1 Для чего и как мы тестируем веб-API

В этой главе

- ✓ Проблемы создания сложных API-платформ
- ✓ Значение и цель тестирования
- ✓ Как выглядит стратегия тестирования API и чем она может быть полезна.

Как мы можем гарантировать качество и ценность для конечных пользователей того, что создаем? Проблема, с которой мы сталкиваемся при выпуске высококачественного продукта, заключается в огромном количестве сложных действий и операций при его создании. Если мы хотим добиться высокого качества, необходимо преодолеть все сложности, а также разобраться, как работают наши системы и что пользователям требуется от них. Вот почему нам нужна стратегия тестирования, которая поможет лучше понять, что мы на самом деле создаем. Итак, прежде чем мы начнем наше путешествие по миру тестирования API, давайте сначала подумаем, почему программное обеспечение такое сложное и в чем польза тестирования.

1.1. ЧТО ПРОИСХОДИТ В ВАШИХ ВЕБ-API

В 2013 году правительство Великобритании разработало стратегию по переводу налоговой и таможенной службы (HMRC) на цифровой стандарт обслуживания. Цель стратегии состояла в том, чтобы перевести работу этих служб в онлайн-режим, улучшить качество услуг и сократить расходы.

К 2017 году налоговая платформа HMRC насчитывала более 100 цифровых услуг, созданных 60 группами в пяти различных центрах. Каждая из этих цифровых услуг поддерживается платформой взаимосвязанных веб-API, число которых постоянно растет. Количество API, созданных для поддержки этих сервисов, просто поражает. Когда я присоединился к проекту в 2015 году, услуг, команд и центров было примерно вдвое меньше, чем сейчас, но уже тогда платформа объединяла более 100 веб-API. С тех пор это число, несомненно, увеличилось. Возникает вопрос: как проект такого масштаба и сложности может предоставлять конечным пользователям высококачественные услуги?

Я привел этот пример, потому что он помогает выделить два уровня сложности при создании веб-API:

- Собственная сложность конкретного веб-API.
- Сложность, обусловленная взаимодействием нескольких веб-API, работающих на одной платформе.

Разобравшись с обеими, мы начнем понимать, зачем нужно тестирование и чем оно может нам помочь.

1.1.1. Собственная сложность веб-API

Может показаться немного банальным начать с вопроса: что такое веб-API? Но если мы потратим время на разбор структуры веб-API, то сможем узнать не только о том, что такое веб-API, но и в чем заключается его сложность. Рассмотрим, к примеру, представленную на рис. 1.1 визуализацию веб-API сервиса бронирования, который мы будем тестировать позже в этой книге.

На рисунке мы видим, что веб-API получает от клиентов данные о резервировании в форме HTTP-запросов, которые обрабатываются в API на разных уровнях. После завершения выполнения и сохранения резервирования веб-API отвечает через HTTP. Более детальное рассмотрение позволит понять, сколько всего происходит в рамках одного веб-API.

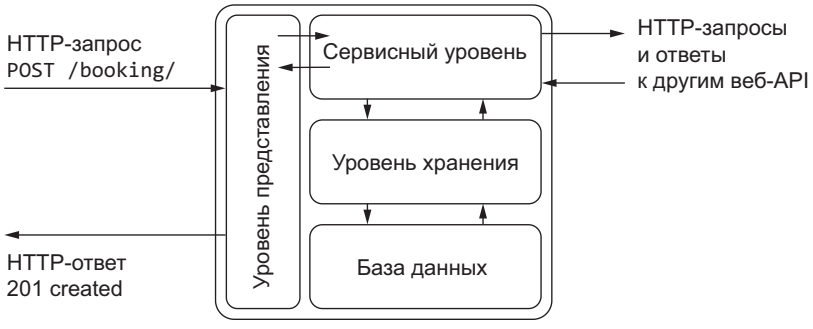

Рис. 1.1. Модель веб-API, изображающая его компоненты и то, как он работает

Прежде всего, на уровне представления HTTP-запрос на бронирование переводится в контент, который может быть прочитан другими уровнями. Сервисный уровень применяет к информации о бронировании бизнес-логику, например проверяет, действительно ли это бронирование и не конфликтует ли оно с другими бронированиями. Наконец, если обработанное бронирование необходимо сохранить, оно подготавливается на уровне хранения, а затем сохраняется в базе данных. Если все выполнено успешно, каждый уровень должен сообщить об этом, чтобы создать ответ, который веб-API отправит создателю запроса.

Каждый из рассмотренных слоев может быть построен по-разному, в зависимости от наших требований и вкусов. Например, можно разрабатывать веб-API с использованием архитектур REST, GraphQL или SOAP, каждая из которых имеет свои собственные шаблоны и правила.

Работа с REST

В этой книге мы будем работать преимущественно с веб-API, созданными с использованием архитектуры REST. Существует множество различных архитектурных стилей, но сегодня наиболее широко используется REST. Однако стоит отметить, что хотя GraphQL и SOAP имеют свои отличия, подходы к тестированию, которые мы рассмотрим, в равной степени применимы к этим типам архитектуры. На протяжении всей книги мы будем кратко отмечать, каким образом изучаемое может быть применено к любому архитектурному стилю.

Сервисный уровень содержит бизнес-логику, которая, в зависимости от контекста, может иметь множество настраиваемых правил. Аналогичный принцип

применяется к уровням хранения. Каждый из этих уровней опирается на зависимости, которые имеют свои собственные циклы разработки. Нам необходимо знать огромное количество информации, чтобы обеспечить высокое качество работы API.

Понимание того, что происходит в наших веб-API и как они помогают другим, требует времени и опыта. Да, можно достичь некоторого понимания, тестируя части по отдельности (к чему я всегда призываю команды; ознакомьтесь с выступлением Дж. Б. Райнсбергера (J. B. Rainsberger) «*Integrated Tests Are a Scam*», чтобы узнать больше: https://youtu.be/VDfX44fZoMc). Но это дает нам информацию только о части головоломки, а не обо всей задаче целиком.

1.1.2. Сложность взаимодействия многих веб-API

В случае с платформой HMRC с ее более чем 100 веб-API требуется понимание не только работы каждого из них, но и их взаимодействия друг с другом. Такие подходы, как микросервисная архитектура, помогают уменьшить сложность отдельных веб-API, делая их меньше и более конкретными. С другой стороны, при этом количество веб-API на платформе еще больше увеличивается. Как в этих условиях обеспечить актуальность наших знаний о платформе? И как следить за взаимодействиями каждого API с другими и контролировать, чтобы их соединения работали в пределах ожидаемого?

Чтобы создать высококачественный продукт, мы должны делать осознанный выбор. Это означает, что наши знания о работе веб-API, а также об их взаимодействиях как друг с другом, так и с конечными пользователями жизненно важны. Иначе мы рискуем столкнуться с проблемами из-за неверной интерпретации того, как работают наши системы. Именно с этой точки зрения тестирование важно для упорядочивания и сохранения понимания работы продуктов.

1.2. КАК ТЕСТИРОВАНИЕ ПОМОГАЕТ НАМ

Если мы как команда собираемся добиться успеха в тестировании, нам необходимо общее понимание его цели и значения. К сожалению, существует множество неправильных представлений о том, что такое тестирование. Чтобы помочь нам всем прийти к единому мнению, позвольте представить модель тестирования (рис. 1.2).

Рис. 1.2. Модель, которая помогает описать значение и цель тестирования

Эта модель, основанная на образе, созданном Джеймсом Линдсеем в его статье *«Why Exploration has a Place in any Strategy»* (http://mng.bz/o2vd), состоит из двух кругов. Левый круг соответствует нашим представлениям о продукте (воображению), а правый круг — реализации, то есть тому, что мы имеем в реальности. Цель тестирования — узнать как можно больше об этих кругах путем проведения тестов. Чем больше мы тестируем, тем больше продвигаемся в решении следующих задач:

- обнаруживаем потенциальные проблемы, которые могут повлиять на качество;
- разбираемся в том, что создаем, и приобретаем уверенность, что это именно тот продукт или услуга, которые мы хотим создать.

Чтобы лучше разобраться в этом, давайте рассмотрим пример, где команда предоставляет гипотетическую функцию поиска, качество которой мы должны обеспечить.

1.2.1. Представление

Круг представления отражает то, что мы хотим от нашего продукта, включая явные и неявные ожидания. В этом круге наше тестирование сосредоточено на том, чтобы узнать как можно больше об этих ожиданиях. Мы узнаем не только то, что было явно заявлено в письменной или устной форме, но также углубляемся в детали и устраняем двусмысленности в терминах и идеях.

В качестве примера допустим, что представитель бизнеса или пользователей (владелец продукта) сформулировал для своей команды требование: «Результаты поиска должны быть упорядочены по релевантности». Представленная явно информация говорит о том, что требуются результаты поиска, упорядоченные по релевантности. Однако мы можем раскрыть много неявной информации, проверяя идеи и концепции, лежащие в основе того, о чем нас просят. Чтобы разобраться в проблеме, можно задать следующие вопросы:

- Что подразумевается под *релевантными результатами*?
- Для кого результаты должны быть релевантны?
- Какая информация предоставляется?
- Как ее упорядочить по релевантности?
- Какие данные мы должны использовать?

Ответы на них дают более полное представление о том, чего от нас хотят, устраняют недопонимания в нашей команде и выявляют потенциальные риски, которые могут повлиять на результаты. Зная больше о том, что нас просят создать, мы с большей вероятностью построим то, что нужно, с первого раза.

1.2.2. Реализация

Проверяя представления, мы получаем более четкое понимание того, что нас просят построить. Но это не означает, что в итоге мы получим продукт, который соответствует ожиданиям. Именно поэтому мы также тестируем реализацию, чтобы узнать следующее:

- Соответствует ли продукт ожиданиям?
- В чем продукт может не оправдать ожиданий?

Обе цели одинаково важны. Мы хотим быть уверены, что создали правильный продукт, но побочные эффекты, такие как странности поведения и уязвимости, будут существовать всегда. На примере задачи о представлении результатов поиска мы могли бы не только проверить, что функция выдает их в требуемом порядке, но и спросить:

- Что, если я введу разные условия поиска?
- Что, если релевантные результаты различны для разных поисковых инструментов?

- Что делать, если во время поиска часть сервиса не работает?
- Что, если я запрошу результаты 1000 раз менее чем за 5 секунд?
- Что произойдет, если результатов не будет?

Выходя за рамки ожиданий, мы лучше понимаем происходящее в нашем продукте и его недостатки. Это гарантирует, что в итоге мы будем верно оценивать поведение нашего приложения и не выпустим некачественный продукт. Кроме того, если мы обнаружим неожиданное поведение, то сможем попытаться устранить ошибки или скорректируем наши ожидания.

1.2.3. Значение тестирования

Модель тестирования представлений и реализации демонстрирует, что тестирование выходит за рамки простого подтверждения ожиданий и бросает вызов нашим предположениям. Чем больше мы узнаем в процессе тестирования о том, что мы хотим построить и что мы создали, тем больше эти два круга совпадают друг с другом. А чем больше они совпадают, тем точнее становится наша оценка качества.

Удивительно, но вы уже тестируете!

Поскольку цель тестирования — понять функции продуктов и разобраться, как они должны работать, стоит отметить, что вы, вероятно, уже проводите тестирование в той или иной форме. Можно утверждать, что в любой деятельности, будь то отладка кода, загрузка API или отправка клиенту вопросов о том, как должен работать ваш API, вы чему-то учитесь. Следовательно, вы тестируете.

Именно поэтому иногда считается, что тестирование — это легкая задача. Но существует различие между ситуативным, неформальным и целенаправленным тестированиями. Мы знаем, насколько сложными могут быть продукты, и только при стратегическом подходе к тестированию это различие становится очевидным.

Команды, которые хорошо информированы о собственной работе, имеют лучшее представление о качестве своего продукта. Они также лучше осведомлены о том, какие шаги предпринять для улучшения качества (что позволяет сосредоточить внимание на конкретных рисках), как внести изменения в продукт, чтобы он лучше соответствовал ожиданиям пользователей, и какие проблемы

исправить, а какие оставить без внимания. В этом и заключается цель хорошего тестирования: оно помогает командам занять позицию, в которой они могут принимать обоснованные решения и уверенно идти по пути разработки высококачественного продукта.

1.2.4. Стратегический подход к тестированию API

Я считаю, что представленная выше модель — отличный способ описать цель и значение тестирования. Однако она может показаться несколько абстрактной. Как применить ее к тестированию API? Как будет выглядеть стратегия тестирования API при таком подходе? Одна из целей этой книги — помочь вам лучше понять эту модель. Давайте рассмотрим пример стратегии тестирования API, разработанной для проекта HMRC, участником которого я был.

Проект представлял собой сервис, позволявший пользователям искать и читать нормативные документы, а также создавать отчеты. Архитектура системы упрощенно представлена на рис. 1.3.

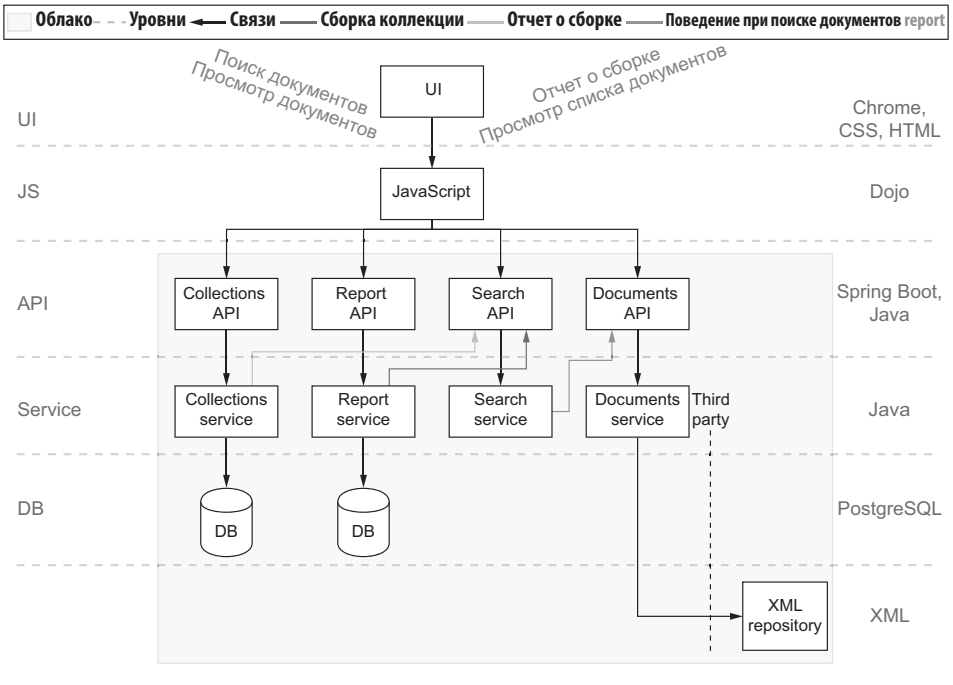

Рис. 1.3. Неформальная визуализация системной архитектуры платформы веб-API

Эта упрощенная модель дает представление о приложениях, для которых нам приходится разрабатывать стратегии тестирования API. Мы обсудим эту модель в главе 2, а здесь отметим, что приложение состоит из серии веб-API, которые предоставляют услуги пользовательскому интерфейсу и друг другу. Например, `Search` API может быть запрошен пользовательским интерфейсом и другим API, таким как `Report` API. Итак, у нас есть пример приложения, но как применить к этому контексту модель тестирования, о которой мы узнали? Опять же, лучше всего это объяснить на визуальной модели, показанной на рис. 1.4.

Рис. 1.4. Экземпляр модели описывает определенные действия по тестированию в рамках общей стратегии тестирования API

Как видим, части кругов представления и реализации заполнены операциями тестирования, которые помогут узнать, как работают наши веб-API. Для представления применимы следующие виды тестирования:

- *Тестирование дизайна API* позволяет нам подвергать сомнению идеи и улучшать общее понимание проблем, которые мы пытаемся решить.
- *Тестирование контрактов* помогает убедиться, что веб-API взаимодействуют друг с другом и корректно обновляются при внесении изменений.

Для реализации применимы такие виды тестирования:

- *Исследовательское тестирование* позволяет понять, как ведут себя веб-API, и обнаружить потенциальные проблемы.

- *Тестирование производительности* помогает лучше понять поведение веб-API под нагрузкой.

И наконец, у нас есть возможность *автоматизированных проверок* API, которые охватывают области пересечения наших представлений и реализации. Эти проверки могут подтвердить, что наши знания о работе API по-прежнему верны, и привлечь внимание к любому потенциальному ухудшению качества.

В этой книге мы узнаем больше об этих и других видах тестирования. Приведенная модель демонстрирует, что различные техники тестирования сосредоточены на разных областях нашей работы и предоставляют разную информацию. Успешная стратегия тестирования API должна быть целостной по своему подходу, сочетая множество различных техник, которые работают вместе, дополняя друг друга. Чтобы создать такую стратегию, требуется следующее:

1. *Проанализируйте контекст и его риски*: кто наши пользователи? Чего они хотят? Как работает наш продукт? Как мы работаем? Как они понимают качество?
2. *Оцените доступные типы тестирования*: знаем ли мы, как эффективно использовать автоматизацию? Знаем ли мы, что можем тестировать идеи и дизайны API до начала написания кода? Как мы можем получить пользу от тестирования в продакшене?
3. *Используйте знание контекста, чтобы выбрать правильные действия по тестированию*: какие риски наиболее важны для нас и как их снизить с помощью тестирования?

В этой книге мы рассмотрим все три пункта, чтобы дать необходимые навыки и знания для определения и реализации стратегии тестирования, которая будет помогать вам, вашей команде и вашей организации. По мере чтения книги мы будем возвращаться к общей модели тестирования, чтобы понять, какие техники тестирования работают лучше всего и где, а затем интегрировать их в эффективную стратегию тестирования. Прежде чем углубиться в многочисленные аспекты тестирования API, давайте познакомимся с несколькими подходами, которые помогут быстро изучить платформы веб-API.

ИТОГИ

- Веб-API содержат несколько уровней. Каждый из них выполняет собственные сложные задачи, которые становятся еще более сложными при их комбинировании.

- Сложность возрастает еще больше, когда несколько веб-API работают вместе в рамках сервисов для конечного пользователя на платформе.
- Понимание этой сложности является ключом к созданию высококачественного продукта.
- Чтобы достичь такого понимания, требуется целенаправленная стратегия тестирования.
- Тестирование можно рассматривать как сосредоточение внимания на двух областях: представлений и реализации.
- Мы проверяем представления, чтобы узнать больше о том, что хотим построить. Мы проверяем реализацию, чтобы узнать больше о том, что уже создали.
- Чем больше мы знаем о представлении и реализации, тем больше они пересекаются и тем лучше мы информированы о качестве нашего продукта.
- Модель тестирования показывает, как тестирование работает в областях представлений и реализации.
- Успешная стратегия тестирования состоит из множества действий, которые работают вместе для поддержки команды.