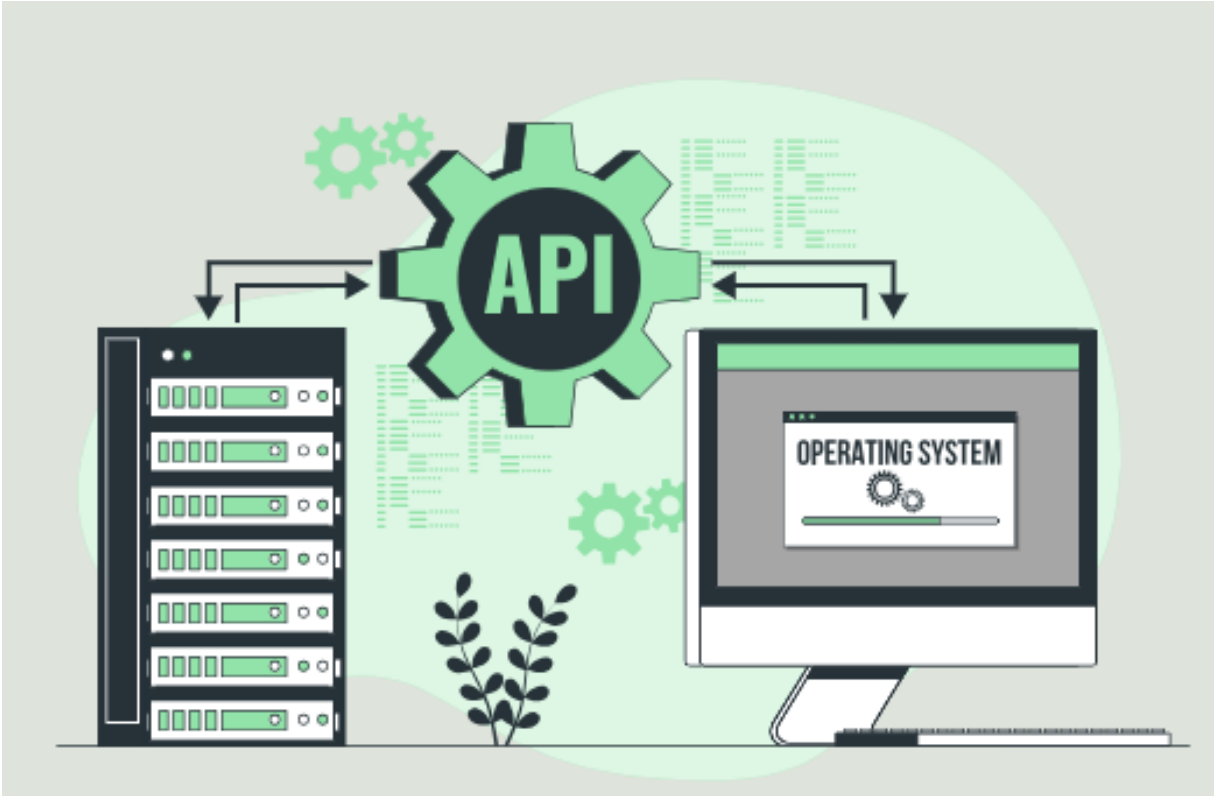
Тестирование API



Тестирование API — это тип тестирования ПО, который анализирует программный интерфейс приложения (API) для проверки того, что он соответствует ожидаемой функциональности, безопасности, производительности и надежности.

Тестирование API фокусируется на анализе бизнес-логики приложения, а также на данных и на безопасности.

В Agile-средах модульные тесты и тесты API предпочтительнее, чем тесты графического пользовательского интерфейса (GUI), поскольку их легко поддерживать, и они более эффективны.

НО, такое тестирование может быть сложным для небольших QA-команд, нанимаемых «под проект».

Все больше компаний переходят на использование микросервисов, а большинство микросервисов используют API, поэтому, рынок тестирования API предсказуемо будет расти.

Типы тестирования API

1. Тестирование методов

   Каждый метод API проверяют по отдельности — чтобы убедиться, что они работают правильно и возвращают ожидаемые результаты. Тестирование включает проверку входных данных, выполнение операций и проверку вывода.

2. Тестирование взаимодействий

   Проверка взаимодействия API с другими API, компонентами программы и сервисами. Показывает, что он успешно отправляет и получает данные, а также обрабатывает различные сценарии использования.

3. Тестирование авторизации и аутентификации

   Проверка доступа к API: как работают механизмы авторизации, кто и к каким функциям и данным имеет доступ.

4. Тестирование обработки ошибок

   Проверка поведения API в случае непредвиденных ситуаций и ошибок — например, передачи некорректных данных. Позволяет убедиться, что API правильно обрабатывает исключения и передаёт в программу верные коды ошибок.

5. Тестирование производительности

   Проверка работы API при повышенных нагрузках — его пропускной способности и производительности. Позволяет убедиться, что API не «отвалится» в случае повышенного спроса на программу.

6. Тестирование безопасности

   Проверка уязвимостей безопасности API, которая помогает предотвратить утечку данных или несанкционированный доступ. Проверяются меры безопасности API и проводятся тесты на проникновение для установления возможных уязвимостей.

Принципы тестирования API

- ✓ Использовать правильные и разнообразные входные данные

  При тестировании важно использовать разные типы данных, граничные значения, некорректные данные. Это помогает убедиться, что API правильно обрабатывает все возможные входные сценарии.

- ✓ Автоматизировать тестирование

  В тестировании API многое можно автоматизировать: например, проверку отдельных функций или обработку ошибок.

- ✓ Проводить непрерывные тесты

  Если в компании есть процессы CI/CD, важно включить в них тестирование API. Это позволит регулярно проверять его работоспособность и получать обратную связь о проблемах сразу после их возникновения.

- ✓ Проверять безопасность

  Поскольку API часто поставляются со стороны, важно тщательно тестировать их на уязвимости и проверять механизмы аутентификации, чтобы и API, и основная система были защищены от потенциальных угроз и атак.

Инструменты тестирования API

- ✓ **Postman**. Позволяет создавать, отправлять и тестировать HTTP-запросы и получать ответы от API. Предоставляет возможность создания автоматизированных тестов, генерации документации и совместной работы в команде.

- ✓ **SoapUI**. Позволяет тестировать и отлаживать SOAP и REST API: создавать и отправлять запросы, автоматизировать тестирование, генерировать тестовые отчеты, мониторить производительность.

- ✓ **JMeter**. Позволяет отправлять HTTP-запросы и проводить нагрузочное тестирование, чтобы проверить, как API справляется с высокими нагрузками, насколько оно производительно и масштабируемо.

- ✓ **REST-assured**. Java-библиотека, которая предоставляет простой и удобный способ для тестирования REST API с использованием DSL-синтаксиса. Она позволяет выполнять запросы, проверять ответы и создавать автоматизированные тесты.

- ✓ **Swagger**. Позволяет автоматически генерировать код для тестирования API и создавать автотесты.

Проектирование тестирования API

Вопросы, которые следует рассмотреть на этом этапе:

- ✓ Каковы функциональные возможности API?
- ✓ Какие конечные точки доступны для тестирования?
- ✓ Какие коды ответов ожидаются для успешных запросов
- ✓ Какие коды ответов ожидаются в случае неудачных запросов?
- ✓ Какое сообщение об ошибке ожидается в теле неудачного запроса?
- ✓ Какие инструменты тестирования API следует использовать?

Этап тестирования API

Тестовые случаи пишутся для API и должны определять условия или переменные, с помощью которых можно определить, отвечает ли конкретная система соответствующим образом и работает ли она правильно.

В процессе выполнения тестов анализируются следующие ответы:

- ✓ Время ответа.
- ✓ Качество данных.
- ✓ Подтверждение авторизации.
- ✓ Коды состояния протокола передачи гипертекста (HTTP).
- ✓ Коды ошибок.

Распространенные баги

- Баги в бизнес-логике, например финансовых платежей в банковском приложении
- Баги хранения данных и их передачи, например неспособность сохранять файлы с кириллическими символами в именах
- Нестабильное поведение, например API недоступен время от времени, или увеличивается время ответа до неприемлемого
- Неправильная обработка исключений, например API не возвращает ожидаемый код ошибки, если что-то нужное не найдено
- Неспособность отрабатывать негативные сценарии

В общих чертах тестирование API происходит по такой схеме

1. Определить требования к API и понимание, как именно оно работает и должно работать.
2. Создать тестовые случаи, которые описывают разные сценарии использования API. Это может быть отправка правильного или неправильного, проверка граничных значений.
3. Настроить окружение для тестирования API: установить необходимые инструменты или создать тестовые данные.
4. Отправить нужные запросы.
5. Проверить ответы от API.
6. Обработать ошибки — отправить намеренно некорректные данные.
7. Сгенерировать отчёты о том, как API прошло тесты.
8. Провести повторные тесты при изменениях в коде API или его окружения.

Пример: тестирование RESTful API для **системы управления задачами**

Шаги тестирования могут быть такими:

Предположим, что **этот API предоставляет следующие методы:**

- **GET /tasks** — получить список всех задач;
- **GET /tasks/{id}** — получить информацию о конкретной задаче;
- **POST /tasks** — создать новую задачу;
- **PUT /tasks/{id}** — обновить информацию о задаче;
- **DELETE /tasks/{id}** — удалить задачу.

1. **Определить требования**

Изучить документацию API, чтобы понять, какие поля должны быть в задаче, как она должна быть создана и обновлена, ожидаемые коды состояния и структуру ответов.

2. **Создать тестовые случаи,** например:

• Тест случая GET /tasks — отправить GET-запрос на /tasks и убедиться, что полученный ответ содержит список задач.

• Тест случая POST /tasks — отправить POST-запрос на /tasks с тестовыми данными для создания новой задачи и убедиться, что задача успешно создана и возвращается правильный код состояния (например, 201 Created).

• Тест случая PUT /tasks/{id} — создать тестовую задачу, получить её идентификатор, затем отправить PUT-запрос на /tasks/{id} с обновлёнными данными и убедиться, что задача обновлена и возвращается правильный код состояния (например, 200 OK).

Шаги тестирования могут быть такими:

3. Настроить окружение
Установить Postman для отправки запросов и проверки ответов API.

4. Отправить запросы
С использованием Postman отправить запросы из тестовых случаев к API.

5. Проверить ответы
Проверить ответы от API, используя ожидаемые результаты, указанные в тестовых случаях. Например, убедиться, что возвращается правильный код состояния, структура ответа соответствует ожидаемой и значения полей задачи корректны.

6. Обработать ошибки
Проверить, как API обрабатывает ошибки, отправляя некорректные запросы. Убедиться, что возвращаются соответствующие коды и описания ошибок.

7. Сгенерировать отчёты
Создавать отчёт о результатах после каждого выполнения тестов. Включить в него информацию о прошедших и не прошедших тестах, кодах состояния, ответах, ошибках, и другие полезные данные.

8. Регулярно проводить повторное тестирование
Особенно после внесения изменений в код API или его окружения, для обнаружения проблем и подтверждения работоспособности API.

Преимущества тестирования API

- **Для автоматизации тестирования API требуется меньше кода**, чем для автоматизированных тестов GUI, что обеспечивает более быстрое тестирование и более низкую общую стоимость.
- Тестирование API позволяет разработчикам получать доступ к приложению без UI, помогая им **выявлять ошибки на ранних этапах жизненного цикла разработки**. Это экономит деньги.
- **Тесты API не зависят от технологии и языка.** Обмен данными осуществляется с использованием JSON или XML , и содержит HTTP-запросы и ответы.
- Тесты API используют экстремальные условия и входные данные при анализе приложений. Это **помогает устранить уязвимости и защищает приложение от вредоносного кода и поломок.**
- Тесты API могут быть интегрированы с тестами GUI. Например, интеграция может позволить создавать новых пользователей в приложении до выполнения теста GUI.
- **Тестирование API можно эффективно автоматизировать**, что сокращает ручные усилия во время регрессионного тестирования и существенно экономит время.
- **Автоматизированное тестирование API не зависит от конкретного типа языка программирования**. Оно позволяет инженерам по обеспечению качества использовать любой язык программирования, поддерживающий технологии XML и JSON, независимо от языков приложений.

Проблемы тестирования API

- ✓ **Выбор параметров** требует проверки параметров, отправляемых через запросы API, — процесс, который может быть сложным. Однако необходимо гарантировать, что все данные параметров соответствуют критериям проверки, таким как использование соответствующих строковых или числовых данных, назначенный диапазон значений и соответствие ограничениям длины.
- ✓ **Комбинирование параметров** может оказаться сложной задачей, поскольку каждую комбинацию необходимо протестировать на предмет наличия проблем, связанных с конкретными конфигурациями.
- ✓ **Последовательность вызовов.** Чтобы гарантировать правильную работу системы, каждый вызов должен появляться в определенном порядке. Это может быть сложным, особенно при работе с многопоточными приложениями.
- ✓ **Тестировщики должны уметь писать код**.

Postman — это сервис для создания, тестирования, документирования, публикации и обслуживания API.

Он позволяет создавать коллекции запросов к любому API, применять к ним разные окружения, настраивать мок-серверы, писать автотесты на JavaScript, анализировать и визуализировать результаты запросов.

Программа поддерживает разные виды архитектуры API: HTTP, REST, SOAP, GraphQL и WebSockets. Postman вовсю используют в Twitter, WhatsApp и Imgur, но благодаря удобному графическому интерфейсу разобраться в платформе может даже новичок.

Скачать Postman можно бесплатно на официальном сайте. Есть версии под Linux, Windows и macOS.

https://reqres.in/api/users

Save | Share

POST https://reqres.in/api/users Send

Params • Authorization Headers (8) Body • Scripts • Settings Cookies

Pre-request

Post-response •

```
1 pm.test("Test Add_new_user", function () {
2     pm.response.to.have.status(201);
3 });
```

Snippets

Get an environment variable

Get a global variable

Get a variable

Get a collection variable

Body Cookies Headers (15) Test Results (1/1) 201 Created 399 ms 1.06 KB

Testing your API endpoints one by one? Run your entire collection and test end-to-end workflows effortlessly. Learn more ↗ Run Collection

All Passed Skipped Failed

PASS Test Add_new_user

POST {{REQRES_URL}}api/users

Params Authorization Headers (9) Body Pre-request Script Tests Settings

```
1  pm.test("Verify status code is 201", function () {
2      pm.response.to.have.status(201);
3  });
4
5  pm.test("Verify that name equals Hlebushek", function () {
6      var jsonData = pm.response.json();
7      pm.expect(jsonData.name).to.eql("Hlebushek");
8  });
```

Body Cookies Headers (13) Test Results (2/2)

All Passed Skipped Failed

PASS Verify status code is 201

PASS Verify that name equals Hlebushek