API

API – это механизмы, которые позволяют двум программным компонентам взаимодействовать друг с другом, используя набор определений и протоколов. Например, система ПО метеослужбы содержит ежедневные данные о погоде. Приложение погоды на телефоне «общается» с этой системой через API и показывает ежедневные обновления погоды на телефоне.

API – Application Programming Interface, что значит программный интерфейс приложения. В контексте API слово «приложение» относится к любому ПО с определенной функцией. Интерфейс можно рассматривать как сервисный контракт между двумя приложениями. Этот контракт определяет, как они взаимодействуют друг с другом, используя запросы и ответы.

Как используется API и кем?

Такие интерфейсы дают возможность разным приложениям взаимодействовать между собой и обмениваться информацией. А если совсем просто, то API предоставляет доступ к функционалу другой программы. Например: для интеграции с YouTube разработчику вашей компании не нужно будет создавать сложную программу. Вместо этого можно использовать API YouTube — готовые части существующих ресурсов, у которых есть доступ к нужной вам информации и данным. Разработчик должен настроить работу с конкретным API, для этого есть документации, в которых описываются классы, методы, переменные, константы, и приводятся функциональные примеры.

Каждый раз, когда вы делаете что-то на сайте через интернет, то вы задействуете API сервисов, которые обрабатывают ваши запросы и формируют вам ответы.

Используется тестировщиками - тестируем какую-то функциональность через графический или программный интерфейс: автотесты на уровне API или интеграция между двумя разными системами.

Как работают API?

Архитектура API обычно объясняется с точки зрения клиента и сервера. Приложение, отправляющее запрос, называется клиентом, а приложение, отправляющее ответ, называется сервером. Итак, в примере с погодой база данных службы – это сервер, а мобильное приложение – это клиент.

Существует четыре различных способа работы API в зависимости от того, когда и почему они были созданы:

SOAP API
SOAP – Simple Object Access Protocol, т. е. простой протокол доступа к объектам. Клиент и сервер обмениваются сообщениями посредством XML. Это менее гибкий API, который был более популярен в прошлом.

RPC API
Такие API называются системой удаленного вызова процедур. Клиент выполняет функцию (или процедуру) на сервере, и сервер отправляет результат обратно клиенту.

Websocket API
Websocket API – это еще одна современная разработка web API, которая использует объекты JSON для передачи данных. WebSocket API поддерживает двустороннюю связь между клиентскими приложениями и сервером. Сервер может отправлять сообщения обратного вызова подключенным клиентам, что делает его более эффективным, чем REST API.

Какие типы API существуют?

API классифицируются как по архитектуре, так и по сфере применения. Мы уже рассмотрели основные типы архитектур API, поэтому мы предлагаем рассмотреть сферы применения.

Частные API
Это внутренние API организаций, которые используются только для соединения систем и данных внутри бизнеса.

Общедоступные API
Это API с общим доступом и могут быть использованы кем угодно. С этими типами API может быть (но не обязательно) сопряжена некоторая авторизация и стоимость.

Партнерские API
Эти API доступны только авторизованным сторонним разработчикам для содействия партнерским отношениям между предприятиями.

Составные API
Эти API объединяют два или более разных API для решения сложных системных требований или поведения.

Стандарты безопасности

Прежде чем мы перейдем к техническим деталям, следует отметить одну важную вещь. Веб-API предоставляет интерфейс для веб-приложения, поэтому вам нужно думать о безопасности на двух уровнях: доступ к API и доступ к приложению.
На уровне API вам необходимы правильная аутентификация, авторизация, права доступа и т. д., Чтобы гарантировать, что только разрешенные клиенты могут использовать интерфейс и выполнять только разрешенные операции. На уровне приложения вы должны убедиться, что конечные точки вашего приложения (то есть URL-адреса, используемые для доступа к интерфейсу) не уязвимы для атак, которые проходят через интерфейс или обходят его.

Существует несколько стандартов, которые помогут нам сформулировать **список требований к безопасности API**:

OWASP (Open Web Application Security Project) известна своими списками рисков в разных

программных технологиях. Нам интересен список «10 наиболее опасных уязвимостей при разработке API»:

- API1:2019 Broken Object Level Authorization (Недостатки контроля доступа к объектам). Другое название этого риска: Insecure Direct Object References (Небезопасные прямые ссылки на объекты)
- API2:2019 Broken User Authentication (Недостатки аутентификации пользователей)
- API3:2019 Excessive Data Exposure (Разглашение конфиденциальных данных)
- API4:2019 Lack of Resources & Rate Limiting (Отсутствие проверок и ограничений)
- API5:2019 Broken Function Level Authorization (Недостатки контроля доступа на функциональном уровне)
- API6:2019 Mass Assignment (Небезопасная десериализация)
- API7:2019 Security Misconfiguration (Некорректная настройка параметров безопасности)
- API8:2019 Injection (Внедрение)
- API9:2019 Improper Assets Management (Недостатки управления API)
- API10:2019 Insufficient Logging & Monitoring (Недостатки журналирования и мониторинга)

Безопасность и авторизация

- Убедитесь, что API разработан в соответствии с правильными принципами безопасности: отказ по умолчанию, безопасное падение сервиса, принцип наименьших привилегий, отклонение всех невалидных данных в запросе и т. д.
    - Позитивные тесты: убедитесь, что API отвечает на правильную авторизацию всеми согласованными методами аутентификации - ответный токен auth 2.0, файлы cookie, дайджест и т. д. - как определено в вашей спецификации.
    - Негативные тесты: убедитесь, что API отклоняет все несанкционированные вызовы.
- Ролевые доступы: убедитесь, что определенные конечные точки доступны пользователю в зависимости от роли. API должен отклонять вызовы конечных точек, которые не разрешены для роли пользователя.
- Проверка протокола: проверьте HTTP / HTTPS в соответствии со спецификацией
- Утечки данных: убедитесь, что представления внутренних данных, которые должны оставаться внутри компании, не просачиваются за пределы общедоступного API в данных ответа.
- Политики ограничения скорости, троттлинга и политики контроля доступа

Что такое интеграции API?

Интеграции API – это программные компоненты, которые автоматически обновляют данные между клиентами и серверами. Некоторыми примерами интеграции API являются автоматическая синхронизация данных в облаке из галереи изображений телефона или автоматическая синхронизация времени и даты ноутбуке при смене часового пояса. Организации также могут использовать их для эффективной автоматизации многих системных функций.

Интеграция API – это соединение между двумя или более приложениями через их API (интерфейсы прикладного программирования), которые позволяют системам обмениваться источниками данных. Интеграция API позволяет управлять процессами во многих секторах и уровнях организации, обеспечивая синхронизацию данных, повышая производительность и увеличивая прибыль.

Зачем нужна интеграция API?

Возможность подключения к API помогает приложениям обмениваться данными и взаимодействовать друг с другом без вмешательства человека. Вы обеспечиваете связь между двумя веб-инструментами или приложениями через их API. Это позволяет организациям автоматизировать системы, улучшить беспрепятственный обмен данными и интегрировать текущие приложения.

Как достигается интеграция API

В настоящее время существует несколько способов добиться интеграции API. Они в значительной степени зависят от индивидуальных потребностей вашей системы или бизнеса.

Индивидуальные интеграции

Интеграция, созданная на заказ, включает в себя рукописный скрипт от разработчика программного обеспечения с глубоким пониманием и знанием документации API. Этот метод стал популярным несколько лет назад, но стоимость разработки (и текущего обслуживания) сделала его менее привлекательным по сравнению с новыми методами интеграции. Использование этого подхода также может занять очень много времени.

Соединитель приложений

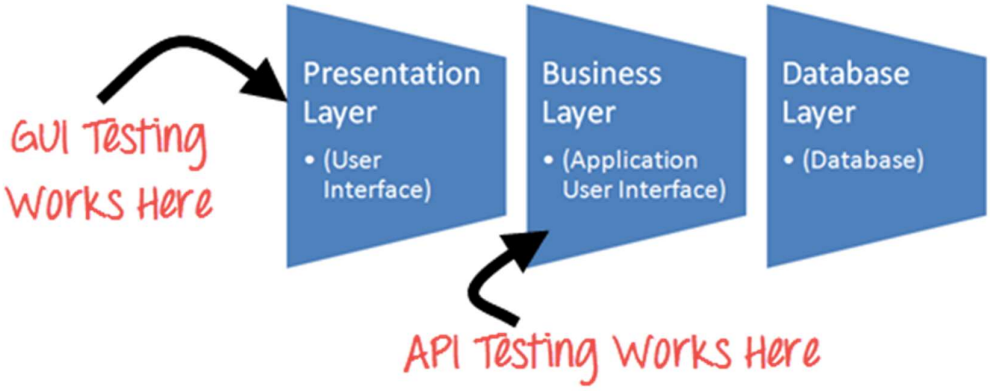
Соединитель приложений предназначен для облегчения передачи данных между двумя хорошо известными программными платформами. Соединители доступны по цене, ускоряют развертывание стандартных решений API и упрощают управление и обслуживание интеграций. Они также уменьшают потребность в управлении API.

Что такое API-тестирование?

API TESTING — это тип тестирования программного обеспечения, который проверяет интерфейсы прикладного программирования (API).

Целью API-тестирования является проверка функциональности, надежности, производительности и безопасности интерфейсов программирования. В тестировании API вместо использования стандартных пользовательских входов (клавиатура) и выходных данных вы используете программное обеспечение для отправки вызовов в API, получения выходных данных и записи ответа системы. Тесты API сильно отличаются от тестов GUI и

не будут концентрироваться на внешнем виде приложения. Он в основном концентрируется на уровне бизнес-логики архитектуры программного обеспечения.



Тестирование API требует приложения, которое может взаимодействовать через API.

Настройка среды тестирования API

- Тестирование API отличается от других типов тестирования программного обеспечения, поскольку графический интерфейс недоступен, и все же вам необходимо настроить начальную среду, которая вызывает API с необходимым набором параметров, а затем, наконец, проверяет результат теста.
- База данных и сервер должны быть настроены в соответствии с требованиями приложения.
- После завершения установки необходимо вызвать функцию API, чтобы проверить, работает ли этот API.

Тестовые случаи для тестирования API:

Тестовые случаи тестирования API основаны на

- Возвращаемое значение на основе входных условий: его относительно легко проверить, поскольку входные данные могут быть определены и результаты могут быть аутентифицированы.
- Ничего не возвращает: при отсутствии возвращаемого значения проверяется поведение API в системе.
- Запуск другого API / события / прерывания: если выход API инициирует какое-либо событие или прерывание, то эти события и прослушиватели прерываний следует отслеживать
- Обновление структуры данных: Обновление структуры данных будет иметь определенный эффект или влияние на систему, и это должно быть аутентифицировано
- Изменить определенные ресурсы: если вызов API изменяет некоторые ресурсы, его следует проверить путем доступа к соответствующим ресурсам.

Подход API тестирования:

Следующие пункты помогают пользователю выполнить тестирование API:

1. Понимание функциональности программы API и четкое определение объема программы
2. Применяйте методы тестирования, такие как классы эквивалентности, анализ граничных значений и отгадывание ошибок, и пишите контрольные примеры для API
3. Входные параметры для API должны быть запланированы и определены соответствующим образом
4. Выполните контрольные примеры и сравните ожидаемые и фактические результаты.

Как сделать API-тестирование

Тестирование API должно охватывать как минимум следующие методы тестирования помимо обычного процесса SDLC.

- Тестирование на обнаружение: тестовая группа должна вручную выполнить набор вызовов, задокументированных в API, таких как проверка того, что определенный ресурс, предоставляемый API, может быть перечислен, создан и удален соответствующим образом
- Тестирование юзабилити: это тестирование проверяет, является ли API функциональным и удобным для пользователя. И API хорошо интегрируется с другой платформой?
- Тестирование безопасности: это тестирование включает в себя, какой тип аутентификации требуется и зашифрованные конфиденциальные данные по HTTP или оба
- Автоматическое тестирование: тестирование API должно завершиться созданием набора скриптов или инструмента, который можно использовать для регулярного выполнения API
- Документация. Команда тестирования должна убедиться, что документация адекватна и предоставляет достаточно информации для взаимодействия с API. Документация должна быть частью окончательного результата

Лучшие практики тестирования API:

- Тестовые случаи должны быть сгруппированы по тестовым категориям
- В верхней части каждого теста вы должны включать объявления вызываемых API.
- Выбор параметров должен быть явно указан в самом тестовом примере
- Расставьте приоритеты вызовов функций API, чтобы тестировщикам было легко тестировать
- Каждый тестовый пример должен быть как можно более независимым и независимым от зависимостей.
- Избегайте «цепочки тестов» в вашей разработке
- Особое внимание следует уделить обработке одноразовых функций вызова, таких как — Delete, CloseWindow и т. Д.

- Последовательность вызовов должна быть выполнена и хорошо спланирована
- Чтобы обеспечить полное покрытие тестами, создайте тестовые случаи для всех возможных входных комбинаций API.

Типы ошибок, которые обнаруживает тестирование API

- Не в состоянии корректно обработать ошибки
- Неиспользованные флаги
- Отсутствует или дублируется функционал
- Вопросы надежности. Сложность подключения и получения ответа от API.
- Проблемы с безопасностью
- Проблемы многопоточности
- Проблемы с производительностью. Время отклика API очень велико.
- Неправильные ошибки / предупреждение звонящему
- Неправильная обработка допустимых значений аргумента
- Данные ответа не структурированы правильно (JSON или XML)

API состоит из набора классов / функций / процедур, которые представляют уровень бизнес-логики. Если API не протестирован должным образом, это может вызвать проблемы не только в приложении API, но и в вызывающем приложении. Это обязательный тест в разработке программного обеспечения.

Контрактное тестирование

Тестирование контрактов гарантирует, что две стороны способны взаимодействовать друг с другом, путем проведения изолированной проверки того, поддерживают ли обе стороны сообщения, которыми обмениваются.

Одна сторона (потребитель) фиксирует связь с другой стороной (поставщиком) и создает контракт. Этот контракт представляет собой спецификацию запросов от потребителя и ответов от поставщика.

Код приложения автоматически генерирует контракты (в большинстве случаев это происходит на этапе модульного тестирования). Автоматическое создание гарантирует, что каждый контракт отражает актуальную реальность.

Когда потребитель опубликует контракт, им сможет воспользоваться поставщик. В своем коде (возможно, и в модульных тестах) поставщик выполняет проверку контракта и публикует результаты.

На обоих этапах тестирования контракта мы работаем только с одной стороной, без какого-либо реального взаимодействия со второй. На практике мы следим, чтобы обе стороны могли общаться друг с другом по отдельным каналам. Таким образом, весь процесс остается асинхронным и независимым.

Если какой-либо из этих двух этапов завершится неудачей, то потребитель и поставщик должны договориться об устранении проблем с интеграцией. В некоторых случаях потребителю следует адаптировать интеграционный код, а в других — поставщику необходимо перенастроить API.

Контрактное тестирование — это не тестирование схемы. Тестирование схемы привязано к одной стороне без подключения к какой-либо другой. Контрактное тестирование проверяет взаимодействие с обеих сторон и гарантирует совместимость любых желаемых версий.

Контрактное тестирование — это не сквозное тестирование. В нем проводится комплексное тестирование группы совместно работающих сервисов, где тестируется вся система, начиная с пользовательского интерфейса и заканчивая хранилищем данных. Контрактное тестирование отдельно выполняет тесты для каждого сервиса. Они изолированы и не требуют запуска более одного единовременно.

Контрактные тесты позволяют использовать реальные версии клиентов и серверов и проверять их в изолированном процессе, чтобы выяснить, совместимы ли эти конкретные версии между собой.

Инструменты тестирования

Postman — это HTTP-клиент для тестирования API. HTTP-клиенты тестируют отправку запросов с клиента на сервер и получение ответа от сервера. API (Application Programming Interface) — это интерфейс для обмена данными с сервера между двумя приложениями или компонентами ПО. Тестировщикам Postman помогает в проектировании дизайна API и создании mock-серверов (имитаторов работы приложения).

Apache JMeter - это проект Apache, который может использоваться в качестве инструмента нагрузочного тестирования для анализа и измерения производительности различных сервисов с акцентом на веб-приложения.

JMeter может использоваться в качестве инструмента модульного тестирования для подключений к базе данных. Можно также настроить JMeter как монитор, хотя обычно это используется как базовое решение для мониторинга, а не для расширенного мониторинга. Его также можно использовать для некоторого функционального тестирования. Кроме того, Jmeter поддерживает интеграцию с Selenium, что позволяет

запускать сценарии автоматизации наряду с производительностью или нагрузочными тестами.

JMeter поддерживает параметризацию переменных, утверждения (проверку ответа), файлы cookie для каждого потока, переменные конфигурации и различные отчеты.

Архитектура JMeter основана на плагинах. Большинство его функций "из коробки" реализованы с помощью плагинов

Katalon Studio-это программное обеспечение для автоматизации тестирования, разработанное компанией Katalon, Inc. Программное обеспечение построено поверх платформ автоматизации с открытым исходным кодом Selenium, Appium со специализированным интерфейсом IDE для тестирования веб-приложений, API, мобильных и настольных приложений.

SoapUI — приложение с открытым исходным кодом для тестирования веб-сервисов сервис-ориентированных архитектур (SOA) и передачи состояний представлений (REST). Его функциональные возможности включают проверку веб-службы, запуск, разработку, моделирование и макетирование, функциональное тестирование, тестирование нагрузки и соответствия. Компания-разработчик программного обеспечения Eviware также создала коммерческую версию SoapUI Pro (ныне носит название ReadyAPI), которая в основном фокусируется на функциях, предназначенных для повышения производительности.