

# API

## API (Application Programming Interface или интерфейс программирования приложений)

Это совокупность инструментов и функций в виде интерфейса для создания новых приложений, благодаря которому одна программа будет взаимодействовать с другой. Это позволяет разработчикам расширять функциональность своего продукта и связывать его с другими.

Когда пользователь посещает любую страницу в интернете, он взаимодействует с API удаленного сервера. Это составляющая сервера, которая получает запросы и отправляет ответы. Кроме того, благодаря API человек может совершать различные действия, не покидая сайт. Именно для этого большинство современных сайтов используют по крайней мере несколько сторонних API, которые предлагают сторонние разработчики. Также компании разрабатывают собственные API и продают их как готовый продукт. К примеру, Weather Underground, которая принадлежит IBM, продает доступ к своему API для получения метеорологических данных. Эту информацию используют погодные приложения и сервисы.

## Как работает API

Интерфейс представляет собой промежуточный слой между двумя приложениями. Он позволяет двум программам обмениваться информацией и выполнять функции, не раскрывая своего внутреннего API. Скрытие части функций называется инкапсуляцией.

## Есть три метода взаимодействия с API:

1. Процесс, который может выполнять программа при помощи этого интерфейса.
2. Данные, которые нужно передать интерфейсу для выполнения им функции.
3. Данные, которые программа получит на выходе после работы с API.

API бывают публичные и частные. Первые предназначены для совместного использования с внешним миром, например, API YouTube. Сторонние разработчики могут создавать приложения, чтобы воспользоваться возможностями этих интерфейсов. Вторые — это внутренние приложения, разработанные для определенной аудитории или пользовательской базы. Они часто используются на предприятиях и внутри компаний. Для работы с таким API нужно получить доступ.

## Для чего используют API

Разработчикам программный интерфейс позволяет:

- упростить и ускорить выпуск новых продуктов, так как можно использовать уже готовые API для стандартных функций;
- сделать разработку более безопасной, выведя ряд функций в отдельное приложение, где они будут скрыты;
- упростить настройку связей между разными сервисами и программами и не сотрудничать для разработки своего продукта с создателями различных приложений;
- сэкономить деньги, так как не нужно разрабатывать все программные решения с нуля.

До появления Windows и других графических операционных систем программистам для создания окон на экране компьютера приходилось писать тысячи строк кода. Когда же Microsoft предоставила разработчикам API Windows, на создание окон стало уходить всего несколько минут работы.

## Примеры API в нашей жизни

**Google Календарь.** Приложение-календарь на Android разработает на API, позволяющем подключить свой календарь напрямую к сторонним приложениям. Пользователи могут использовать несколько разных программ с встроенными и обновляемыми календарями, где будут все важные события,

встречи и т.д. Компании могут встраивать API календаря на свои сайты, чтобы, к примеру, записывать своих клиентов на прием. Встраивание в форму записи Google Календаря позволяет клиентам автоматически создавать событие и вносить детали о предстоящей встрече. Благодаря API сервер сайта напрямую обращается к серверу Google с запросом на создание события, получает ответ Google, обрабатывает его и передает соответствующую информацию в браузер, которая поступает клиенту в виде сообщения с подтверждением.

**Заказ авиабилетов.** Многие пользуются агрегаторами билетов, такими как Aviasales и SkyScanner. Такие сервисы собирают информацию о стоимости авиабилетов в разных авиакомпаниях и отображают ее в едином окне. Это позволяет реализовать API, встроенный в сайты авиакомпаний, который помогает в реальном времени обновлять информацию о направлениях и стоимости.

**Поиск авиабилета на Aviasales.** Многие пользуются агрегаторами билетов, такими как Aviasales и SkyScanner. Такие сервисы собирают информацию о стоимости авиабилетов в разных авиакомпаниях и отображают ее в едином окне. Это позволяет реализовать API, встроенный в сайты авиакомпаний, который помогает в реальном времени обновлять информацию о направлениях и стоимости.

**Навигация на сайтах и в приложениях.** Крупные компании, в том числе Apple, Google, «Яндекс» и другие, разработали API, позволяющие подключить собственный картографический сервис к другим площадкам. Так, в «Яндекс.Карты» встроены сервисы «Транспорт» и «Пробки». Многие приложения на Android, например, по доставке еды или для спорта, используют встроенный в ОС API, чтобы подключить карты Google к своему сервису. На iOS аналогичная ситуация с Apple Maps.

**Кнопки авторизации.** На многих сайтах есть кнопки, позволяющие зарегистрироваться через уже существующие аккаунты на популярных площадках и в соцсетях. Это возможно благодаря API, которые есть у Google, Facebook, Apple, Twitter, «ВКонтакте» и других компаний.

## Что такое API тестирование?

Тестирование API полностью отличается от тестирования графического интерфейса и в основном концентрируется на слое бизнес-логики архитектуры программного обеспечения. GUI в API тестировании практически не нужен.

Вместо стандартных видов ввода пользовательских данных (заполнение форм) здесь для передачи данных используется программное обеспечение.

## Понимание подходов тестирования API

Соглашение о подходе к тестированию API — важная часть стратегии тестирования. Желательно договориться об этом перед началом разработки API. Ниже перечислены несколько принципов:

- Прозрачное описание области тестирования и хорошее понимание функциональности API;
- Понимание основных методологий тестирования, таких как анализ граничных значений и классов эквивалентности — часть составления тест-кейсов для API;
- Планирование, определение и подготовка входящих параметров, пустых запросов и тестовых примеров для API;
- Определение и сравнение ожидаемых и фактических результатов. Проверка того, что отличий между ними нет.

## Типы API тестирования

- **Модульное тестирование (Unit testing)** — тесты, которые валидируют результаты отдельных методов.

*Модульные тесты обычно пишутся разработчиками. Считается, что юнит-тестирование — это хорошая практика, которая позволяет снизить технический долг и стоимость обслуживания системы в будущем.*

*Атомарность и изолированность методов API позволяет хорошо покрывать код тестами.*

- **Тесты валидации.** В отличие от юнит-тестов, которые проверяют маленькие кусочки функциональности, валидационные тесты находятся на более высоком уровне и отвечают на набор вопросов, после ответа на которые программное обеспечение может перейти к следующему этапу разработки.

Вопросы для валидации могут быть следующие:

- Вопросы специфичные для продукта: “Это именно тот функционал, который был нужен?”

- Вопросы ожидаемого поведения: “Этот метод ведет себя так, как ожидалось?”
- Вопросы, связанные с эффективностью: “Этот метод использует ресурсы максимально независимо и эффективно?”

Все эти вопросы служат для проверки API в разрезе согласованных критериев приемки. Еще они позволяют быть уверенным в соблюдении стандартов доставки ожидаемой конечной ценности и безупречном удовлетворение потребностей и требований пользователей.

*Валидация здесь является частью пользовательского тестирования.*

*Доступ к методу включается на тестовом контуре после функционального тестирования и бизнес-подразделение проверяет его соответствие бизнес-требованиям. Например, корректность работы калькулятора, который использует API-вызовы через веб-интерфейс*

- **Функциональные тесты.** Тесты, выполняющие конкретные методы API. Проверка количества активных пользователей через API, регрессионные тесты и выполнение тестовых случаев относятся к функциональным тестам.
- **E2E и UI тесты.** Тесты, которые запускают и проверяют сценарии конечного использования продукта, включая пользовательский интерфейс, API и БД. Проверяется результаты выполнения метода на каждом этапе транзакции.

*E2E проводится силами нескольких команд: Тестировщики фронта, тестировщики бэка, специалисты инфраструктурных технических отделов, ответственные за передаваемые данные. Предварительно лучше составить список тестов, который удовлетворит видению бизнеса. Можно взять и из плана тестирования, но там он часто избыточен. Этот список согласуется со всеми участниками e2e тестирования, чтобы к вам не было потом лишних вопросов.*

- **Нагрузочное тестирование.** Увеличение количества пользователей не должно влиять на производительность приложения. Нагрузочное тестирование выявит проблемы производительности и доступности сервисов при масштабировании, а также проверит производительность API в нормальных условиях.

*Нагрузочное тестирование — ресурсоемкий процесс. У нас на проекте делается единожды перед сдачей очередной версии продукта, по просьбе заказчика или владельца. .*

- ♦ **Тесты на дефекты в процессе работы ПО.** Тесты, помогающие наблюдать за выполнением ПО и фиксировать проблемы, связанные с неопределенностью параллелизма (race conditions), исключениями и утечкой ресурсов. Ниже содержится краткая информация об этих факторах.
  - **Тесты мониторинга API:** Проверяют реализацию на различные ошибки, сбои внутренних обработчиков и другие неотъемлемые проблемы в кодовой базе API. Эти тесты гарантируют отсутствие дыр, которые могут привести к проблемам с безопасностью приложения.
  - **Ошибки выполнения:** Проверка позитивных запросов к API на корректность ответа всегда присутствует в плане тестирования, однако проверка негативных сценариев не менее важная часть стратегии тестирования API.
  - **Утечка ресурсов.** *Негативные тесты для проверки сбоев в работе основных ресурсов API путем отправки некорректных запросов. Ресурсы здесь — это память, данные, уязвимости, операции тайм-аута и так далее.*
  - **Отслеживание сбоев.** *Обнаружение сбоев сетевого обмена. Сбои аутентификации из-за предоставления неверных учетных данных являются примером сценария обнаружения ошибок. Эти тесты гарантируют, что ошибки фиксируются, а затем устраняются.*

*При правильно составленных функциональных тест-кейсах часть этих проверок уже будет включена в их состав, а именно негативные тесты авторизации, утечки и корректность ответа.*

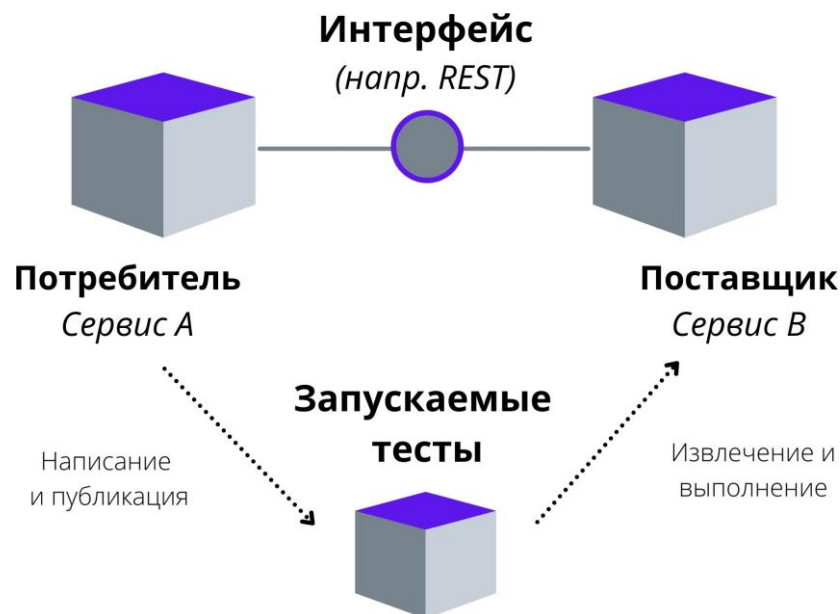
## Что же такое Consumer Driven Contract (CDC)?

Как известно, каждый интерфейс имеет поставщика (supplier) и потребителя (consumer). Само собой, сервисы поставщика и потребителя распределены между разными командами, мы оказываемся в ситуации, когда чётко прописанный интерфейс между ними (или контракт) просто необходим. Обычно многие подходят к этой проблеме следующим образом:

1. Пишут подробное описание спецификации интерфейса - контракт
2. Реализуют сервис поставщика согласно спецификации
3. Передают спецификацию интерфейса потребителю
4. Ждут реализации от другой стороны

5. Запускают ручные системные тесты, чтобы всё проверить
6. Держат кулачки, что обе стороны будут вечно соблюдать описанный интерфейс

Разберемся во взаимодействии на примере REST архитектуры: поставщик создает API с некоторым endpoint, а потребитель отправляет запрос к API, например, с целью получения данных или выполнения изменений в другом приложении.



Это контракт, который описывается с помощью DSL (domain-specific language).

Он включает API описание в форме сценариев взаимодействия между потребителем и поставщиком. С помощью CDC выполняется тестирование клиента и API с использованием заглушек, которые собираются на основе контракта. Основной задачей CDC является сближение восприятия между командами разработчиков API и разработчиков клиента. Таким образом, участники команды потребителей пишут CDC тесты (для всех данных проекта разработки), чтобы команда поставщика смогла запустить тесты и проверить API. В итоге команда поставщика с легкостью разработает свой API, используя тесты CDC. **Результатом прогона контрактных тестов является понимание, что поставщик уверен в исправной работе API у потребителя.**

Следует обратить внимание, что команда потребителя должна регулярно осуществлять поддержку CDC-тестов при каждом изменении, и вовремя передавать всю информацию команде поставщика. Если регулярно фиксируем неудачно выполненные CDC-тесты, то следует пойти (в буквальном смысле слова, к пострадавшей стороне теста и узнать, в рамках какой задачи были изменения (что привело к падению теста), а также уточнить, к чему в перспективе приведет данное изменение - прокачиваем софт скиллы:)

### **Тезисы для выполнения контрактного тестирования:**

1. Команда разработчиков (тестировщиков) со стороны потребителей пишет автоматизированные тесты с ожидаемыми параметрами со стороны потребителей.
2. Тесты передаются команде поставщика.
3. Команда поставщика запускает контрактные тесты и проверяет результат их выполнения. Если происходит падение тестов, то команды должны зафиксировать сбой и перепроверить документацию (согласованность разработки).

Использование CDC-тестов является важным шагом в разработке микросервисной архитектуры приложений, т.к. позволяет использовать автономные группы тестов. Т.к. в данный момент микросервисная архитектура становится широко применяемой в IT, то подход с CDC-тестированием становится популярным и, что немаловажно, достаточно эффективным из-за своей позиции в иерархии видов тестов.

### **Минусы CDC**








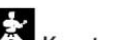
Было сказано много чего хорошего, но какие минусы есть у CDC? В целом сводятся в основном к следующим ограничениям:

1. CDC тесты не заменяют E2E тесты. По факту я склонен отнести CDC к заглушкам, которые являются моделями реальных компонентов, но не являются ими, т.е. это еще одна абстракция, которую нужно поддерживать и применять в нужных местах (сложно реализовать сложные сценарии).
2. CDC тесты не заменяют функциональные тесты API. Если убрать контракт и это не вызывает ошибки или неправильную работу клиента, то значит он не нужен. Пример: Нет необходимости проверять все коды ошибок через контракт, если клиент обрабатывает их (ошибки) одинаково. Таким образом контракт то, что важно для клиента сервиса, а не наоборот.
3. CDC тесты дороже в поддержке, чем функциональные тесты.
4. Для реализации CDC-тестов нужно использовать (изучать) отдельные инструменты тестирования - Spring Cloud Contract, PACT.



## Сводная таблица инструментов для тестирования API

На следующем снимке экрана с инструментами тестирования API приведен краткий обзор и подробности о каждом инструменте, простоте их использования, а также поддерживаемых платформах.

							
API	API, Web & Mobile	API	API, Web & Mobile	API	API	API, Web & Mobile	API
Paid + Free	Free	Open Source	Paid + Free	Paid + Free	Open Source	Paid + Free	Open Source
Windows Linux MacOS	Windows Linux MacOS	Windows Linux MacOS	Windows	Windows Linux MacOS	Windows Linux MacOS	Windows Linux MacOS	Windows Linux MacOS
Easy to setup & use	Easy to setup & use	Advanced programming skills needed to setup & use	Easy to setup. Need training to properly use the tool	Require end-points management knowledge to use	Easy to setup & use	Easy to setup. Need training to properly use the tool	Easy to setup & use

Все эти инструменты используются в реальных проектах. Чаще, в публикуемых вакансиях можно встретить Postman и SoapUI — оба нужны для тестирования различных API. Первый для REST API, второй для API, построенных на SOAP архитектуре. Их можно и взаимозаменять, так как Postman позволяет делать SOAP запросы и наоборот. Но возможны и некоторые коллизии. Кроме всем известных инструментов, перечисленных в таблице, используется еще десяток сервисов или внутренних разработок для анализа и подготовки тестовых данных. “Стандартный” стек, кроме тех, что в описан выше: Nodepad++ с плагинами Json\XML, VS Code(swaggerplugin), GIT, SQL Developer, Powershell\Unix bash, Kibana(logs) или аналог.