

3. Модуль: Аутентификация API

Вход в API:

- Проверить, что аутентификация в API происходит успешно при отправке POST-запроса с корректными данными.
- Проверить код ответа (должен быть 200).

Запуск тестов при обновлении системы

При каждом обновлении системы необходимо повторно запускать все тесты для проверки сохранения функциональности и отсутствия регрессии.

Требования к передаче наружу

Результаты тестирования должны быть документированы и переданы команде разработки для дальнейшего анализа и исправления ошибок.

Критерии завершения:

1. Все запланированные тестовые сценарии выполнены.
2. Все критические дефекты устранены или получено подтверждение их допустимости.

Таким образом, данный тест-план предоставляет комплексный подход к тестированию API платформы OpenCart с использованием методологии "черного ящика", что позволяет обеспечить высокий уровень качества и надежности системы.

Этот тест-план является комплексным инструментом для обеспечения качества API платформы OpenCart и должен быть строго соблюден всеми участниками тестирования.

Выводы

1. Разработана детализированная система требований к API платформы OpenCart, фокусируясь на ключевых функциональных аспектах, таких как аутентификация и управление заказами, при этом исключая из рассмотрения вопросы интеграции, масштабируемости и безопасности.
2. Разработан комплексный тест-план для API платформы OpenCart, охватывающий ключевые функциональные аспекты системы. План включает в себя методологию "черного ящика" и различные техники тестирования, такие как эквивалентное разделение и граничные значения.
3. Определены основные задачи и модули для тестирования, включая функционал корзины, заказ товара, форму входа и аутентификацию в API. Для каждого модуля разработаны детализированные чек-листы, что обеспечивает систематический и организованный подход к тестированию.
4. Установлены критерии завершения тестирования и требования к документации результатов. Это обеспечивает четкую последовательность действий и позволяет команде разработки эффективно анализировать и исправлять обнаруженные дефекты.
5. Определены инструменты и оборудование для тестирования, включая Postman и локальный сервер. Это обеспечивает необходимую инфраструктуру для эффективного и надежного тестирования.

Раздел 3. Проектирование коллекции тестов

Введение

Postman — это платформа для тестирования, разработки и документирования API. Она предоставляет интуитивно понятный интерфейс для отправки HTTP-запросов, анализа ответов и автоматизации тестовых сценариев [7]. В данном разделе рассматривается проектирование тестовых сценариев для API интернет-магазина на платформе OpenCart с использованием Postman.

Используемые переменные окружения

- «{{base_url}}»: Базовый URL интернет-магазина.
- «{{api_token}}»: Токен для аутентификации в API.

Коллекция запросов

Демонстрация коллекции запросов приведена в приложении 1.

1. Cart (Корзина)

1.1. Add Product (Добавление продукта)

Метод: POST

URL: «{{base_url}}/en-gb?route=checkout/cart.add&api_token={{api_token}}»

Параметры:

- «product_id»: ID продукта
- «quantity»: Количество

1.2. Remove product (Удаление продукта)

Метод: POST

URL: «{{base_url}}/en-gb?route=checkout/cart.remove&api_token={{api_token}}»

Параметры:

- «quantity»: Количество
- «key»: Ключ продукта

1.3. Edit product quantity (Изменение количества продукта)

Метод: POST

URL: «{{base_url}}/en-gb?route=checkout/cart.edit&api_token={{api_token}}»

Параметры:

- «key: Ключ продукта
- «quantity: Количество

1.4. Save shipping method (Выбор варианта доставки)

Метод: POST

URL: «{{base_url}}/index.php?route=extension/opencart/total/shipping.quote&language=en-gb&api_token={{api_token}}»

Параметры:

- «shipping_method»: Метод доставки

1.5. Use coupon code (Использование купона)

Метод: POST

URL: «{{base_url}}/en-gb?route=extension/opencart/total/coupon.save&api_token={{api_token}}»

Параметры:

- «coupon»: Код купона

1.6. Use voucher (Использование ваучера)

Метод: POST

URL: «{{base_url}}/en-gb?route=extension/opencart/total/voucher.save&api_token={{api_token}}»

Параметры:

- «voucher»: Код ваучера

2. Order (Заказ)

2.1. Order create (Создание заказа)

Метод: POST

URL: «{{base_url}}/index.php?route=checkout/register.save&language=en-gb&api_token={{api_token}}»

Параметры:

- "firstname": Имя пользователя
- "lastname": Фамилия пользователя
- "email": e-mail пользователя
- "shipping_address_1": Адрес доставки
- "shipping_city": Город доставки
- "shipping_country_id": ID страны
- "shipping_zone_id": ID зоны доставки

3. API authentication (Аутентификация API)

3.1. API login (Вход в API)

Метод: POST

URL: «http://localhost:8081/index.php?route=api/account/login»

Параметры:

- «username»: Имя пользователя
- «key»: Ключ API

API token — это уникальный идентификатор, который используется для аутентификации пользователя или приложения, пытающегося получить доступ к API. Этот токен обычно передается в заголовке или параметрах запроса для идентификации и валидации запроса. В контексте OpenCart API, API token необходим для обеспечения безопасности и ограничения доступа только к авторизованным пользователям или системам.

Это исчерпывающее описание проектирования тестовых сценариев в Postman для API интернет-магазина на платформе OpenCart. Каждый запрос имеет свои параметры и тестовые сценарии, направленные на проверку различных аспектов функциональности и производительности API [10].

Выводы

1. Разработана коллекция тестов, эмулирующая взаимодействие пользователя с интерфейсом интернет-магазина на платформе OpenCart. Сценарии охватывают ключевые аспекты функциональности, включая работу с корзиной, оформление заказов и управление учетной записью пользователя.
2. Применение Postman позволяет автоматизировать процесс тестирования и обеспечивает высокую точность в воспроизведении пользовательских действий. Использование переменных окружения упрощает управление тестами и повышает их гибкость.
3. Проектирование тестов в Postman для OpenCart API является эффективным методом для обеспечения качества программного продукта, позволяющим быстро идентифицировать возможные дефекты и улучшить пользовательский опыт.

Раздел 4. Развертывание серверной части приложения виртуальном окружении Docker и ручное тестирование сервиса OpenCart

Процесс развертывания серверной части приложения

Docker представляет собой платформу для разработки, доставки и эксплуатации приложений в контейнерах [11]. Выбор в пользу Docker обусловлен его высокой эффективностью, модульностью и возможностью изоляции ресурсов, что обеспечивает универсальность и воспроизводимость развертывания приложений.

Файл «docker-compose.yml» является инструкцией для Docker Compose, инструмента для определения и запуска многоконтейнерных приложений Docker. Этот файл описывает, как создать ваше приложение на основе Docker-контейнеров.

Подробное описание файла docker-compose.yml для создания приложения для работы с OpenCart представлено в приложении 2.

Этапы развертывания:

1. Подготовка: Убедитесь, что установлены Docker и Docker Compose.

2. Конфигурация: Создайте или модифицируйте файл «docker-compose.yml».

3. Инициализация: Запустите команду `docker-compose up` для создания и старта всех сервисов.

4. Проверка: После успешного запуска, OpenCart будет доступен по адресу “localhost:8081”.

Демонстрация работы Docker представлена в приложении 3.

Отчет о тестировании

Цель данного отчета — представить результаты тестирования API интернет-магазина OpenCart с использованием инструмента Postman. Тестирование проводилось с целью проверки корректности работы функций корзины, заказов и аутентификации API.

Демонстрация коллекции запросов приведена в приложении 1.

1. Cart (Корзина)

1.1. Add Product (Добавление продукта)

Метод: POST

URL: «{{base_url}}/en-gb?route=checkout/cart.add&api_token={{api_token}}»

Параметры:

- «product_id»: 40
- «quantity»: 3

Тестовые сценарии:

- Проверка статус-кода 200: Удостоверение в успешной обработке запроса.

- Проверка времени ответа: Оценка производительности API, время ответа должно быть меньше 200 мс.
- Проверка успешного добавления: Подтверждение, что продукт был успешно добавлен в корзину.

Фактический результат:

- Статус-код 200.
- Время ответа 69 мс.
- Тело ответа:

{

"success": "Success: You have added iPhone to your shopping cart!"

}

1.2. Remove product (Удаление продукта)

Метод: POST

URL: «{{base_url}}/en-gb?route=checkout/cart.remove&api_token={{api_token}}»

Параметры:

- «key»: 27

Тестовые сценарии:

- Проверка статус-кода 200: Удостоверение в успешной обработке запроса.
- Проверка времени ответа: Оценка производительности API, время ответа должно быть меньше 200 мс.
- Проверка успешного удаления: Подтверждение, что продукт был успешно удален из корзины.

Фактический результат:

- Статус-код 200.
- Время ответа 41 мс.
- Тело ответа:

{

"success": "Success: You have removed an item from your shopping cart!"

}

1.3. Edit product quantity (Изменение количества продукта)

Метод: POST

URL: «{{base_url}}/en-gb?route=checkout/cart.edit&api_token={{api_token}}»

Параметры:

- «key: 28
- «quantity: 33

Тестовые сценарии:

- Проверка статус-кода 200: Удостоверение в успешной обработке запроса.
- Проверка времени ответа: Оценка производительности API, время ответа должно быть меньше 200 мс.
- Проверка успешного изменения: Подтверждение, что количество продукта в корзине было успешно изменено.

Фактический результат:

- Статус-код 200.
- Время ответа 84 мс.
- Тело ответа:

{
"success": "Success: You have modified your shopping cart!"
}

1.4. Save shipping method (Выбор варианта доставки)

Метод: POST

URL:
«{{base_url}}/index.php?route=extension/opencart/total/shipping.quote&language=en-gb&api_token={{api_token}}»

Параметры:

- “shippimh_method”: flat.flat

Тестовые сценарии:

- Проверка статус-кода 200: Удостоверение в успешной обработке запроса.
- Проверка времени ответа: Оценка производительности API, время ответа должно быть меньше 200 мс.
- Проверка корректности результата: Подтверждение, что выбор варианта доставки был сохранен.

Фактический результат:

- Статус-код 200.
- Время ответа 59 мс.
- Тело ответа:

{

"success": "Success: Your shipping estimate has been applied!"

}

1.5. Use coupon code (Использование купона)

Метод: POST

URL: «{{base_url}}/en-gb?route=extension/opencart/total/coupon.save&api_token={{api_token}}»

Параметры:

- «coupon»: 2222

Тестовые сценарии:

- Проверка статус-кода 200: Удостоверение в успешной обработке запроса.
- Проверка времени ответа: Оценка производительности API, время ответа должно быть меньше 200 мс.
- Проверка применения купона: Подтверждение, что купон был успешно применен и скидка учтена.

Фактический результат:

- Статус-код 200.
- Время ответа 59 мс.
- Тело ответа:

{

"success": "Success: Your coupon discount has been applied!"

}

1.6. Use voucher (Использование ваучера)

Метод: POST

URL: «{{base_url}}/en-gb?route=extension/opencart/total/voucher.save&api_token={{api_token}}»

Параметры:

- «voucher»: 1611

Тестовые сценарии:

- Проверка статус-кода 200: Удостоверение в успешной обработке запроса.
- Проверка времени ответа: Оценка производительности API, время ответа должно быть меньше 200 мс.
- Проверка применения ваучера: Подтверждение, что ваучер был успешно применен и скидка учтена.

Фактический результат:

- Статус-код 200.

- Время ответа 59 мс.
- Тело ответа:

{

"success": "Success: Your gift certificate discount has been applied!"

}

2. Order (Заказ)

2.1. Order create (Создание заказа)

Метод: POST

URL: «{{base_url}}/index.php?route=checkout/register.save&language=en-gb&api_token={{api_token}}»

Параметры:

- "firstname": “Ivan”
- "lastname": “Ivanov”
- "email": “ivan@ya.ru”
- "shipping_address_1": “Address”
- "shipping_city": “Moscow”
- "shipping_country_id": 176
- "shipping_zone_id": 4333

Тестовые сценарии:

- Проверка статус-кода 200: Удостоверение в успешной обработке запроса.
- Проверка времени ответа: Оценка производительности API, время ответа должно быть меньше 200 мс.
- Проверка успешного создания заказа: Подтверждение, что заказ был успешно создан с указанными параметрами.

Фактический результат:

- Статус-код 200.
- Время ответа 51 мс.
- Тело ответа:

{

"success": "Success: Your guest account information has been saved!"

}

3. API authentication (Аутентификация API)

3.1. API login (Вход в API)

Метод: POST

URL: «http://localhost:8081/index.php?route=api/account/login»

Параметры:

- «username»: Marat
- «key»: 81kRJlSnNg4VA8JaxQEd1TLR1ZzE3e6bEiuUVU2OCNhXatCWZQZss76upjUZ4DNqLVyexdlp87KZ5Wqt2ywipXJV2rWzslFvzbmuVDbmWvYXuGDVyaSOeJ2oFhIQ2i4qdwr7iSuwoCUv0mNbatbmKlZkppkHzoQgRGdCc61ICaQhLiUgnZsLYuBPj3bmdgkjb8H1ze044a8mmIJzftD41Gpwvo67PQ4zi0NMCeHWHMSVv4Y2EBaMhHMwR2veHFTa

Тестовые сценарии:

- Проверка статус-кода 200: Удостоверение в успешной обработке запроса.
- Проверка времени ответа: Оценка производительности API, время ответа должно быть меньше 200 мс.
- Проверка успешного входа: Подтверждение, что токен API успешно получен.

Фактический результат:

- Статус-код 200.
- Время ответа 146 мс.
- Тело ответа:


```
{
   "success": "Success: API session successfully started!",
   "api_token": "3b79682d9d277f4c9992ac91e4"
}
```


Было проведено частичное тестирование API интернет-магазина OpenCart. Тестирование включало в себя проверку функциональности корзины, процесса создания заказов и аутентификации API. Для этих целей использовался инструмент Postman, который позволяет эмулировать различные типы HTTP-запросов и анализировать ответы сервера.

В ходе тестирования были проверены ключевые параметры, такие как статус-коды ответов, время ответа сервера и корректность выполнения операций (добавление/удаление товаров в корзину, создание заказов, аутентификация). Все тестовые сценарии успешно прошли проверку, подтверждая высокую надежность и производительность API.

Выводы

1. Развертывание серверной части приложения OpenCart в Docker обеспечивает высокую степень модульности и изоляции, что является критически важным для обеспечения надежности и масштабируемости в современных распределенных системах.

2. Тестирование API OpenCart показало, что система способна обрабатывать запросы с высокой производительностью, что подтверждается временем ответа менее 200 мс для всех тестовых сценариев. Это указывает на высокую оптимизацию и производительность серверной части приложения.
3. Все тестовые сценарии, включая операции с корзиной, создание заказов и аутентификацию, прошли успешно, подтверждая консистентность и надежность API. Это гарантирует стабильное и надежное взаимодействие между клиентской и серверной частями приложения.
4. Использование файла docker-compose.yml для конфигурации всех компонентов приложения обеспечивает централизованный контроль и упрощает процесс развертывания и масштабирования.
5. Система разработана таким образом, что позволяет легко добавлять новые тестовые сценарии и модифицировать существующие, что делает ее гибкой и легко адаптируемой к изменяющимся требованиям.

Заключение

В ходе данной работы было проведено комплексное исследование, направленное на анализ методологии тестирования API, моделирование предметной области тестирования OpenCart API, проектирование коллекции тестов в Postman, а также развертывание серверной части приложения в виртуальном окружении Docker и ручное тестирование сервиса OpenCart.

Основные результаты и достижения цели

- Анализ методологии тестирования API: Изучены и проанализированы ключевые аспекты тестирования API, включая методы HTTP-запросов, передачу параметров, статус-коды и современные инструменты для тестирования API. Это позволило сформулировать эффективные методы и техники для дальнейшего тестирования.
- Моделирование предметной области: Разработаны требования и план тестирования для OpenCart API, что обеспечило четкую структуру и направленность исследования.
- Проектирование коллекции тестов в Postman: Создана коллекция запросов, которая может быть использована для автоматизированного тестирования OpenCart API, ускоряя процесс верификации и валидации.
- Развертывание и тестирование в Docker: Успешно развернута серверная часть приложения в Docker и проведено ручное тестирование, подтвердив эффективность и надежность системы.

Цель работы — исследовать и оптимизировать процесс тестирования API интернет-магазина OpenCart — была успешно достигнута. Все поставленные задачи выполнены, и методология тестирования оптимизирована для текущих и будущих проектов.

Перспективы дальнейших исследований

- Автоматизация Тестирования: Возможность дальнейшей автоматизации тестов для ускорения процесса разработки и обновления [12].
- Интеграция с CI/CD: Интеграция с системами непрерывной интеграции и доставки для автоматического запуска тестов при обновлении кода.
- Расширение Области Тестирования: Включение дополнительных модулей и функциональности OpenCart в область тестирования для более комплексной верификации системы.
- Применение ML и AI для Тестирования: Исследование возможности применения машинного обучения и искусственного интеллекта для предсказания потенциальных ошибок и оптимизации тестовых сценариев.

Таким образом, данная работа не только успешно достигла поставленных целей, но и открыла новые перспективы для дальнейших исследований в области тестирования API и развертывания серверных приложений.

Список литературы

1. Маршал А. Видеокурс по тестированию API [Электронный ресурс]. – Режим доступа: https://www.youtube.com/playlist?list=PLZqgWWF4O-zg03RGSZ2GpHLE3BmO8bjKo. – Дата доступа: 1.03.2023.
2. Куликов С. Тестирование программного обеспечения. Базовый курс [Текст] / С. Куликов. – М.: Издательство "Инфра-М", 2015. – 256 с.
3. Блэк Р. Ключевые процессы тестирования [Текст] / Р. Блэк. – М.: Издательство "Лори", 2006. – 544 с.
4. Копланда Л. A Practitioner's Guide to Software Test Design [Текст] / L. Copeland. – Boston: Artech House, 2004. – 279 p.
5. Савин Р. Тестирование Дот Ком, или Пособие по жестокому обращению с багами в интернет-стартапах [Текст] / Р. Савин. – М.: Издательство "Ridero", 2017. – 312 с.
6. Канер С., Фолк Д., Нгуен Е. К. Тестирование программного обеспечения. Фундаментальные концепции менеджмента бизнес-приложений - М.: Издательство «ДиаСофт», 2001. - 544 с;
7. Postman. Официальная документация [Электронный ресурс]. – Режим доступа: https://www.postman.com/. – Дата доступа: 17.03.2023.
8. Паттон Р. Software Testing [Текст] / R. Patton. – Indianapolis: Sams Publishing, 2006. – 408 p.
9. OpenCart. Официальная документация [Электронный ресурс]. – Режим доступа: https://www.opencart.com/. – Дата доступа: 15.03.2023.
10. Rubin J. Handbook of Usability Testing: How to Plan, Design, and Conduct Effective Tests [Текст] / J. Rubin, D. Chisnell. – 2nd ed. – Indianapolis: Wiley Publishing, 2008. – 384 p.
11. Docker. Официальная документация [Электронный ресурс]. – Режим доступа: https://www.docker.com/. – Дата доступа: 24.04.2023.
12. Fewster M. Software Test Automation [Текст] / M. Fewster, D. Graham. – Boston: Addison-Wesley, 1999. – 608 p.

Приложение 1

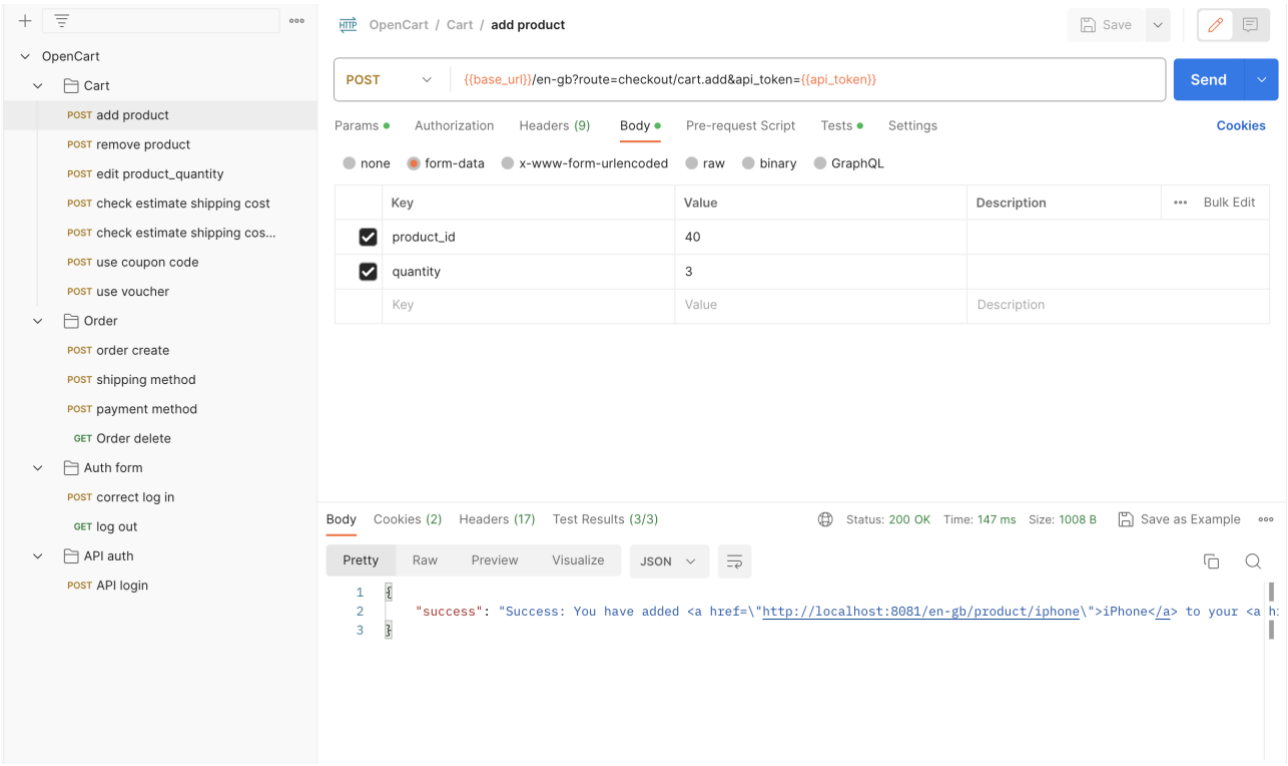

Рисунок 1 - демонстрация коллекции тестов

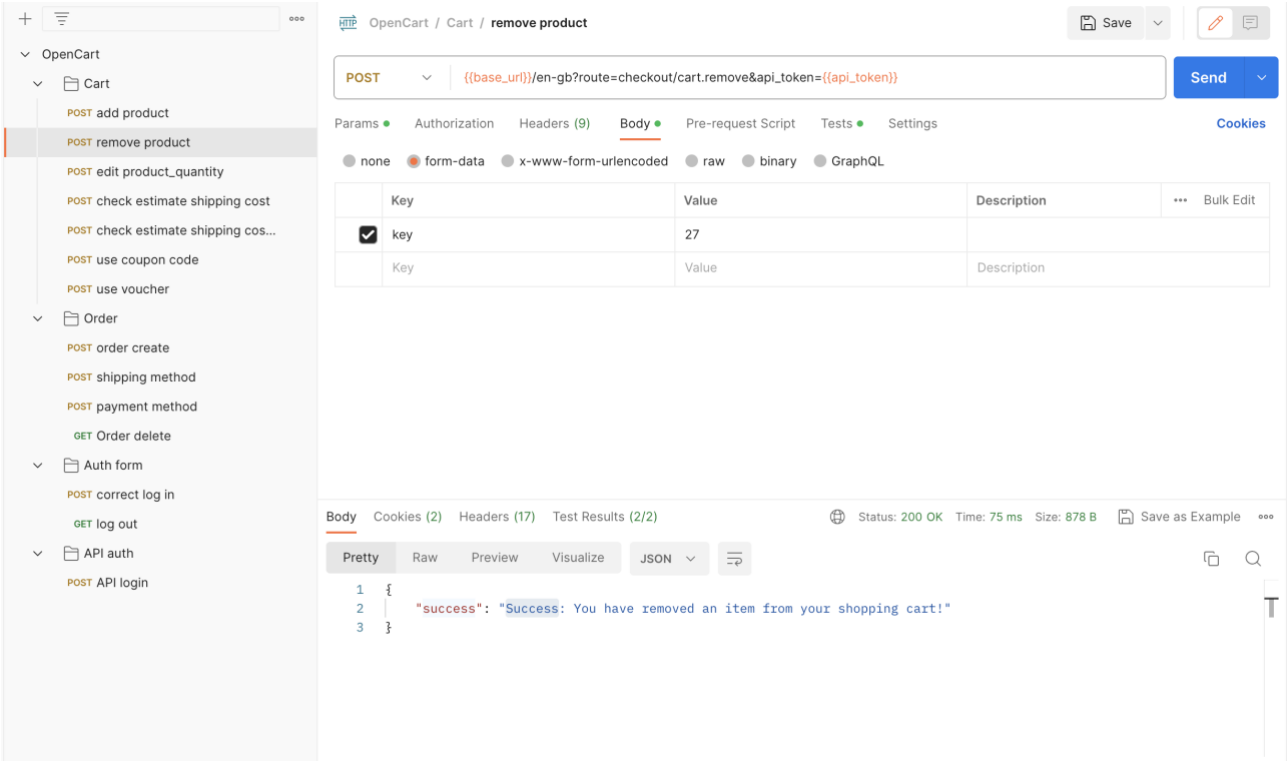

Рисунок 2 - демонстрация коллекции тестов

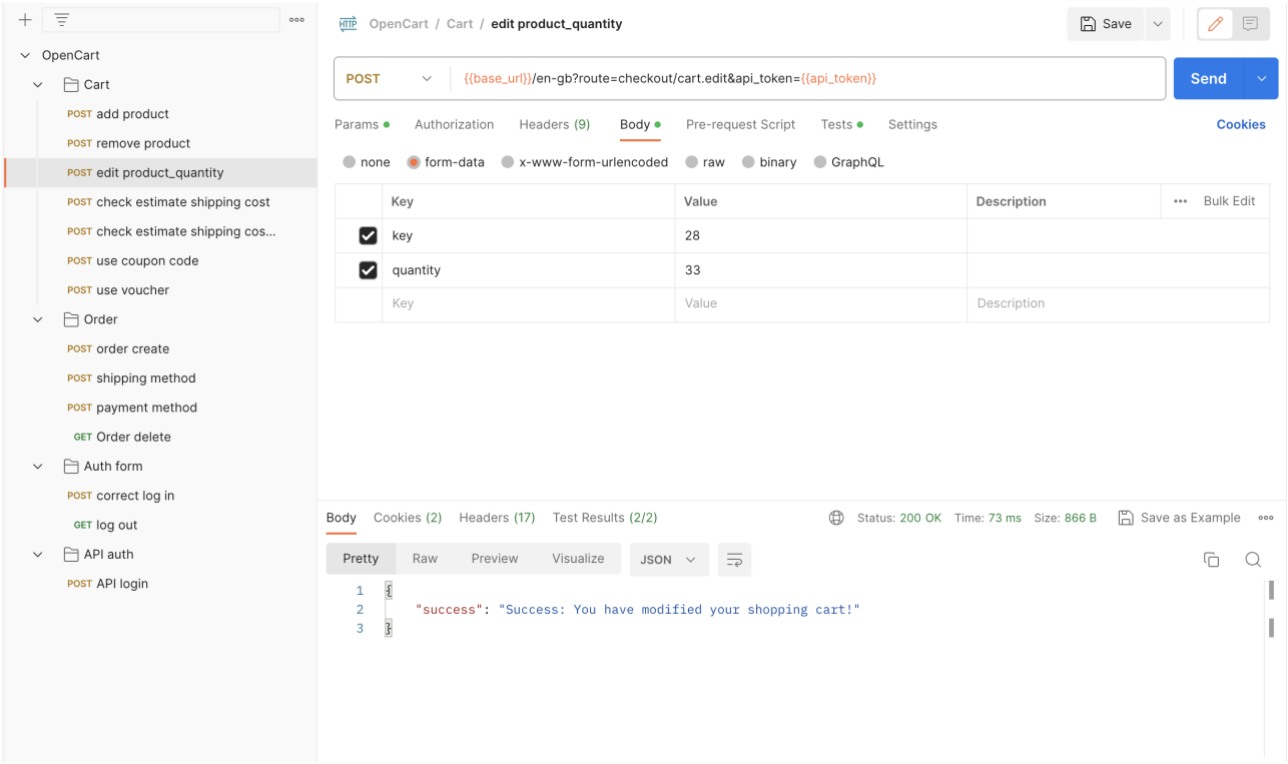

Рисунок 3 - демонстрация коллекции тестов

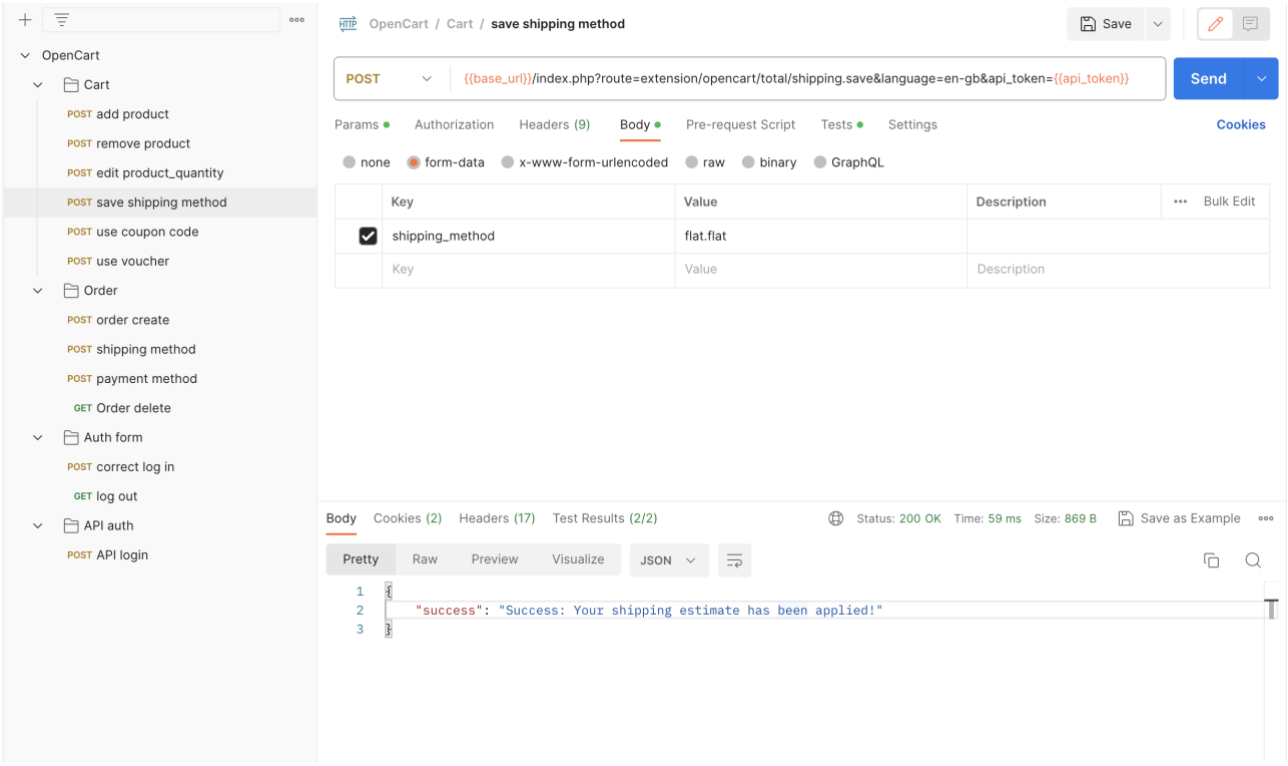

Рисунок 4 - демонстрация коллекции тестов

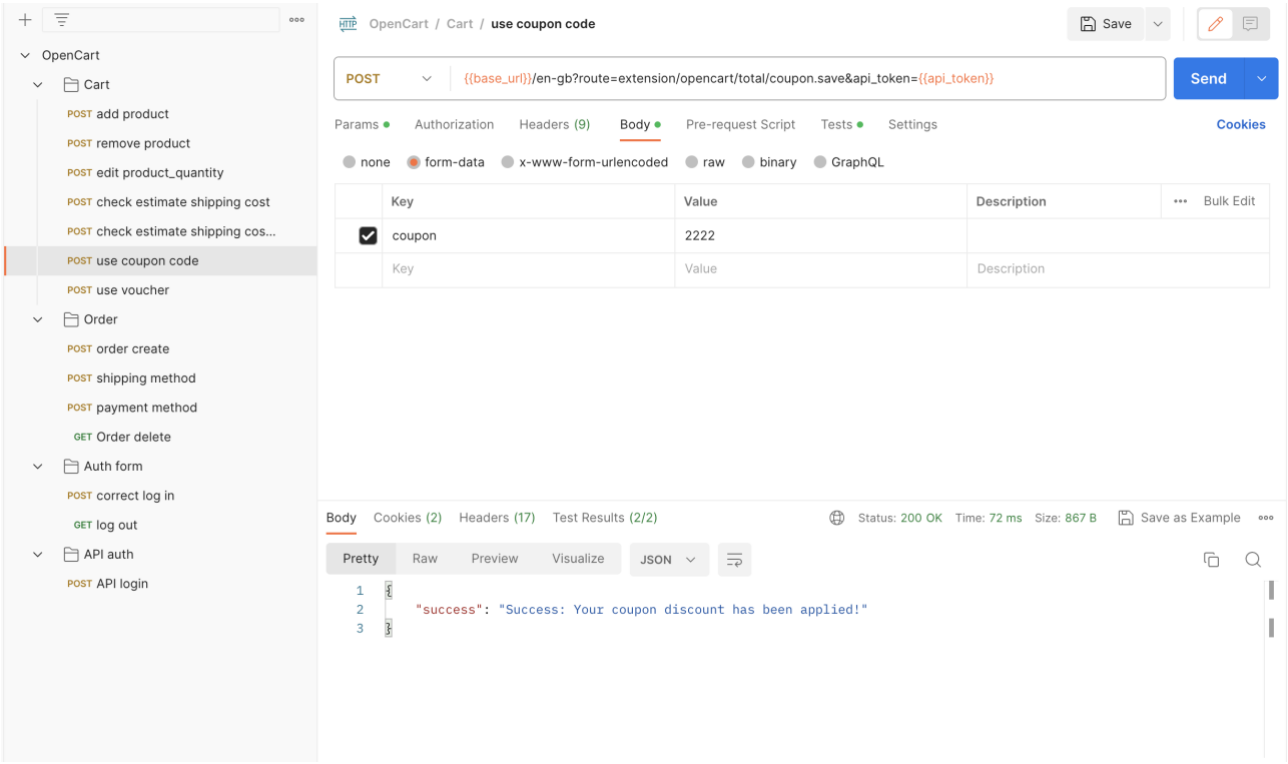

Рисунок 5 - демонстрация коллекции тестов

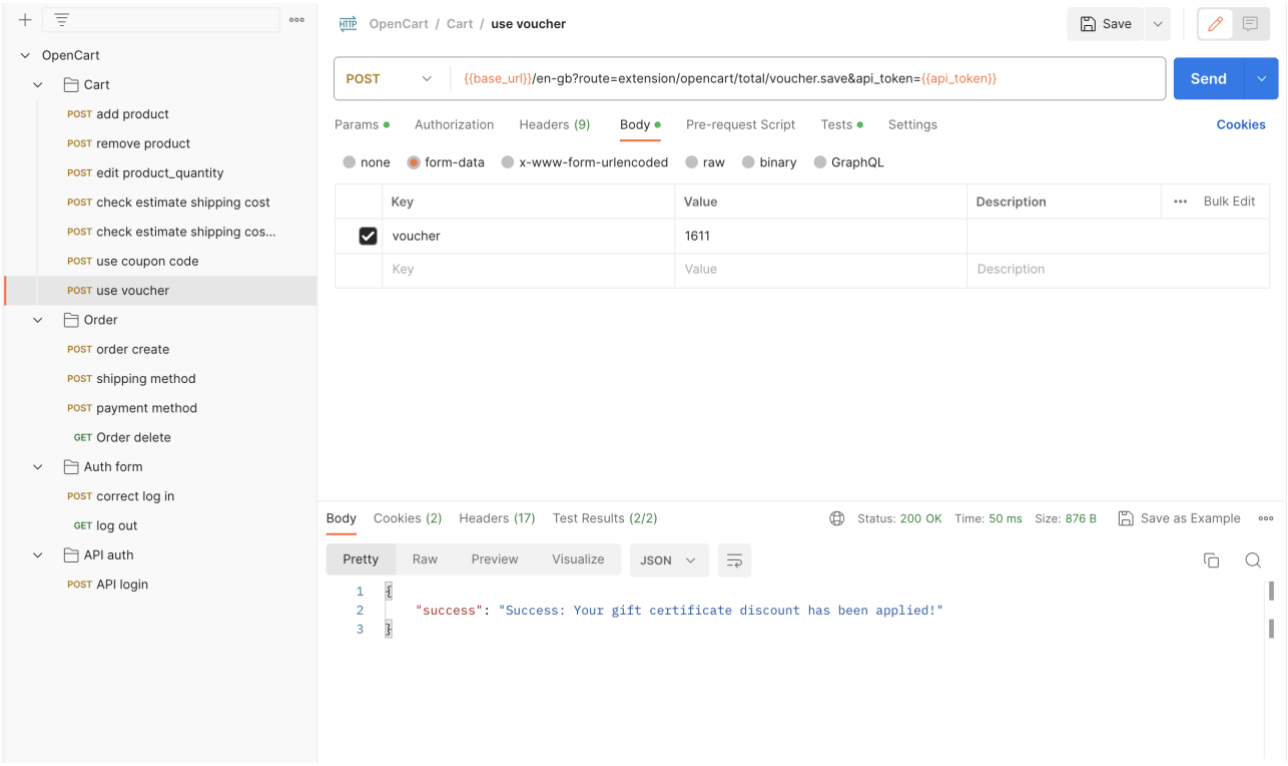

Рисунок 6 - демонстрация коллекции тестов

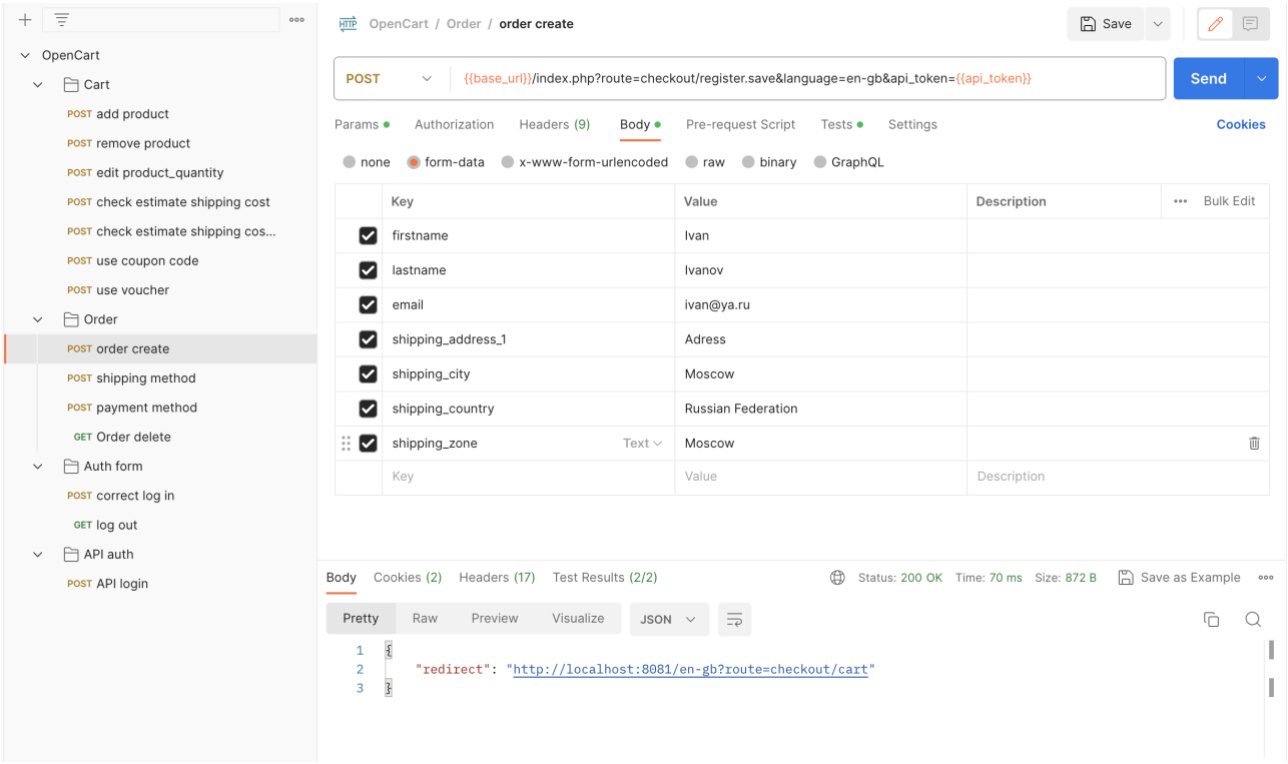

Рисунок 7 - демонстрация коллекции тестов

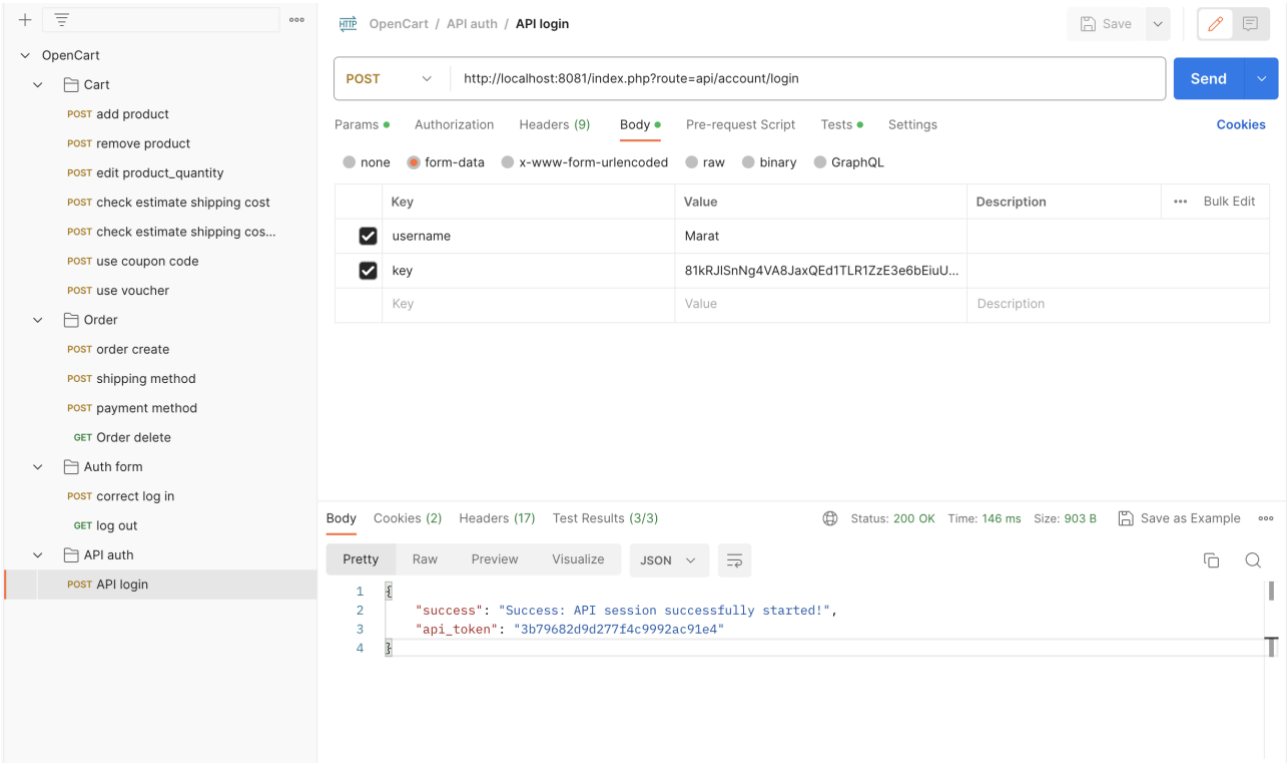

Рисунок 8 - демонстрация коллекции тестов

Приложение 2

```
/*Содержимое файла docker-compose.yml*/
/*Указывает версию синтаксиса файла docker-compose*/
version: '3'
services:
  /*Определяет контейнер для базы данных MariaDB*/
  mariadb:
    /*Указывает Docker-образ для MariaDB*/
    image: bitnami/mariadb:10.11.2-debian-11-r24'
    /*Переменные окружения для конфигурации MariaDB.*/
    environment:
      - ALLOW_EMPTY_PASSWORD=yes
      - MARIADB_USER=bn_opencart
      - MARIADB_DATABASE=bitnami_opencart
    /*Монтирует тома для хранения данных MariaDB*/
    volumes:
      - 'mariadb_data:/bitnami/mariadb'
    /*Отображает порты контейнера на хост-машину*/
    ports:
      - '3306:3306'

  /*Определяет контейнер для OpenCart*/
  opencart:
    /*Указывает Docker-образ для OpenCart*/
    image: docker.io/bitnami/opencart:4.0.2-1-debian-11-r7
    /*Отображает порты контейнера на хост-машину*/
    ports:
      - '8081:8080'
      - '443:8443'
    /*Переменные окружения для конфигурации OpenCart*/
    environment:
      - OPENCART_HOST=localhost:8081
      - OPENCART_DATABASE_HOST=mariadb
      - OPENCART_DATABASE_PORT_NUMBER=3306
      - OPENCART_DATABASE_USER=bn_opencart
```

```
      - OPENCART_DATABASE_NAME=bitnami_opencart
      - ALLOW_EMPTY_PASSWORD=yes
    /*Монтирует тома для хранения данных OpenCart*/
    volumes:
      - 'opencart_data:/bitnami/opencart'
      - 'opencart_storage_data:/bitnami/opencart_storage/'
    /*Указывает зависимости от других сервисов*/
    depends_on:
      - mariadb

/*Определяет тома для постоянного хранения данных*/
volumes:
  mariadb_data:
    driver: local
  opencart_data:
    driver: local
  opencart_storage_data:
    driver: local
```

Приложение 3

Приложение содержит серию иллюстраций, демонстрирующих работу Docker.

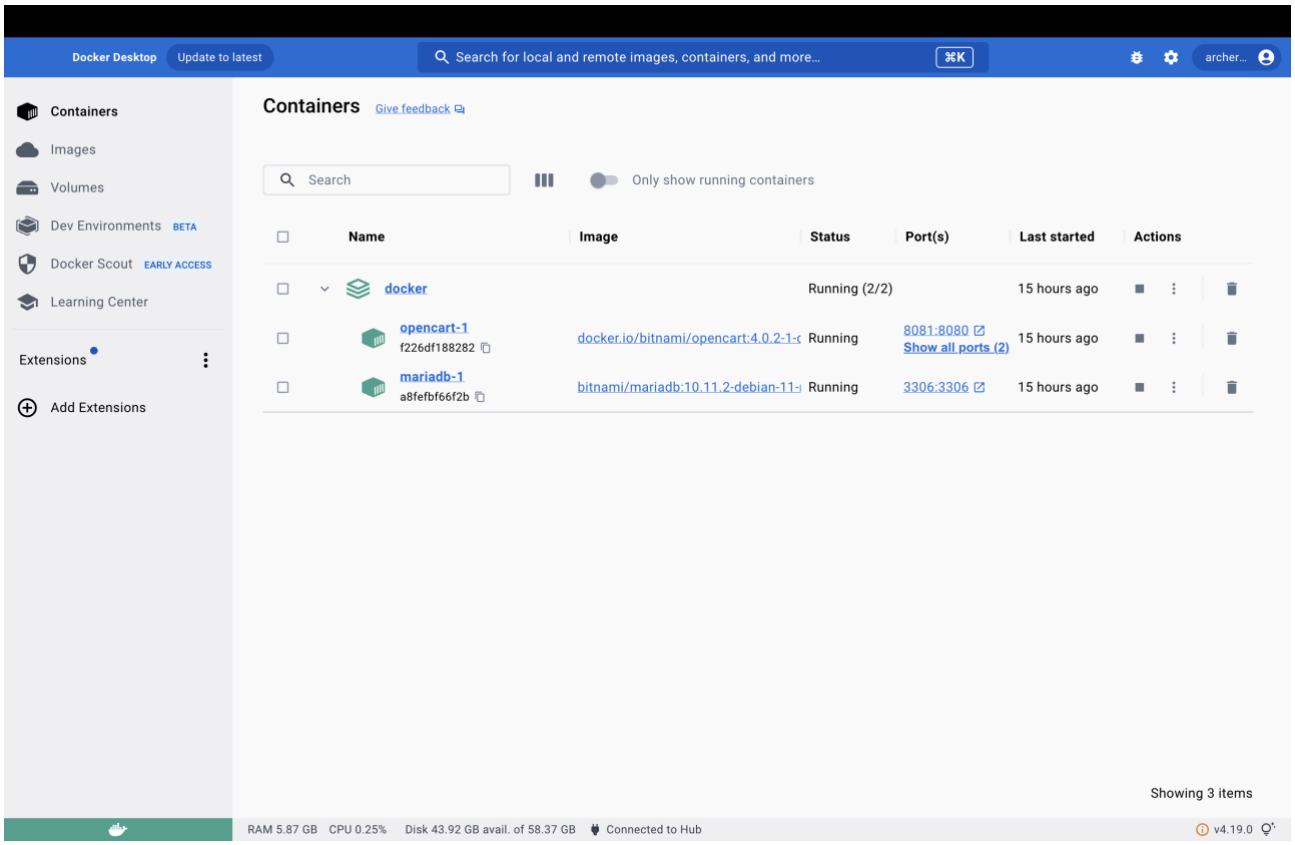

Рисунок 9 - главный экран Docker

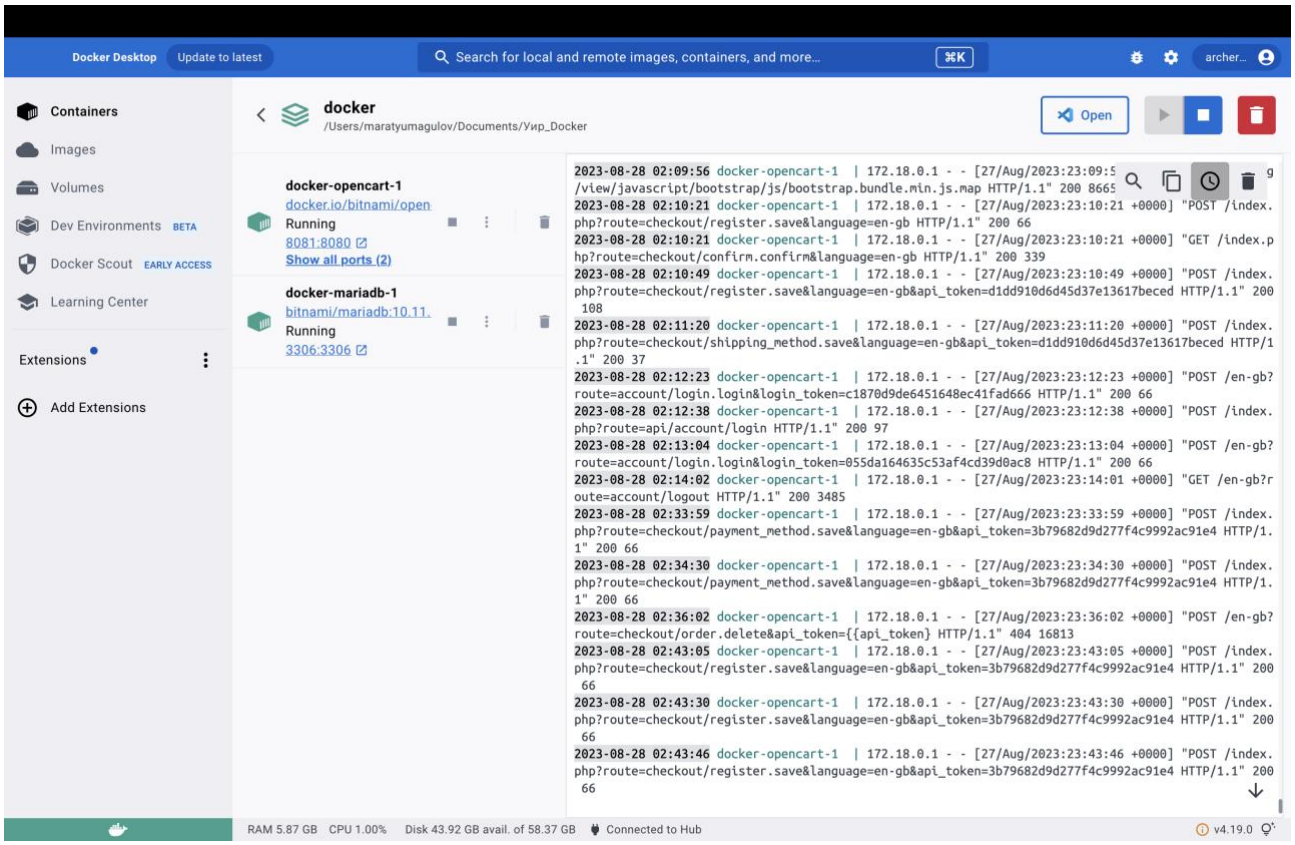

Рисунок 10 - демонстрация логов запросов