Гайд по интеграционному, E2E-тестированию и API тестированию (на простом языке)

Этот гайд поможет разобраться в основных типах тестирования, которые используются для проверки взаимодействия разных частей системы. Мы рассмотрим интеграционное тестирование, сквозное тестирование (E2E-тестирование), тестирование API. Также мы узнаем, как работают интеграции между системами и как их тестировать.

Дудник Н.В.

Ссылки на ресурсы:

- https://t.me/info_course_protestinginfo
- https://t.me/protestinginfo
- Нельзяграм
- https://protestinginfo.ru/

by ProTestingInfo QA

Интеграционное тестирование (Integration Testing)

Интеграционное тестирование проверяет, как разные части системы взаимодействуют между собой. Основная цель — убедиться, что модули, системы или сервисы работают вместе так, как это описано в документации.

Основные типы интеграций:

- **Модули в одном приложении**: проверка взаимодействия компонентов внутри программы.
- **Взаимодействие между сервисами**: проверка, как две и более системы обмениваются данными.
- **Интеграция с внешними системами**: проверяем, как приложение взаимодействует с другими системами, например, с операционной системой или оборудованием.

Пример: отправить запрос (POST) с телом в одну систему и проверить, что другая система сохранила данные в своей базе данных.

Тестирование API (API Testing)

API-тестирование проверяет, как приложение взаимодействует с другими системами через внешние интерфейсы. Это важно для проверки правильной работы клиент-серверного взаимодействия.

Способы работы API:

1. **SOAP API**: Проверка запросов и ответов в формате XML (SOAP требует наличия WSDL (Web Services Description Language) для описания веб-служб).
2. **REST API**: Тестирование POST, PUT, PATCH запросов с данными в формате JSON или XML, а также тестирование GET, DELETE запросов (в REST API методы определены с использованием HTTP-методов).
3. **GraphQL API**: Тестирование сложных запросов с выбором конкретных данных (GraphQL передает в приложение необходимые данные за один запрос, даже если они находятся в нескольких источниках).
4. **gRPC API**: Проверка вызовов между клиентами и серверами с использованием протокола gRPC (в gRPC API методы определены в специальных файловых протоколах - .proto)
5. **WebSocket API**: Тестирование двусторонней связи в режиме реального времени между клиентом и сервером через протокол WebSocket.

Сквозное тестирование (E2E-тестирование)

E2E-тестирование проверяет выполнение полного клиентского сценария от начала до конца, чтобы убедиться, что вся цепочка операций работает корректно. Этот подход важен, когда нужно протестировать функциональность на уровне, который будет полезен конечному пользователю.

Пример на основе реального проекта (без раскрытия конфиденциальной информации):

В одном из проектов я проводила E2E-тестирование, которое было завершающим этапом проверки. Необходимо было убедиться, что весь процесс — от создания статьи до её публикации на основном источнике — работает корректно, то есть протестировать реальные сценарии взаимодействия пользователей с системой.

Мы проверяли все интегрированные части приложения, а также его взаимодействие с другими сервисами. Особое внимание уделялось тестированию функциональности этих систем/сервисов, и, конечно, важно было убедиться, что итоговый результат соответствует требованиям.

E2E-тестирование — это самый длительный и трудоёмкий этап, который требует значительных ресурсов и времени. Наша команда старалась автоматизировать этот процесс, но это оказалось непростой задачей. Например, для интеграции одной системы с другой занимало в среднем 70 минут (нужно было дождаться создания и назначения ID сущности (статьи)) .

Основные действия:

1. Загрузить пакет метаданных и содержимое научной статьи в нужный сервис через FTP.
2. Убедиться, что в необходимом сервисе (система 1) создан идентификатор статьи.
3. Изменить статус статьи в UI-системе (система 1) для последующих действий и проверить, что обязательные поля отображаются корректно.
4. Проверить два API-сервиса (система 2 и система 3)и убедиться, что соответствующие идентификаторы (ID) в них появились.
5. Убедиться, что в системе 4 после обновления статуса системы 1 создана папка с метаданными статьи. Папка должна иметь правильное наименование, а внутри должен находиться соответствующий контент.
6. Проверить корректность взаимодействия системы 4, которая связана с системой 5 через брокер сообщений. Убедиться, что сообщения успешно передаются между системами, и данные обрабатываются корректно.
7. Выгрузить архив-файл с новым содержимым из системы 5 и загрузить через UI-систему 6 данный файл и проверить отображение содержимого статьи после загрузки.
8. Если все ключевые проверки, включая взаимодействие через брокер сообщений, успешно выполнены после каждого обновления системы, основное тестирование можно считать завершённым.
9. После этого можно продолжить работу с научной статьей: назначить авторов, добавить её в публикацию и опубликовать на соответствующем сайте.

Интеграции между сервисами

Есть несколько способов, как системы могут взаимодействовать между собой:

- Прямые API-вызовы: Одна система отправляет запрос (например, через REST API), а другая отвечает. По принципу «запрос-ответ» или «в одну сторону».
- Брокеры сообщений: Вместо того чтобы передавать данные напрямую, системы используют посредников, таких как RabbitMQ или Kafka. Это похоже на почтовую службу, которая доставляет сообщения между системами.
- Обмен файлами: Иногда системы обмениваются данными через файлы, используя FTP или другие протоколы.
- На уровне баз данных тестирование может включать:
  - Одна общая БД для нескольких систем: тестирование совместного использования базы данных разными системами.
  - Связанные базы данных: проверка взаимодействия через ETL/ELT-процессы для передачи данных между различными БД.

Пример: Система А отправляет запрос в систему В через брокер сообщений, чтобы передать данные для дальнейшей обработки.

Слайд на основе **источника**

Типы интеграций

Интеграции могут быть как внутренними, так и внешними, и важны для обеспечения корректного взаимодействия различных частей системы.

Внутренние интеграции:

1. Взаимодействие микросервисов в платежной системе: В приложении микросервис обработки платежей получает запрос на оплату от пользователя, после чего отправляет запрос в микросервис проверки баланса. Микросервис баланса обращается к базе данных и возвращает информацию о доступных средствах. После этого микросервис обработки платежей завершает транзакцию, списывая средства с аккаунта.

2. Интеграция между логикой авторизации и микросервисом уведомлений: Когда пользователь проходит авторизацию в приложении, микросервис авторизации подтверждает его личность и передает результат другому микросервису, который отвечает за уведомления. Этот микросервис отправляет пользователю уведомление об успешном входе через e-mail или push-уведомление.

Внешние интеграции:

1. Интеграция с внешними платежными сервисами: После того как пользователь добавляет товары в корзину и переходит к оплате, система e-commerce взаимодействует с внешним платежным шлюзом (например, PayPal или Stripe). После отправки данных о транзакции внешний сервис обрабатывает оплату, а система интернет-магазина получает ответ о статусе: успешный платеж или ошибка.

2. Взаимодействие с API стороннего геолокационного сервиса: Приложение для доставки продуктов определяет местоположение пользователя через API стороннего геолокационного сервиса, например, Google Maps. Получив данные, система определяет ближайшие к пользователю магазины и отображает только те товары, которые доступны в его районе.

Примеры интеграционного тестирования

Пример интеграционного тестирования:

1. Клиентское приложение (например, мобильное приложение) взаимодействует с системой А через API.
2. Система А отправляет запрос на бэк-сервер, который проверяет данные в базе.
3. После этого система А обращается к системе В через API или через брокер сообщений (RabbitMQ или Kafka).
4. Система В обрабатывает запрос, используя свои сервисы и базы данных, и возвращает результат.
5. Система А передает результат обратно клиенту через API.

На каждом этапе проверяются:

- Корректность работы API (API тестирование).
- Взаимодействие систем через брокеры сообщений.
- Работа с базами данных обеих систем.

Примеры интеграционного тестирования

Хочу поблагодарить за вопросы, которые пишите в директ, это тоже опыт для меня. И вот был следующий вопрос: "Добрый день! Подскажите, что значит высокоуровневые и низкоуровневые модули?" после того как я запостила информацию про расшифровку понятий. И тут я поняла, что общими фразами как-то понятно , но нет примеров... Согласны?

Давайте разберемся.

Интеграционное тестирование - тестирование связи между модулями и тестирование связи между частями ПО.

Целью данного уровня тестирования является выявление дефектов взаимодействия между этими программными модулями при их интеграции.

Driver, Stub - эмулируемые компоненты системы.

#Драйвер (Driver) - это заглушка, которая отправляет запросы к нашей тестируемой системе (или другим образом управляет).

#Заглушка (Stub) - это заглушка, которая получает запросы от системы и отправляет какой-то "вшитый" ответ.

И заглушки, и драйверы являются фиктивными модулями и создаются только для тестовых целей.

#Заглушки используются при тестировании сверху вниз, когда основной модуль готов к тестированию, но подмодули еще не готовы. Таким образом, на простом языке заглушки - это «вызываемые» программы, которые вызываются для проверки функциональности основного модуля. Например, в ситуации, когда у одного есть три разных модуля: Login, Home, User. Предположим, модуль входа в систему готов к тестированию, но два второстепенных модуля Home и User, которые вызываются модулем входа, еще не готовы для тестирования. В это время написан кусок фиктивного кода, который имитирует вызываемые методы Home и User. Эти фиктивные фрагменты кода являются заглушками. Или Например, ситуация, в которой веб-приложение готово, но база данных ещё не готова, и вместо базы данных делают временно заглушку. Вместо базы данных будет использована заглушка.

#Драйверы являются «вызывающими» программами. Драйверы используются при восходящем тестировании (Снизу вверх). Драйверы - это фиктивный код, который используется, когда подмодули готовы, но основной модуль еще не готов. Возьмем тот же пример, что и выше. Предположим, на этот раз модули User и Home готовы, но модуль Login не готов к тестированию. Теперь, когда Home и User возвращают значения из модуля Login, создается фиктивный фрагмент кода, имитирующий модуль Login. Этот фиктивный код затем называется Driver.

Пример: **https://ru.abcdef.wiki/wiki/Test_stub**

На моём опыте было реализовано приложение, в котором была разработана имитация логина в систему, так как нужна была внешняя интеграция с другой системой, поэтому сначала разрабатывались нижние модули: личный кабинет пользователя, список сущностей и их создание, обновление и удаление и т.д.

Примеры интеграционного тестирования

Подходы к интеграционному тестированию

Расшифровка понятий:

Интеграционное тестирование (Integration Testing)
Проверяется взаимодействие между компонентами системы после проведения компонентного тестирования.

Подходы к интеграционному тестированию:

- **Снизу вверх (Bottom Up Integration)**
Все низкоуровневые модули, процедуры или функции собираются воедино и затем тестируются. После чего собирается следующий уровень модулей для проведения интеграционного тестирования. Данный подход считается полезным, если все или практически все модули, разрабатываемого уровня, готовы.

- **Сверху вниз (Top Down Integration)**
Вначале тестируются все высокоуровневые модули, и постепенно один за другим добавляются низкоуровневые. Все модули более низкого уровня симулируются заглушками с аналогичной функциональностью, затем по мере готовности они заменяются реальными активными компонентами. Таким образом мы проводим тестирование сверху вниз.

- **Большой взрыв («Big Bang» Integration)**
Все или практически все разработанные модули собираются вместе в виде законченной системы или ее основной части, и затем проводится интеграционное тестирование. Такой подход очень хорош для сохранения времени. Однако если тест кейсы и их результаты записаны не верно, то сам процесс интеграции сильно осложнится, что станет преградой для команды тестирования при достижении основной цели интеграционного тестирования

PRO TESTING INFO

Схема (API 4 шт., Message broker 1 шт., Интеграционные потоки 4 шт.)

mellarius.ru

Литература

Habr

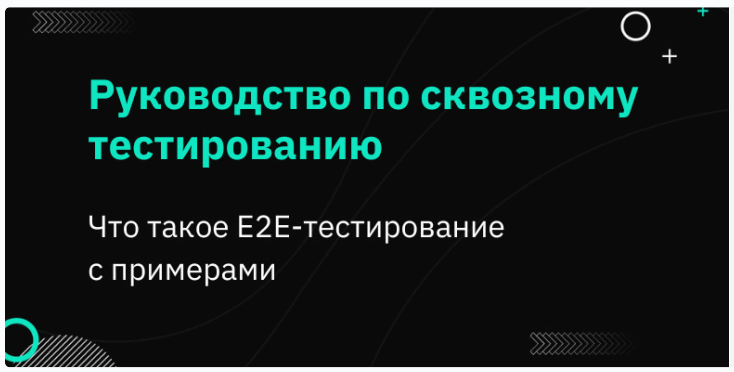
Интеграционное тестирование: виды, примеры и инструменты

Интеграционное тестирование нередко вспоминают на собеседованиях, когда спрашивают о видах и уровнях тестирования. И, как любую...

Публичная личная база знаний

API-, интеграционное и E2E-тестирование

Общий подход к.



Habr

Руководство по сквозному тестированию: что такое E2E-тест...

Сквозное тестирование Сквозное тестирование (End-to-end, E2E, Chain testing) — это вид тестирования, используемый для проверки...