МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение

высшего образования

«МИРЭА – Российский технологический университет»

РТУ МИРЭА

Институт Информационных технологий

Кафедра Инструментального и прикладного программного обеспечения

ПРАКТИЧЕСКАЯ РАБОТА №4

по дисциплине «Разработка серверных частей интернет-ресурсов»

Студент группы ИКБО-10-19 Буланов В.А.

_____

(подпись студента)

Руководитель практической работы преподаватель Волков М.Ю.

_____

(подпись руководителя)

Работа представлена «____»_____ 2021 г.

Допущен к работе «___» _____ 2021 г.

Москва 2021

Оглавление

Цель работы

Получить навыки работы с интерфейсами прикладного программирования, а также создать собственное API.

Задание

Предполагается реализация интерфейса прикладного программирования для доступа к некоторым данным по варианту. Предполагается реализация серверной части обработки запросов и тестирование данного интерфейса с использованием программы Postman. Для реализации данного сервиса предлагается использовать серверную конфигурацию, модернизированную в течение первых трех практических работ. Важной частью данной практической работы является сохранение функциональности, реализованной в практической работе №3. То есть интерфейс предлагается создать уже в существующем веб-приложении. Также предполагается использование темы практической работы №3 для продолжения модернизирования собственной системы. Изменение темы согласовывается отдельно с преподавателем. Хранение данных предполагается уже в существующей базе данных.

Технические требования к реализации интерфейса:

1. Доступ как минимум к 2 независимым сущностям
2. Реализация как минимум операций группы CRUD (создание, чтение, обновление, удаление). Приветствуется реализация дополнительной функциональности
3. Тестирование всех функциональных возможностей созданного интерфейса с использованием программы Postman.

Ход работы

Для реализации интерфейса были выбраны сущности пользователя и товаров магазина компьютерных комплектующих. Для каждой сущности были реализованы операции создания (см. Рисунок 1 и Рисунок 2), чтения (см.

Рисунок 3 и Рисунок 4), обновления (см. Рисунок 5 и Рисунок 6) и удаления (см. Рисунок 7 и Рисунок 8).

По созданному интерфейсу была сгенерирована спецификация (см. Рисунок 9).

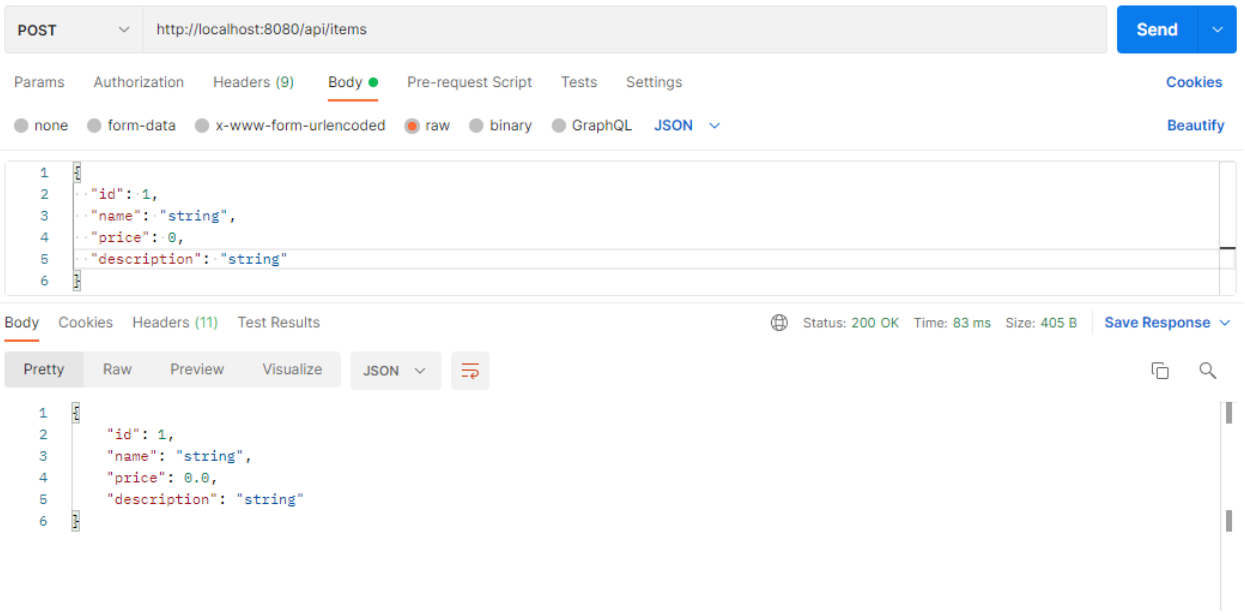

Рисунок 1 POST-запрос по товарам

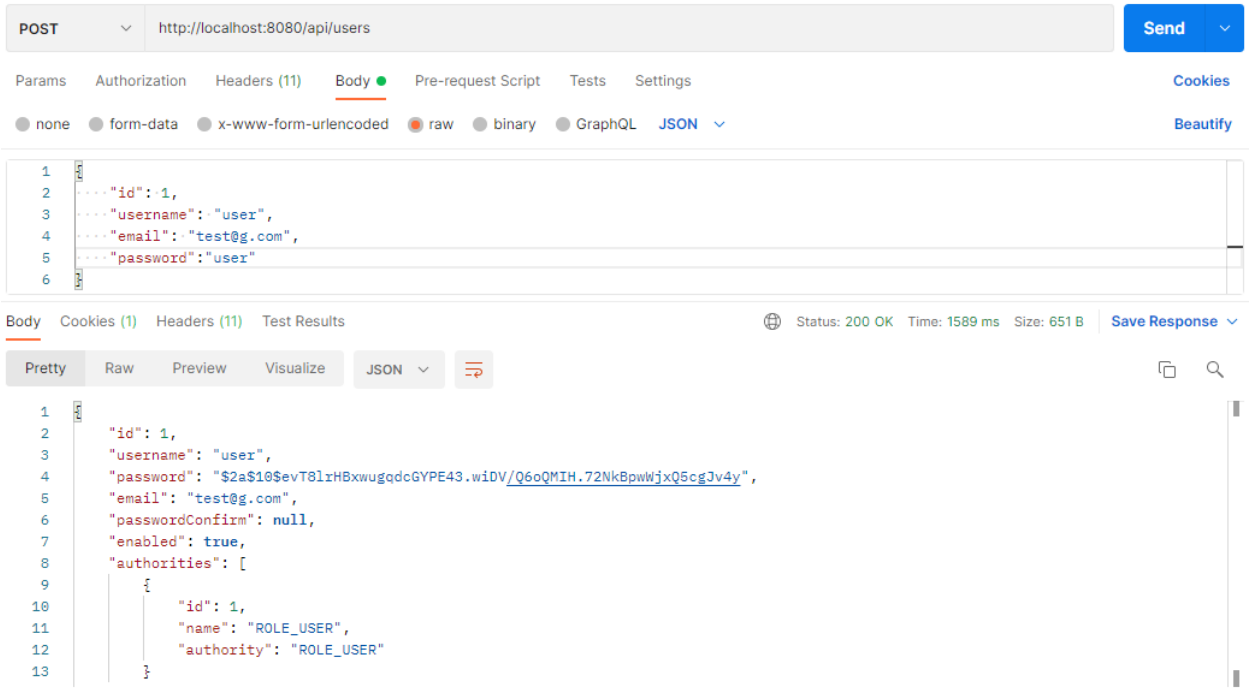

Рисунок 2 POST-запрос по пользователям

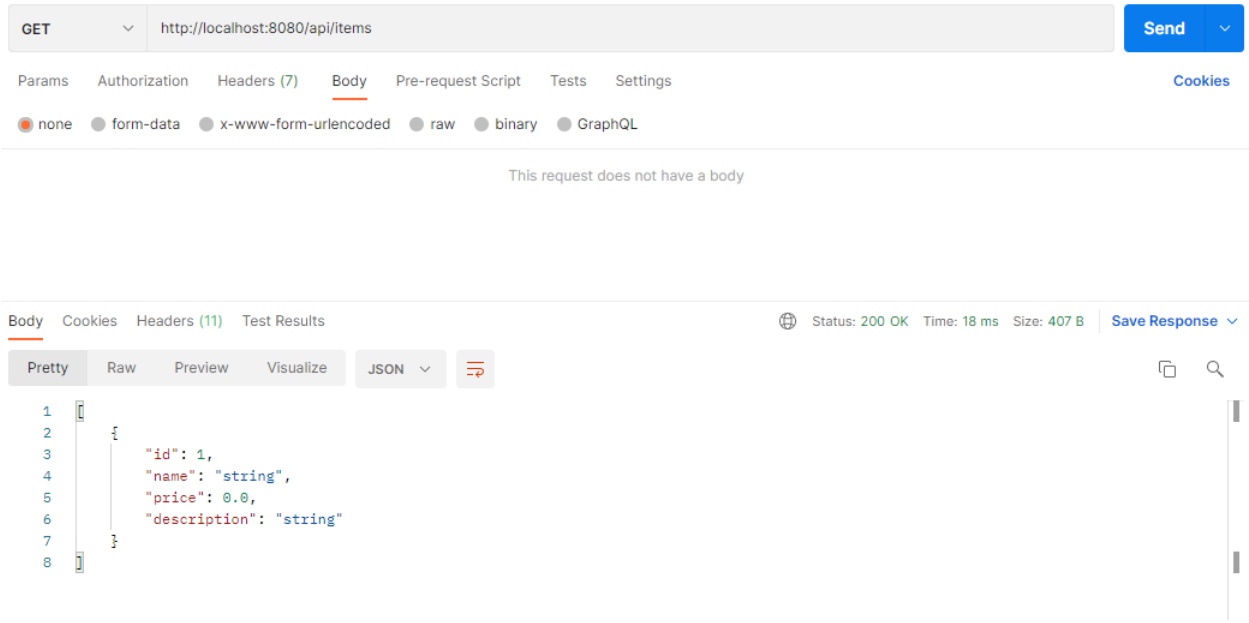

Рисунок 3 GET-запрос по товарам

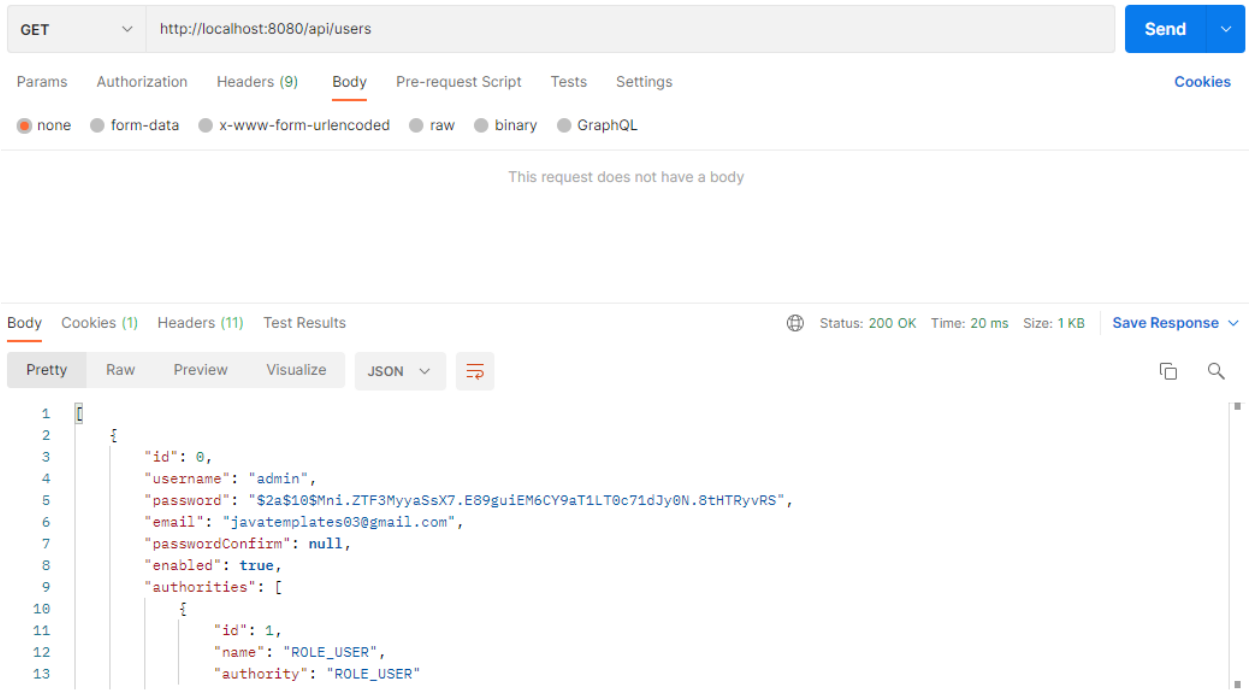

Рисунок 4 GET-запрос по пользователям

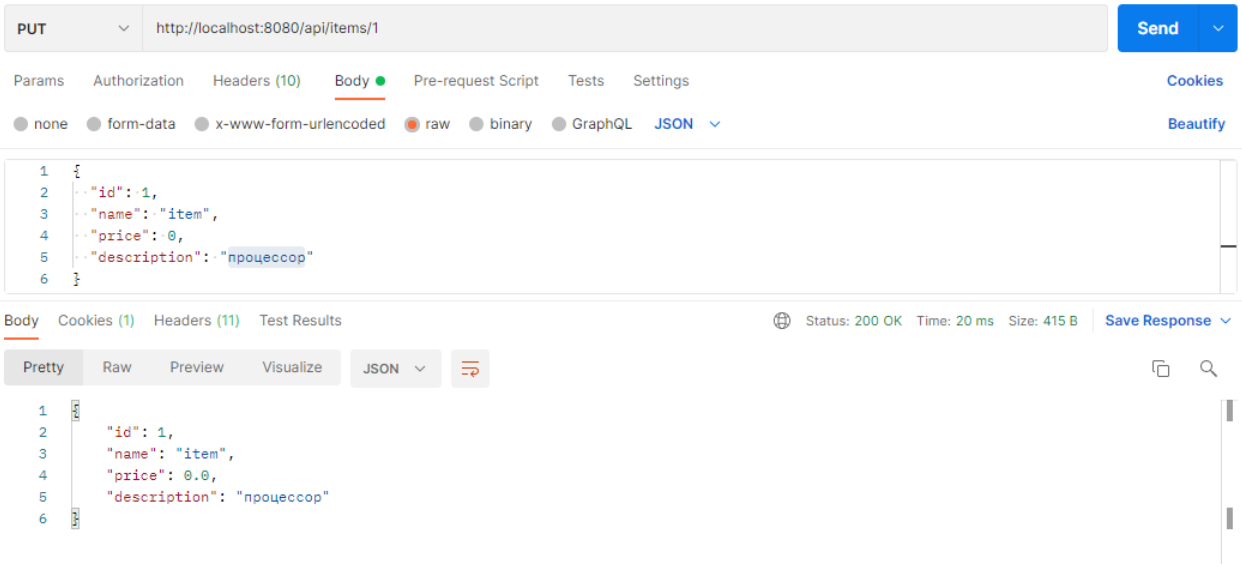

Рисунок 5 PUT-запрос по товарам

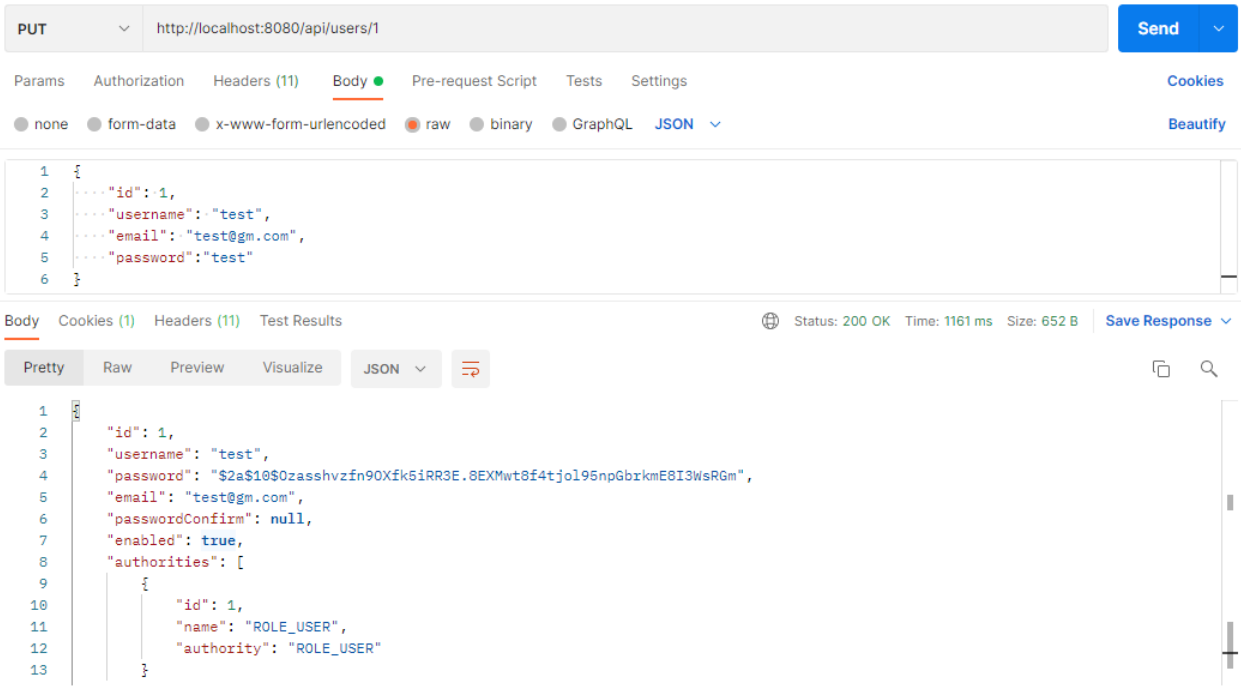

Рисунок 6 PUT-запрос по пользователям

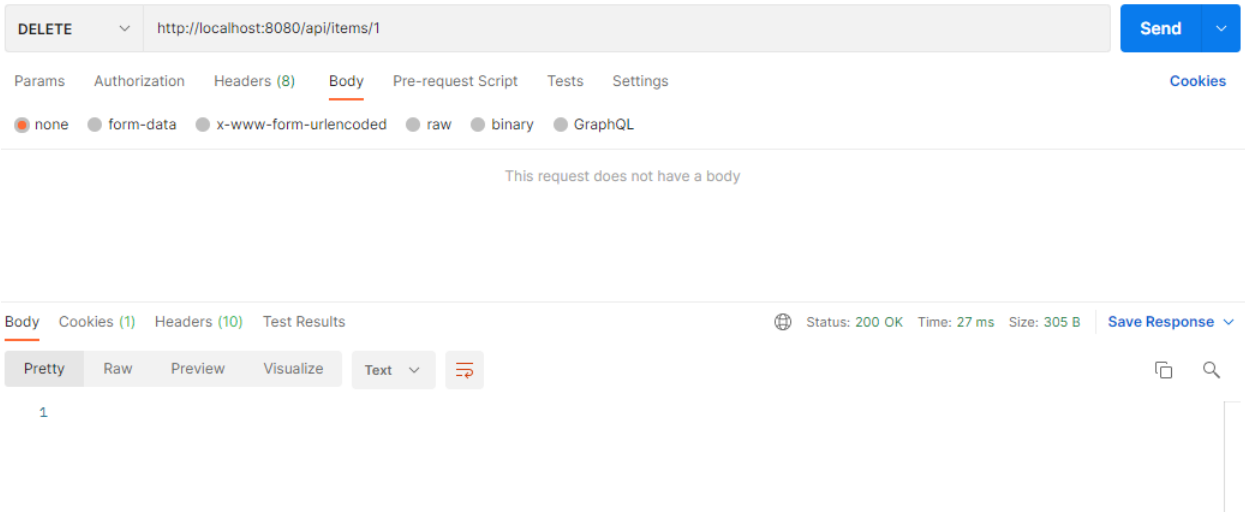

Рисунок 7 DELETE-запрос по товарам

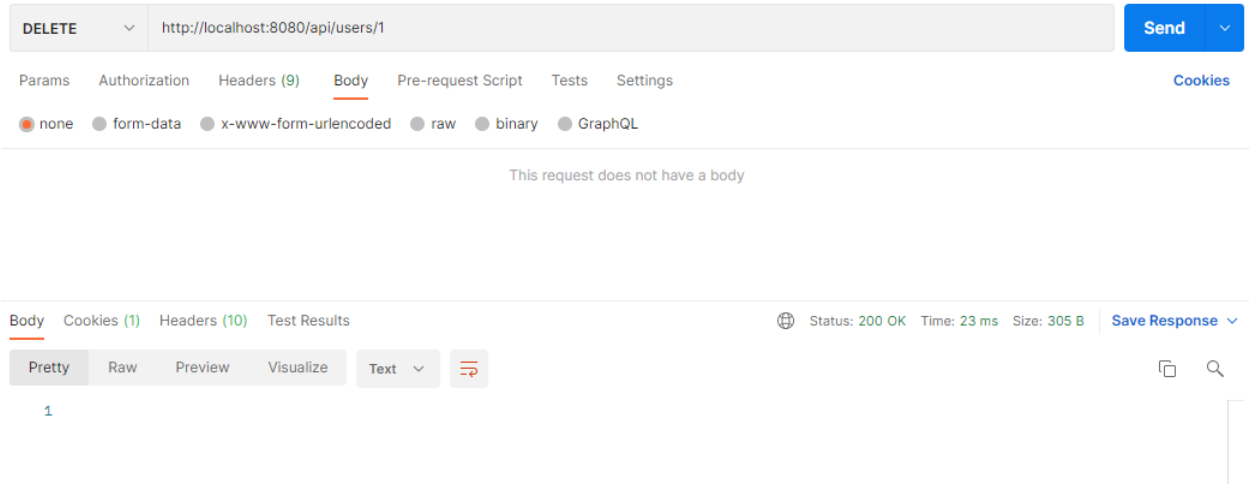

Рисунок 8 DELETE-запрос по пользователям

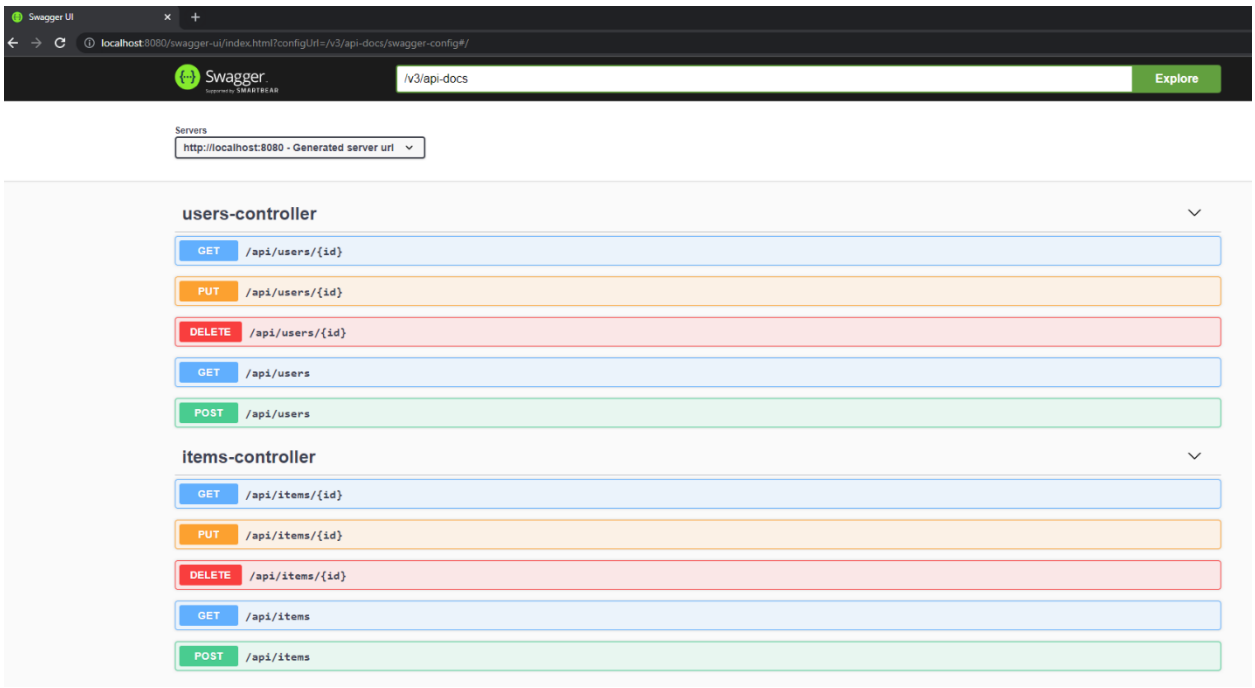

Рисунок 9 HTML-страница со спецификацией

Вывод

В ходе выполнения работы были получены навыки по созданию REST API и были реализованы операции группы CRUD.

Ответы на вопросы к практической работе

1. Что такое HTTP-запрос?

Запрос — это сообщение, посылаемое клиентом серверу.
Первая строка этого сообщения включает в себя метод, который должен быть применен к запрашиваемому ресурсу, идентификатор ресурса и используемую версию протокола.

2. Опишите существующие HTTP-запросы.

Тип HTTP-запроса (также называемый HTTP-метод) указывает серверу на то, какое действие мы хотим произвести с ресурсом. Изначально предполагалось, что клиент может хотеть от ресурса только одно — получить его, однако сейчас по протоколу HTTP можно создавать посты, редактировать профиль, удалять сообщения и многое другое. И эти действия сложно объединить термином «получение».

Для разграничения действий с ресурсами на уровне HTTP-методов и были придуманы следующие варианты:

- GET — получение ресурса
- POST — создание ресурса
- PUT — обновление ресурса
- DELETE — удаление ресурса

3. Опишите обработку запроса на PHP. Что нужно использовать, как вычленить параметры запроса?

4. Опишите создание HTML-форм на PHP.

Каждый элемент формы автоматически становится доступным вашим программам на PHP.

Когда происходит отправка данных формы PHP-скрипту, информация из этой формы автоматически становится доступной ему. Есть только два способа получить доступ к данным из форм HTML. GET-форма, за исключением того, что вместо POST, вам нужно будет использовать соответствующую предопределённую переменную GET.

5. Что такое API?

Каждый раз, когда пользователь посещает какую-либо страницу в сети, он взаимодействует с API удалённого сервера. API — это составляющая часть сервера, которая получает запросы и отправляет ответы.

API (программный интерфейс приложения, интерфейс прикладного программирования) (англ. application programming interface, API [эй-пи-ай]) — описание способов (набор классов, процедур, функций, структур или констант), которыми одна компьютерная программа может взаимодействовать с другой программой.

6. Опишите API как средство интеграции приложений.

Если программу (модуль, библиотеку) рассматривать как чёрный ящик, то API — это набор «ручек», которые доступны пользователю данного ящика и которые он может вертеть и дёргать.

Программные компоненты взаимодействуют друг с другом посредством API. При этом обычно компоненты образуют иерархию — высокоуровневые компоненты используют API низкоуровневых компонентов, а те, в свою очередь, используют API ещё более низкоуровневых компонентов.

По такому принципу построены протоколы передачи данных по Интернету. Стандартный стек протоколов (сетевая модель OSI) содержит 7 уровней (от физического уровня передачи бит до уровня протоколов приложений, подобных протоколам HTTP и IMAP). Каждый уровень пользуется функциональностью предыдущего («нижележащего») уровня передачи данных и, в свою очередь, предоставляет нужную функциональность следующему («вышележащему») уровню.

Понятие протокола близко по смыслу к понятию API. И то, и другое является абстракцией функциональности, только в первом случае речь идёт о передаче данных, а во втором — о взаимодействии приложений.

API библиотеки функций и классов включает в себя описание сигнатур и семантики функций.

7. Что такое Web API?

Веб-API — это интерфейс прикладного программирования для веб-сервера или веб-браузера. Это концепция веб-разработки, обычно ограниченная клиентской стороной веб-приложения (включая любые используемые веб-фреймворки), и поэтому обычно не включает детали реализации веб-сервера или браузера, такие как SAPI или API, если они не доступны для общего доступа через удаленное веб-приложение.

8. Приведите пример API.

С помощью специального API-протокола в социальных сетях Facebook, «ВКонтакте» и на других платформах пользователь получает возможность использовать упрощённый доступ к продуктам компании, пройдя быструю регистрацию на сайте через авторизацию своего аккаунта.

Одним из примеров API в интернет-рекламе является приложение, которое использует «Яндекс.Директ». Для настройки и более эффективного управления рекламными кампаниями создаются специальные модули, которые позволяют улучшить параметры поисковой оптимизации (SEO) путём информационного взаимодействия.

9. Что такое REST?

REST (от англ. Representational State Transfer — «передача репрезентативного состояния» или «передача "самоописываемого" состояния») — архитектурный стиль взаимодействия компонентов распределённого приложения в сети. Другими словами, REST - это набор правил о том, как программисту организовать написание кода серверного приложения, чтобы все системы легко обменивались данными и приложение можно было масштабировать. REST представляет собой согласованный набор ограничений, учитываемых при проектировании распределённой гипермедиа-системы.

10. Как организована передача данных в архитектуре REST?

В общем случае REST является очень простым интерфейсом управления информацией без использования каких-то дополнительных внутренних прослоек. Каждая единица информации однозначно определяется глобальным идентификатором, таким как URL. Каждая URL в свою очередь имеет строго заданный формат.

А теперь тоже самое более наглядно:

Отсутствие дополнительных внутренних прослоек означает передачу данных в том же виде, что и сами данные. Т.е. мы не заворачиваем данные в XML,

как это делает SOAP и XML-RPC, не используем AMF, как это делает Flash и т.д. Просто отдаем сами данные.

11.Как организована работа REST?

REST API используют запросы HTTP для выполнения стандартных функций базы данных, таких как создание, чтение, обновление и удаление записей (так называемые функции CRUD). Например, REST API может использовать запрос GET для получения записи, запрос POST для создания записи, запрос PUT для обновления записи и запрос DELETE для удаления записи. В вызовах API поддерживаются все методы HTTP. Хорошо продуманный REST API можно сравнить с веб-сайтом, который работает в веб-браузере со встроенной поддержкой HTTP.

Состояние ресурса в любой момент времени («отметка времени») называется представлением ресурса. Эта информация может быть предоставлена клиенту практически в любом формате, включая JavaScript Object Notation (JSON), HTML, XLT, Python, PHP или текстовом формате. Популярность JSON обусловлена тем, что данный формат не зависит от языка программирования и понятен как человеку, так и компьютеру.

Важную роль в вызовах REST API также играют заголовки и параметры запросов, поскольку они содержат такую важную информацию, как метаданные, разрешения, универсальные коды ресурсов (URI), сведения о кэшировании, файлы cookie и многое другое. Заголовки запросов и ответов применяются в хорошо продуманных REST API вместе с обычными кодами состояния HTTP.

12.Что такое SOAP?

SOAP (от англ. Simple Object Access Protocol — простой протокол доступа к объектам) — протокол обмена структурированными сообщениями в распределённой вычислительной среде. Первоначально SOAP

предназначался в основном для реализации удалённого вызова процедур (RPC). Сейчас протокол используется для обмена произвольными сообщениями в формате XML, а не только для вызова процедур.

13. Чем SOAP отличается от REST?

REST и SOAP на самом деле не сопоставимы. REST — это архитектурный стиль. SOAP — это формат обмена сообщениями. Давайте сравним популярные реализации стилей REST и SOAP.

- Пример реализации RESTful: JSON через HTTP
- Пример реализации SOAP: XML поверх SOAP через HTTP

На верхнем уровне SOAP ограничивает структуры ваших сообщений, тогда как REST — это архитектурный подход, ориентированный на использование HTTP в качестве транспортного протокола.

- Специфика SOAP — это формат обмена данными. С SOAP это всегда SOAP-XML, который представляет собой XML, включающий:
  — Envelope (конверт) – корневой элемент, который определяет сообщение и пространство имен, использованное в документе,
  — Header (заголовок) – содержит атрибуты сообщения, например: информация о безопасности или о сетевой маршрутизации,
  — Body (тело) – содержит сообщение, которым обмениваются приложения,
  — Fault – необязательный элемент, который предоставляет информацию об ошибках, которые произошли при обработке сообщений. И запрос, и ответ должны соответствовать структуре SOAP.
- Специфика REST — использование HTTP в качестве транспортного протокола. Он подразумевает наилучшее использование функций, предоставляемых HTTP — методы запросов, заголовки запросов, ответы, заголовки ответов и т. д.

14.Для чего нужен SOAP-процессор?

SOAP определяет внешний элемент для всех SOAP-сообщений, которыми обмениваются Web-служба и ее клиент. Подобно тому, как элемент stylesheet в пространстве имен XSLT сообщает XSLT-процессору, что в действительности это XML - преобразование, элемент Envelope в SOAP пространстве имен показывает SOAP-процессору, что это XML - SOAP-сообщение. Тогда процессор сможет искать индивидуальную часть сообщения: обязательный элемент Body, который содержит действительный запрос, и факультативный элемент Header, который содержит элементы расширения.

15.Опишите общую структуру SOAP-сообщения.

SOAP-сообщение представляет собой XML-документ; сообщение состоит из трех
основных элементов: конверт (SOAP Envelope), заголовок (SOAP Header) и тело (SOAP Body).

16.Что такое и что содержит Конверт (SOAP Envelope)?

Является самым «верхним» элементом SOAP сообщения. Содержит корневой элемент
XML-документа. Описывается с помощью элемента Envelope с обязательным пространством имен

17.Что такое и что содержит Заголовок SOAP (SOAP Header)?

Первый прямой дочерний элемент конверта. Не обязательный. Заголовок кроме
атрибутов xmlns может содержать 0 или более стандартных атрибутов:

•encodingStyle
•actor (или role для версии 1.2)
•mustUnderstand
•relay

18. Что такое и что содержит Тело SOAP (SOAP Body)?

Элемент Body обязательно записывается сразу за элементом Header, если он есть в
сообщении, или первым в SOAP-сообщении, если заголовок отсутствует. В элемент Body
можно вложить произвольные элементы, спецификация никак не определяет их структуру.
Определен только один стандартный элемент, который может быть в теле сообщения -
Fault, содержащий сообщение об ошибке.

19. Опишите SOAP-сообщение с вложением.

Существуют ситуации, когда клиент и сервер должны обмениваться данными в
формате, отличном от текстового. Это могут быть данные в форматах каких-то приложений,
мультимедийные данные и т.п. С точки зрения обмена данными все нетекстовые данные
рассматриваются как данные в двоичных кодах.
Двоичные данные включаются в сообщение в виде «вложения». В 2000 году консорциумом W3C выпущена спецификация «SOAP-сообщения с вложениями» (SOAP with
Attachments) (http://www.w3.org/TR/SOAP-attachments/), опирающаяся на версию SOAP 1.1. В
спецификации описаны правила включения SOAP-сообщения в MIME-сообщение типа
multipart/related и правила пересылки его по протоколу HTTP.

20. Что такое graphql?

В двух словах, GraphQL это синтаксис, который описывает как запрашивать данные, и, в основном, используется клиентом для загрузки данных с сервера. GraphQL имеет три основные характеристики:

- Позволяет клиенту точно указать, какие данные ему нужны.
- Облегчает агрегацию данных из нескольких источников.
- Использует систему типов для описания данных.

21. Что такое Распознаватели (resolvers) в graphql?

Используя распознаватель GraphQL понимает, как и где получить данные, соответствующие запрашиваемому полю.

22. Из чего состоит экосистема graphql, что нужно, чтобы использовать данную технологию?

23. Что такое валидация данных и для чего она нужна?

Казалось бы, «невалидные» данные, не удовлетворяющие определённым ограничениям, могут вызвать сбой в работе программы. Но что это означает? Предположим, в каком-то месте программы возникает исключение при попытке преобразовать строку в число, если строка имеет некорректный формат. Разумеется, если исключение не будет нигде перехвачено, это может привести к аварийному завершению программы. Но это маловероятный сценарий развития событий. Скорее всего в каком-то месте сработает перехватчик, который либо выдаст пользователю какое-то сообщение об ошибке в программе, либо сделает запись в журнал ошибок, после чего программа постарается восстановиться от сбоя и продолжить работу. То есть даже если валидацию не выполнять, вполне вероятно, что ничего страшного не случится.

Но определённые негативные последствия у отсутствия валидации всё таки могут быть, давайте чуть подробнее рассмотрим, какие проблемы при этом могут возникнуть.

1. Невозможность восстановиться после сбоя. Не всегда программа способна «вернуть всё назад». Возможно, в процессе работы программа выполнила какие-то необратимые действия — удалила файл, отправила данные по сети, напечатала что-то на принтер, запустила резец станка и он частично произвёл обработку заготовки детали. Но даже если восстановление в принципе возможно, алгоритм восстановления может тоже содержать ошибки, и это иногда приводит к совсем печальным последствиям.
2. Дополнительная нагрузка на систему. Восстановление после сбоя — это лишняя работа. Вся работа, которая была выполнена до момента сбоя — тоже лишняя. А это означает дополнительную нагрузку на систему, которой можно избежать, если заранее проверить данные. С другой стороны, валидация — это тоже дополнительная нагрузка, причём восстановление приходится делать лишь изредка, а проверку надо выполнять каждый раз, так что ещё неизвестно, что выгоднее.
3. Инъекции не вызывают сбоев. Один из основных способов эксплуатации уязвимостей в программах заключается в том, чтобы «обмануть» валидаторы, то есть передать данные, которые валидатор признаёт корректными, но при этом они интерпретируются непредусмотренным образом, так что злоумышленник может получить несанкционированный доступ к данным или некоторым возможностям программы, либо способен разрушить данные или программу. Если валидации нет вообще, задача злоумышленника максимально упрощается.
4. Сложность идентификации причины проблемы. Если исключение вылетело откуда-то из глубины программы, определить причины его возникновения не так-то просто. И даже если это возможно, может оказаться нелегко объяснить пользователю, что сбой вызван данными, которые он ввёл некоторое время назад в каком-то совершенно другом месте программы. А если проверка выполнена немедленно после ввода

данных, никаких сложностей с идентификацией источника проблемы не возникает.

24.Где и когда выполнять валидацию данных?

Как уже было сказано выше, с точки зрения уменьшения нагрузки лучше всего вообще не выполнять валидацию данных.

Но если всё-таки проверка нужна, логика подсказывает, что удобно проверять данные в том месте, где они попадают в программу из внешнего мира. После такой проверки можно быть уверенным, что в программу попадают правильные данные и в дальнейшем они могут использоваться без дополнительных проверок.Это может быть пользовательский интерфейс, через который человек вводит данные. Это может быть файл, содержащий настройки программы или данные, которые программа должна обработать. Это может быть база данных, в которую информация может попадать из других программ. Это может быть сетевой протокол обмена данными с другими программами. Наконец, это может быть программный интерфейс, который использует другая программа, вызывая некоторые функции/процедуры и передавая в них параметры.

Увы, здравый смысл иногда вынужден отступить перед натиском действительности. «Фейс-контроль» данных на входе иногда не просто нецелесообразен, но вообще невозможен. Ниже приведены некоторые причины этого.

1. Для валидации требуется доступ к недоступной части состояния системы. Это особенно характерно для проверки данных, вводимых человеком через графический интерфейс пользователя. Современные приложения часто построены с использованием многоуровневой архитектуры, которая предполагает, что реализация пользовательского интерфейса выделена в презентационный слой, а для проверки требуется доступ к другим слоям, вплоть до слоя базы данных.

Особенно хорошо это заметно для веб-приложений, где пользовательский интерфейс реализуется в браузере и выполняется на стороне клиента, а для проверки ввода требуется сравнение с тем, что хранится в базе данных. В этой ситуации проверку приходится выполнять уже после отправки данных на сервер. (Впрочем, сейчас с появлением AJAX-технологии эта проблема частично решена).

2. Валидация требует полностью повторить логику обработки. Как уже отмечено двумя абзацами выше, при многослойной архитектуре приложения пользовательский интерфейс обычно выделяется в специальный презентационный слой, а логика обработки данных находится на другом слое. И бывают такие ситуации, когда для валидации нужно практически полностью выполнить эту обработку, потому не существует более короткого способа понять, завершится она успехом или нет.

25. Как выполнять валидацию данных?

Впрочем, где бы ни выполнялась валидация, можно это делать несколькими различными способами, в зависимости от того, какие ограничения накладываются на данные.

1. Посимвольная проверка. Как правило такие проверки выполняются в пользовательском интерфейсе, по мере ввода данных. Но не только. Например, лексический анализатор компилятора тоже выявляет недопустимые символы непосредственно в процессе чтения компилируемого файла. Поэтому такие проверки можно условно назвать «лексическими».
2. Проверка отдельных значений. Для пользовательского интерфейса это проверка значения в отдельном поле, причём выполняться она может как по мере ввода (проверяется то неполное значение, которое введено к настоящему моменту), так и после завершения ввода, когда поле теряет фокус. Для программного интерфейса (API) это проверка одного

из параметров, переданных в вызываемую процедуру. Для данных, получаемых из файла, это проверка какого-то прочитанного фрагмента файла. Такие проверки, опять-таки по аналогии с компиляторной терминологией, можно назвать «синтаксическими».

3. Совокупность входных значений. Можно предположить, что в программу сначала передаются какие-то данные, после чего подаётся некоторый сигнал, который инициирует их обработку. Например, пользователь ввёл данные в форму или в несколько форм (в так называемом «визарде») и наконец нажал кнопку «OK». В этот момент можно выполнить так называемые «семантические» проверки, нацеленные на валидацию не только отдельных значений, но и взаимосвязей между ними, взаимных ограничений.
   Вполне возможна ситуация, когда каждое отдельное значение «синтаксически» корректно, но вместе они образуют несогласованный набор. Для программного интерфейса эта разновидность валидации предполагает проверку набора входных параметров вызываемой процедуры, для случая получения данных из файла это проверка всех прочитанных данных.
4. Проверка состояния системы после обработки данных. Наконец, есть последний способ, к которому можно прибегнуть, если валидацию непосредственно входных данных выполнить не удаётся — можно попытаться их обработать, но оставить возможность вернуть всё к исходному состоянию. Такой механизм часто называется транзакционным.

26. Приведите пример с поэтапной валидацией данных.

27. Что такое запрос и мутация в graphql и чем они отличаются?

Запросы GraphQL — это сущности, представляющие собой запрос к серверу на получение неких данных. Например, у нас есть некий пользовательский интерфейс, который мы хотим заполнить данными. За этими данными мы и

обращаемся к серверу, выполняя запрос. При использовании традиционных REST API наш запрос принимает вид GET-запроса.

В то время как запросы GraphQL выполняют загрузку данных, мутации ответственны за внесение в данные изменений. Мутации могут быть использованы в виде базового механизма RPC (Remote Procedure Call, вызов удалённых процедур) для решения различных задач наподобие отправки данных пользователя API стороннего разработчика.

Список использованной литературы

1. Статья о назначении докера простыми словами: https://habr.com/ru/post/309556/
2. Более сложная и подробная статья про докер: https://habr.com/ru/post/277699/
3. Хорошая статья с пингвинами для прочтения после туториала по докеру: https://habr.com/ru/post/250469/
4. Статья о конкретном опыте использования докер контейнеров: https://habr.com/ru/post/247969/
5. Туториал по докеру: https://badcode.ru/docker-tutorial-dlia-novichkov-rassmatrivaiem-docker-tak-iesli-by-on-byl-ighrovoi-pristavkoi/
6. Туториал по докеру с Хабра: https://habr.com/ru/post/310460/
7. Документация по PHP: https://www.php.net/manual/ru/