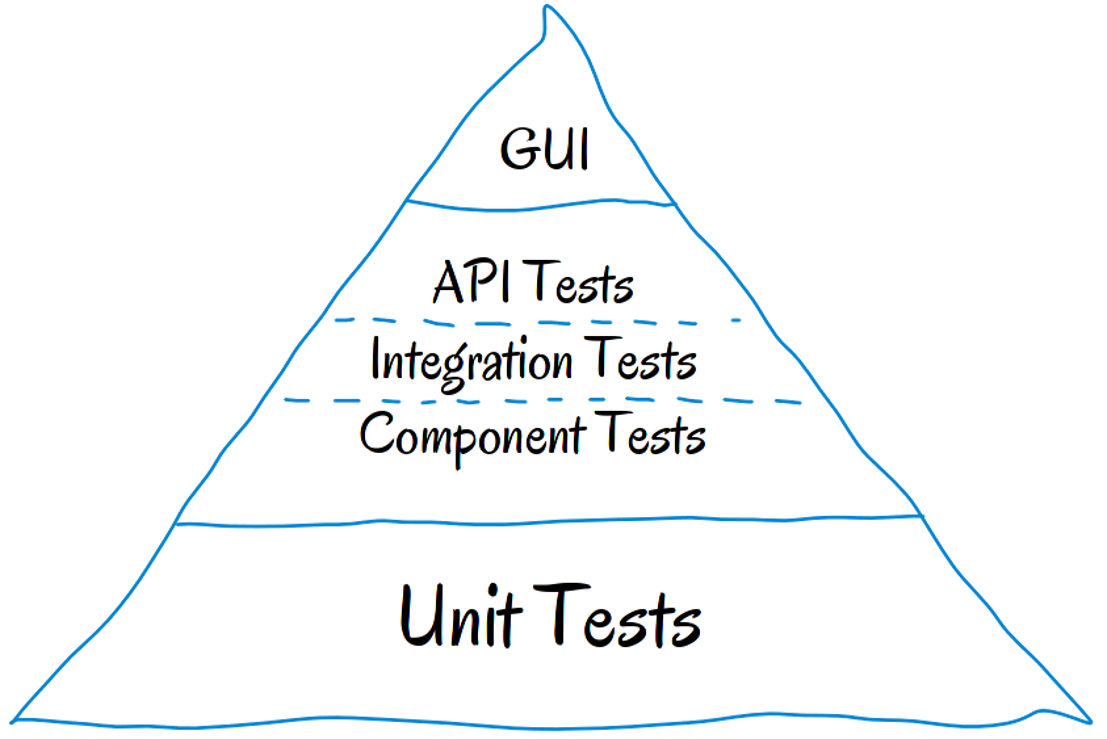
**Стратегия тестирования REST API: что именно вам нужно тестировать?**

Слой API любого приложения - один из важнейших программных компонентов системы. Это канал, который соединяет клиента с сервером (или один микросервис с другим), управляет бизнес-процессами и представляет сервисы, которые приносят пользу пользователям.

Общедоступный API, ориентированный на клиента, который делают открытым для конечных пользователей, сам по себе становится продуктом. Если он сломается, это подвергнет риску не только одно приложение, но и целую цепочку бизнес-процессов, построенных вокруг него.

Знаменитая [пирамида тестов](https://www.360logica.com/blog/sneak-peek-test-framework-test-pyramid-testing-pyramid/) Майка Кона помещает тесты API на сервисный уровень (интеграционный), что предполагает, что около 20% или более всех наших тестов должны быть сосредоточены на уровне API (точный процент зависит от наших потребностей).

Пирамида тестов

Когда у нас уже есть прочный фундамент из модульных тестов, охватывающих отдельные функции, тесты API обеспечивают более высокую надежность. Они проверяют интерфейс, более близкий к пользователю, но не имеют недостатков тестов пользовательского интерфейса.

Тесты API проходят быстро, обеспечивают высокую рентабельность инвестиций и упрощают проверку бизнес-логики, безопасности, соответствия и других аспектов приложения. В случаях, когда API является общедоступным, предоставляя конечным пользователям программный доступ к нашему приложению или службам, тесты API фактически становятся сквозными тестами и должны охватывать всю пользовательскую историю(*user story* - прим.переводчика).

Становится понятно, что важность тестирования API очевидна. Некоторые методологии и ресурсы помогают нам узнать **КАК** тестировать API - вы можете использовать ручное тестирование, автоматическое тестирование, тестовые среды, инструменты, библиотеки и фреймворки. Однако, независимо от того, чем вы будете пользоваться - Postman, supertest, pytest, JMeter, mocha, Jasmine, RestAssured или любыми другими инструментами - прежде чем открывать любой инструмент тестирования, вам необходимо определить, **что тестировать**...

**Стратегия тестирования API**

Стратегия тестирования - это высокоуровневое описание требований к тестированию, из которого впоследствии может быть составлен подробный план тестирования с указанием отдельных тестовых сценариев и тестовых случаев. Наша первая задача - это **функциональное тестирование**, чтобы убедиться, что API работает правильно.

Основными задачами функционального тестирования API являются:

* убедиться, что реализация API работает правильно, как и ожидалось - без ошибок!
* гарантировать, что реализация API работает в соответствии со спецификацией требований (которая позже становится нашей документацией по API).
* предотвратить регрессий между написанным кодом(merge) и релизом.

**API как соглашение - сначала проверьте спецификацию!**

API - это, по сути, соглашение между клиентом и сервером или между двумя приложениями. Перед тем, как начать любое тестирование функциональности, важно убедиться в правильности соглашения. Это можно сделать сначала проверив спецификацию (или само соглашение службы, например, интерфейс Swagger или ссылку OpenAPI) и убедившись, что:

* **эндпоинты правильно именованы**;
* **ресурсы и их типы правильно отражают объектную модель**;
* **нет отсутствующей или дублирующей функциональности**;
* **отношения между ресурсами правильно отражаются в API**.

Приведенные выше рекомендации применимы к любому API, но для простоты в этом посте мы предполагаем наиболее широко используемую архитектуру веб-API - **REST через HTTP**. Если ваш API спроектирован именно как RESTful API, важно убедиться, что **контракт REST действителен**, включая всю семантику, соглашения и принципы HTTP REST. (описание [тут](https://techbeacon.com/app-dev-testing/guide-restful-api-design-35-must-reads), [тут](https://restfulapi.net/rest-api-design-tutorial-with-example/), и [здесь](https://hackernoon.com/restful-api-designing-guidelines-the-best-practices-60e1d954e7c9)).

Если у вас общедоступный API, ориентированный на клиента, такое тестирование может быть вашим последним шансом убедиться, что все требования соглашения выполнены. После публикации и использования API любые внесенные вами изменения могут внести ошибки в код клиента.(Конечно, когда-нибудь вы сможете опубликовать новую версию API (например, /api/v2 /), но даже в этом случае обратная совместимость скорее всего будет обязательной).

**Итак, какие аспекты API мы должны протестировать?**

После того как мы проверили соглашение API, мы можем поразмышлять о том, что тестировать. Независимо от того, думаете ли вы об автоматизации тестирования или ручном тестировании, наши функциональные тест-кейсы имеют одинаковый набор **тестовых действий**. Они являются частью более широких **категорий тестовых сценариев** и их можно разделить на три **потока тестирования.**

**Этапы тестирования API**

Каждый тест состоит из тестовых шагов. Это отдельные атомарные действия, которые тест должен выполнять в каждом потоке тестирования API. Для каждого запроса API тест должен будет выполнить следующие действия:

1. **Проверьте корректность кода состояния HTTP**. Например, создание ресурса должно возвращать 201 CREATED, а запрещенные запросы должны возвращать 403 FORBIDDEN и т. Д.
2. **Проверьте полезную нагрузку ответа**. Проверьте правильность тела JSON, имен, типов и значений полей ответа, в том числе в ответах на ошибочные запросы.
3. **Проверьте заголовки ответа.** Заголовки HTTP-сервера влияют как на безопасность, так и на производительность.
4. **Проверьте правильность состояния приложения**. Это необязательно и применяется в основном к ручному тестированию или когда пользовательский интерфейс или другой интерфейс можно легко проверить.
5. **Проверьте базовую работоспособность**. Если операция была завершена успешно, но заняла неоправданно много времени, тест не пройден.

**Категории тестовых сценариев**

Наши тест-кейсы делятся на следующие общие группы *тестовых сценариев*:

* Основные позитивные тесты (прохождение успешного сценария по умолчанию)
* Расширенное позитивное тестирование с дополнительными параметрами
* Негативное тестирование с валидными входными данными
* Негативное тестирование с недопустимыми входными данными
* Деструктивное тестирование
* Тесты безопасности, авторизации и доступности (которые выходят за рамки этой статьи)

**Тестирование успешного сценария по умолчанию** проверяет базовую функциональность и критерии приемки API. Позже мы расширим положительные тесты, чтобы включить дополнительные параметры и дополнительные функции.

Следующая группа тестов - это **негативное тестирование**, при котором мы ожидаем, что приложение будет корректно обрабатывать проблемные сценарии как с валидным вводом пользователя (например, попытка добавить существующее имя пользователя), так и с недопустимым вводом пользователя (попытка добавить имя пользователя, которое имеет значение null).

**Деструктивное тестирование** - это более глубокая форма негативного тестирования, когда мы намеренно пытаемся сломать API, чтобы проверить его надежность (например, отправляя заведомо большое тело запроса в попытке переполнить систему).

**Тестовые потоки**

Давайте разделим и обозначим три вида потоков тестирования, которые составляют наш план тестирования:

1. **Изолированное тестирование запросов** - выполнение одного запроса API и соответствующая проверка ответа. Такие базовые тесты - это минимальные строительные блоки, с которых мы должны начинать. И нет смысла продолжать тестирование, если эти тесты упадут.
2. **Многоступенчатый рабочий поток с несколькими запросами** - тестирование серии запросов, которые являются обычными действиями пользователя, поскольку одни запросы могут зависеть от других. Например, мы выполняем запрос POST, который создает ресурс и возвращает автоматически сгенерированный идентификатор в своем ответе. Затем мы используем этот идентификатор, чтобы проверить, присутствует ли этот ресурс в списке элементов, полученных запросом GET. Затем мы используем PATCH для обновления новых данных и снова вызываем запрос GET для проверки этого обновления. И в завершении, мы УДАЛЯЕМ этот ресурс и снова используем GET, чтобы убедиться, что записи больше нет.
3. **Комбинированные тесты API и тесты веб-интерфейса** - это в основном относится к ручному тестированию, при котором мы хотим обеспечить целостность данных и согласованность между пользовательским интерфейсом и API.

Мы выполняем запросы через API и проверяем действия через пользовательский интерфейс веб-приложения и наоборот. Цель этих потоков проверки целостности состоит в том, чтобы гарантировать, что, хотя на ресурсы влияют различные механизмы, система по-прежнему поддерживает ожидаемую целостность и согласованный поток данных.

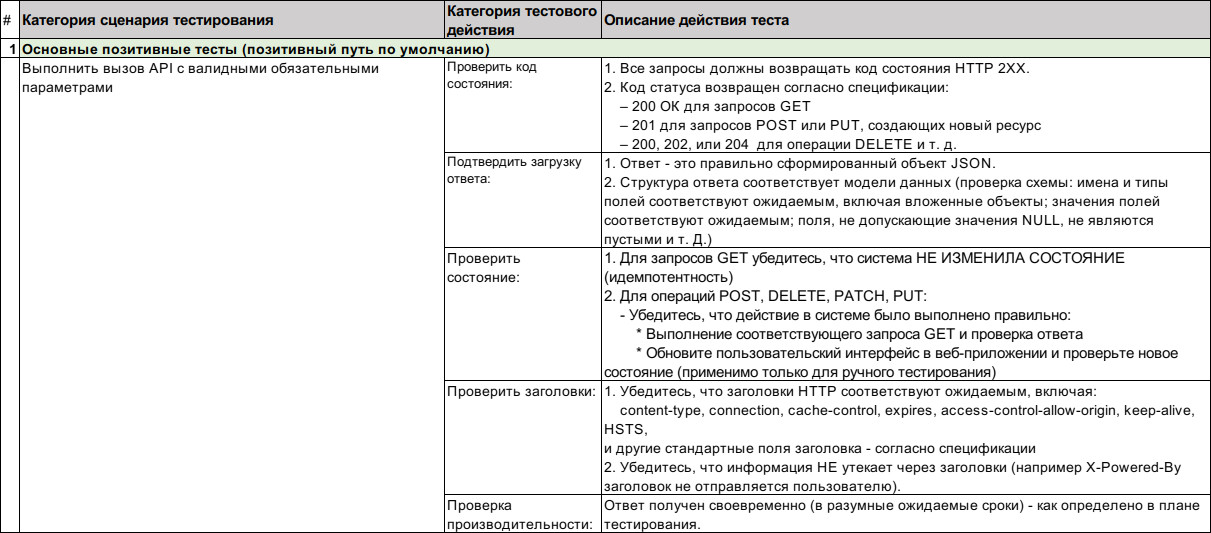
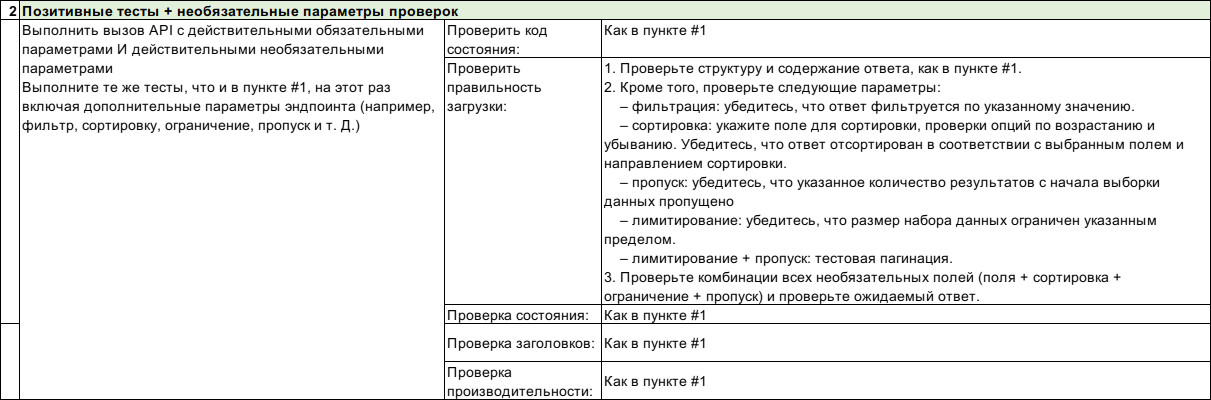
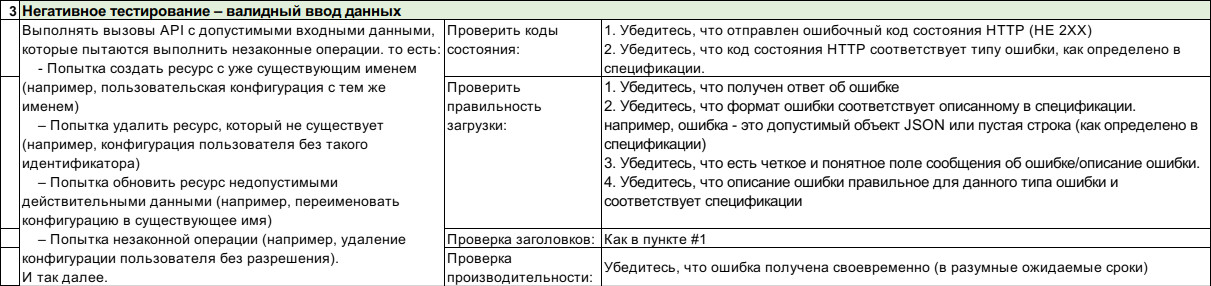
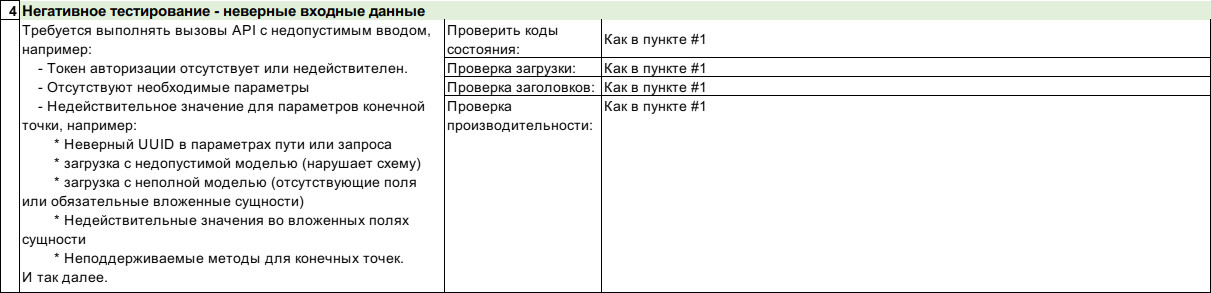
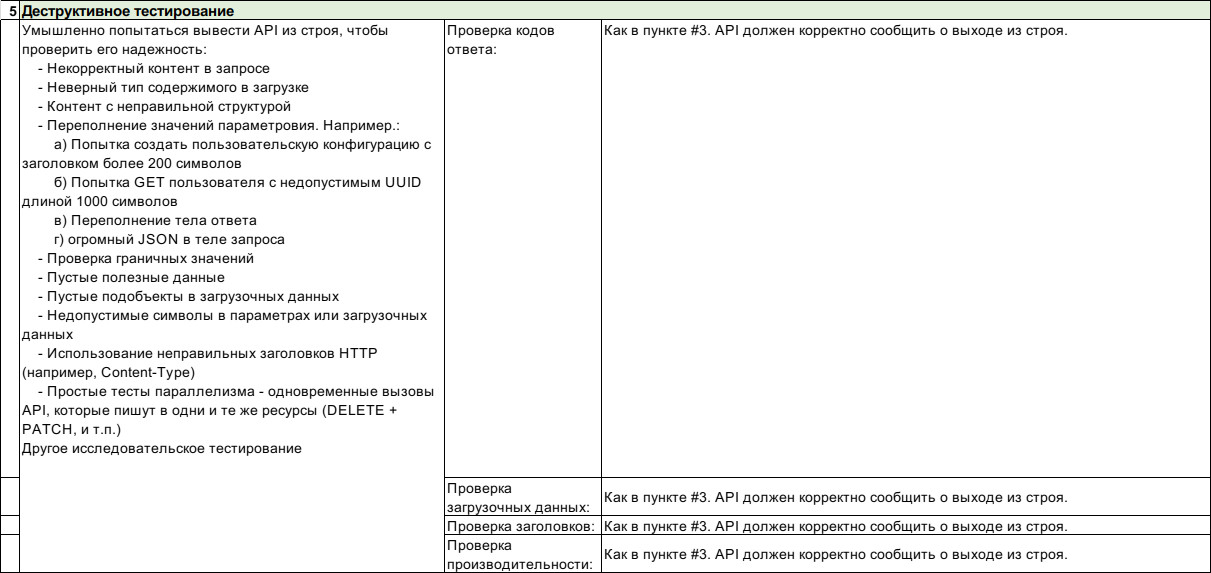
**Пример API и тестовая матрица**

Теперь мы можем отобразить все в виде матрицы и использовать ее для написания подробного плана тестирования (для автоматизации тестирования или ручных тестов).

Предположим, что подмножеством нашего API является конечная точка /users, которая включает следующие вызовы API:

|  |  |
| --- | --- |
| **Вызов API** | **Действие** |
| GET /users | Просмотреть всех пользователей |
| GET /users?name={username} | Получить пользователя по имени пользователя |
| GET /users/{id} | Получить пользователя по ID |
| GET /users/{id}/configurations | Получите все конфигурации для пользователя |
| POST /users/{id}/configurations | Создать новую конфигурацию для пользователя |
| DELETE /users/{id}/configurations/{id} | Удалить конфигурацию для пользователя |
| PATCH /users/{id}/configuration/{id} | Обновить конфигурацию для пользователя |

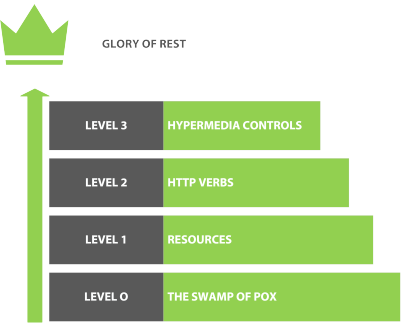
Где {id} - это UUID, а все конечные точки GET позволяют фильтровать, сортировать, исключать и ограничивать дополнительные параметры запроса для фильтрации, сортировки и разбивки на страницы тела ответа.

Основные позитивные тесты (позитивный путь по умолчанию)Позитивные тесты + необязательные параметры проверокНегативное тестирование – валидный ввод данныхНегативное тестирование - неверные входные данныеДеструктивное тестирование

Тест-кейсы, полученные из приведенной выше таблицы, должны охватывать различные потоки тестирования в соответствии с нашими потребностями, ресурсами и приоритетами ([перевод таблицы в формате xls](https://drive.google.com/file/d/1ePHKuEw_QErn9g5TzgA4UGaQJ4OG3jTp/view?usp=sharing)).

**Выходим за рамки функционального тестирования**

Следуя приведенной выше тестовой матрице, вы должны сгенерировать достаточно тест-кейсов, чтобы было что тестировать некоторое время и обеспечить хорошее функциональное покрытие API. Прохождение всех функциональных тестов подразумевает хороший уровень зрелости API (про зрелость [тут](https://blog.restcase.com/4-maturity-levels-of-rest-api-design/). *прим. переводчика*), но этого недостаточно для обеспечения высокого качества и надежности API.

Уровни зрелости API

В следующем разделе этой статьи мы рассмотрим следующие нефункциональные подходы к тестированию, которые необходимы для проверки качества API.

**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

**Производительность**

* Проверка времени ответа API, задержки, TTFB / TTLB в различных сценариях (изолированно и под нагрузкой)

**Нагрузочные тесты (позитивные), стресс-тесты (негативные)**

* Найдите точки ограничения производительности и убедитесь, что система работает должным образом под нагрузкой и корректно выходит из строя под нагрузкой.

**Юзабилити-тесты пользователей API**

* Для общедоступных API: ручной тест на уровне продукта, который проверяет весь путь разработчика от документации, входа в систему, аутентификации, примеров кода и т. д., Чтобы гарантировать удобство использования API для пользователей, не знакомых с вашей системой.