

Тестирование API. CI




Артем
Романов



Артём Романов

Инженер по обеспечению качества

 [romanov-artyom](#)

 [Романов Артём](#)

 [@Artem_Telegram](#)



План занятия

1. [API](#)
2. [HTTP, REST и JSON](#)
3. [REST-assured](#)
4. [Continuous Integration](#)
5. [Итоги](#)



API



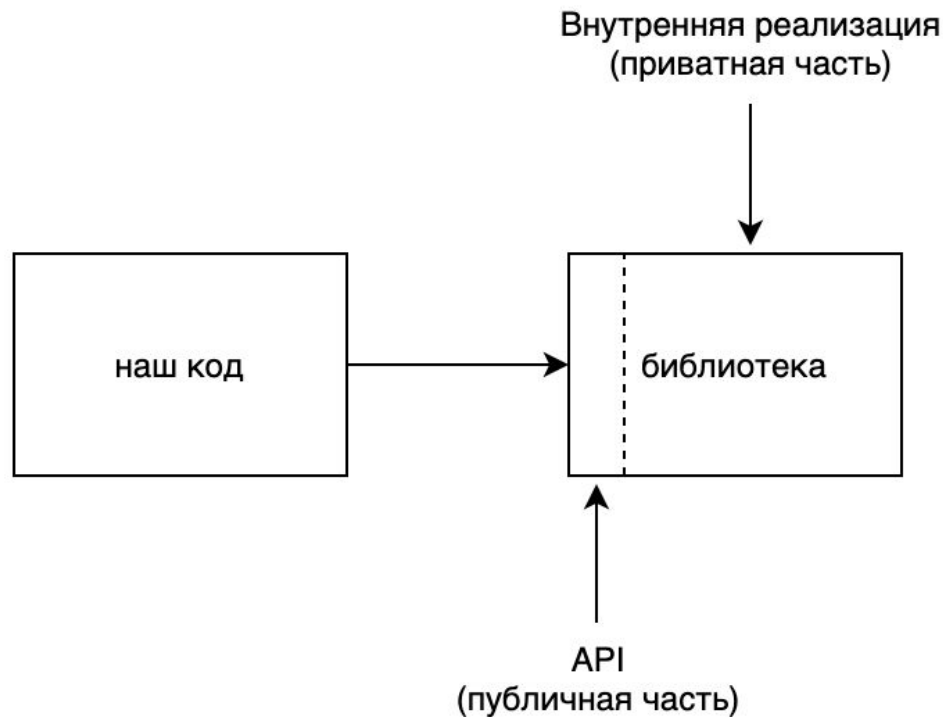
API

API (Application Programming Interface) – это спецификация программного взаимодействия двух систем.

Если говорить проще, то это просто определённые правила взаимодействия двух приложений (либо частей одного приложения).

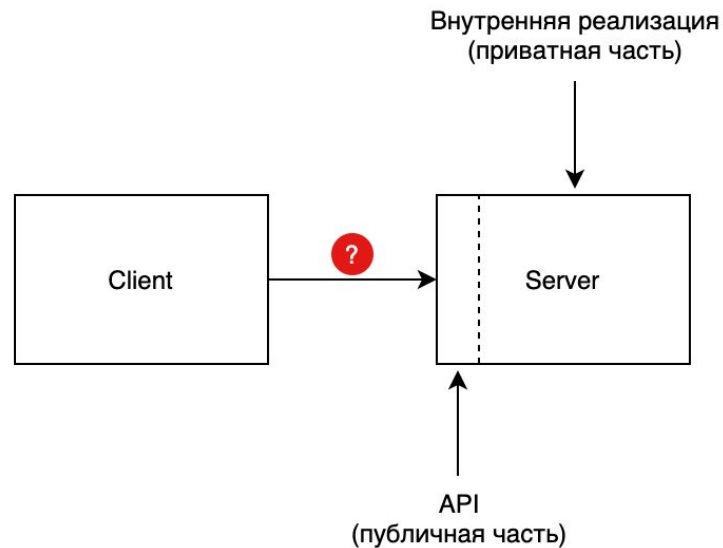
API

Когда мы с вами подключаем библиотеки с Maven Central, то API определялось тем, какие публичные классы и методы эта библиотека представляет (т.е. что мы можем использовать).



Client-Server

Начиная с сегодняшней лекции, мы выходим за рамки одного приложения (у нас больше не будет доступа к классам и методам SUT) и будем рассматривать самый популярный метод взаимодействия двух систем:



Примечание*: SUT (System Under Test) – тестируемая система.



Client & Server

Client (клиент) — приложение, задача которого **подготавливать запросы и обрабатывать ответы.**

Server (сервер) — приложение, задача которого **обрабатывать запросы и подготавливать ответы.**

Аналогия из жизни: в реальной жизни можете представлять себе сервер как магазин, а клиент — покупатель. Это очень яркая аналогия, которая поможет вам понять почти все происходящие процессы.

Например, один магазин обслуживает много клиентов. Если клиентов много, то может образоваться очередь на «обслуживание».



Client & Server

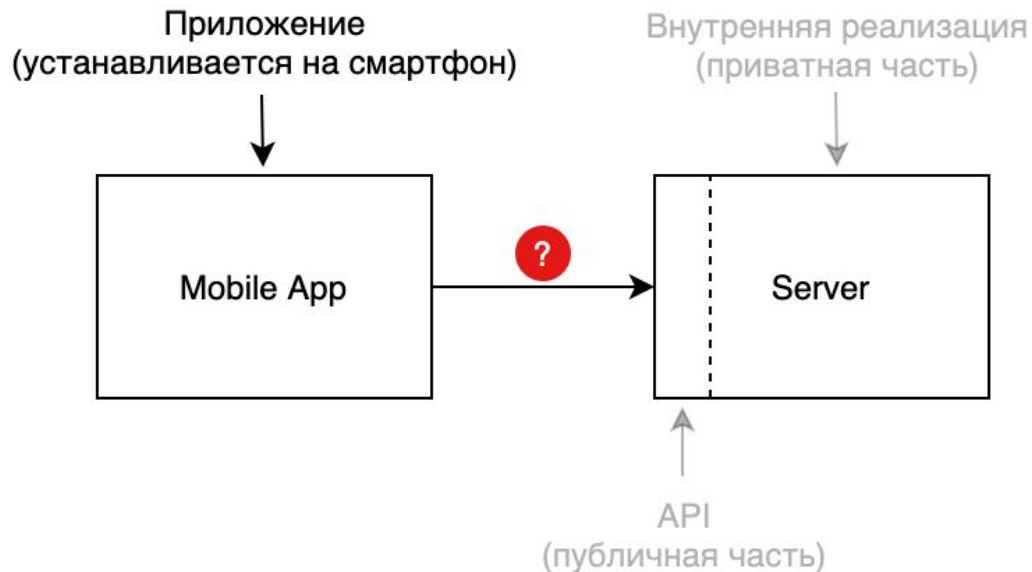
Вопрос к аудитории: у большинства из вас в смартфонах установлены приложения с мобильными банками. Давайте подумаем, где в этом приложении:

- клиент;
- сервер.

Mobile Banking

В данном случае клиентом будет выступать приложение на вашем смартфоне, а сервер — какой-то сервер банка, к которому вы можете через приложение обращаться с различными запросами.

Этот же сервер отвечает за взаимодействие с хранилищами данных, в которых хранится информация о ваших счетах, картах, кредитах:



Q & A

Q: Разве сервер — это не «большой» компьютер, а программа?

A: К сожалению, одними и теми же терминами могут называть совершенно разные вещи. Сервером называют как компьютер, так и приложение, обслуживающее клиентов. Что конкретно имеется в виду стоит уточнять из контекста.



Client vs Server

Чаще всего клиент — достаточно ненадёжная сторона: у него может пропасть интернет (прямо во время запроса), сесть батарея, кто-то может клиенту позвонить или произойти еще что-то (вспомните людей в очереди на кассу).

Сервер же, наоборот, должен быть очень надёжным и, желательно, работать 24x7 (потому что клиент может захотеть что-то заказать и в выходной, и ночью).

Вопрос к аудитории: как вы думаете, что более приоритетно — тестирование сервера или клиента?



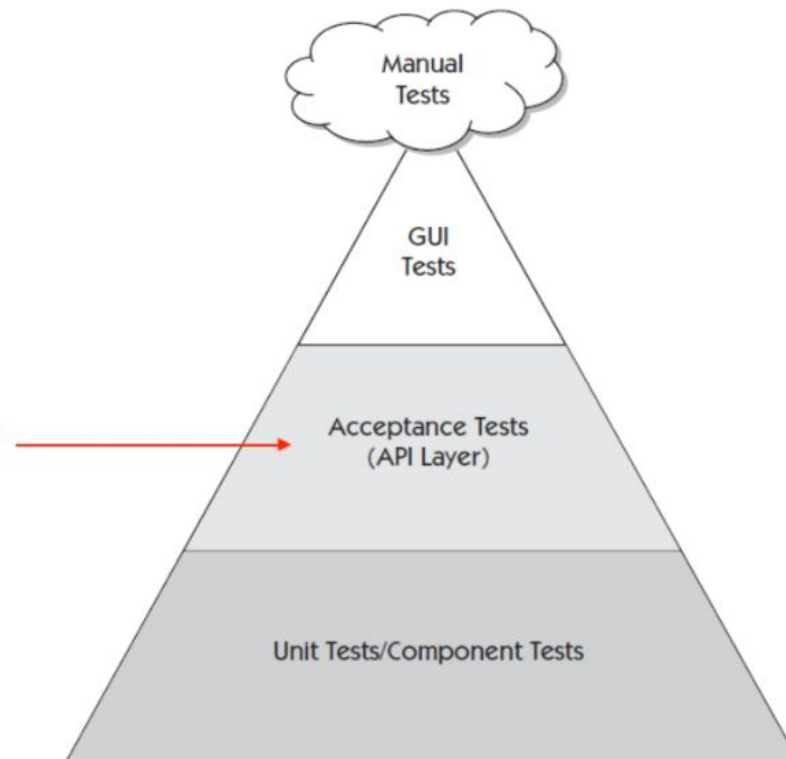
Client vs Server

Вопрос поставлен некорректно: из него не ясен контекст.

Например, можно отталкиваться от самой важной для Банка услуги. Если для её предоставления нужен и сервер, и клиент, то тесты должны быть и на сервер, и на клиент.

Client vs Server

В этой лекции мы с вами сосредоточимся на тестировании сервера, а именно API, которое предоставляется клиенту.





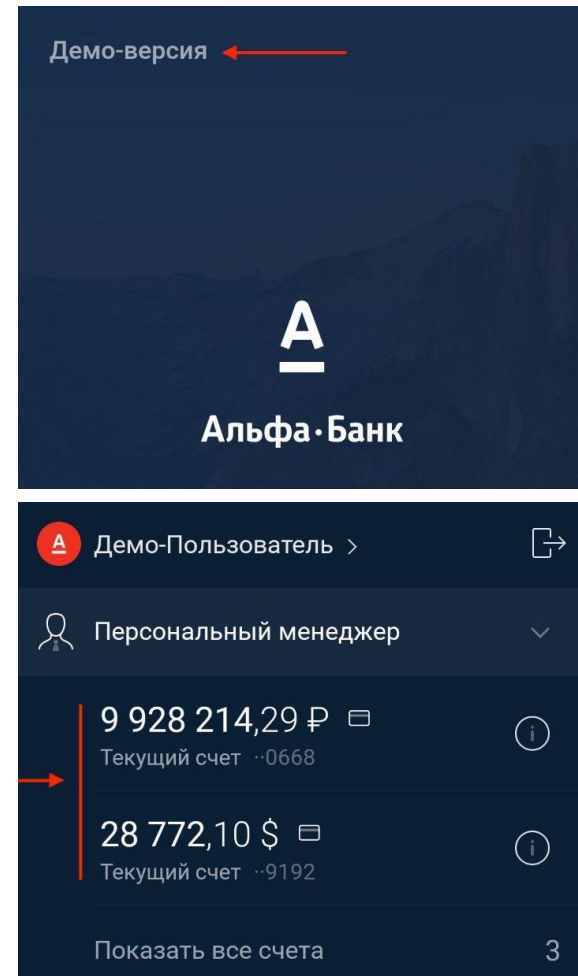
HTTP, REST и JSON

Задача

Мобильное приложение банка предоставляет возможность посмотреть демо без авторизации.

Нас будет интересовать только список счетов.

Вопрос к аудитории: каким образом мы можем передать серверу информацию о выбранном городе, а он нам ответ с офисами и банкоматами?





Передача запроса и получение ответа

Сервер и клиент работают на разных устройствах и должны обмениваться данными в рамках обработки запроса.

Кроме того, сервер и клиент могут быть написаны на разных языках программирования.

Всё это значит, что:

1. Должен быть выбран инструмент (транспорт) для доставки информации (набора байт) от клиента к серверу.
2. Должен быть выбраны правила интерпретации этих байт (клиент и сервер должны понимать друг друга).



Передача запроса и получение ответа

Аналогии из реальной жизни: допустим, вы хотите сделать заказ в магазине (что-то купить).

1. В качестве инструмента доставки информации вы можете выбрать заказ по телефону, по мессенджеру, на сайте или «ногами» прийти в сам магазин.
2. В качестве правила интерпретации чаще всего будет использоваться русский язык (попробуйте оформить заказ на китайском в российском магазине).



Передача запроса и получение ответа

Мы будем рассматривать достаточно распространённую схему, когда:

1. В качестве транспорта используется протокол HTTP (а именно HTTP 1.1).
2. В качестве «правил интерпретации» будет выбрано REST API с форматом данных JSON.

Примечание*: мы настоятельно рекомендуем вам хотя бы раз ознакомиться со спецификацией HTTP.

URI, URL

В сети интернет есть такое понятие как ресурс — некая информационная единица, которая идентифицируется посредством [URI \(Uniform Resource Identifier\)](#).

Примеры URI:

- `http://www.ietf.org/rfc/rfc2396.txt`
- `mailto:John.Doe@example.com`
- `tel:+1-816-555-1212`

Достаточно часто вы будете слышать термин URL (Uniform Resource Locators), который является подмножеством URI.

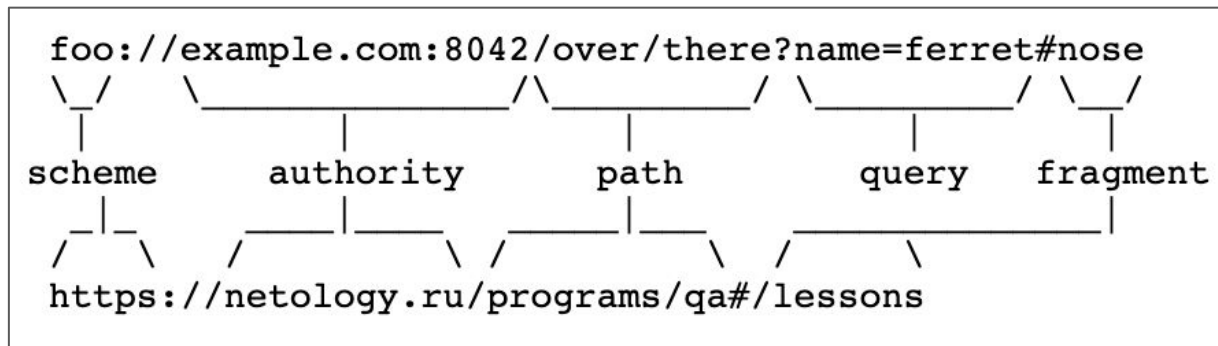
В повседневной жизни чаще всего URI и URL используются как синонимы, что не совсем корректно.

URI

URI состоит из трёх частей:

```
URI = scheme ":" hier-part [ "?" query ] [ "#" fragment ]
```

Иерархическая часть состоит из **authority** и **path**.



authority может включать себя логин и пароль пользователя, хост (или домен) и порт. Логин и пароль указываются достаточно редко. Порт также не указывается, если используется [общепринятый](#) (80 для http, 443 для https).

HTTP

С протоколом HTTP вы уже кратко знакомились в рамках курса «Введение в тестирование».

HTTP 1.1 — это текстовый протокол. Протокол означает «правила взаимодействия».

В рамках этих правил определяются форматы запросов и ответов:

4.1 Message Types

HTTP messages consist of requests from client to server and responses from server to client.

HTTP-message = Request | Response ; HTTP/1.1 messages

HTTP

5 Request

A request message from a client to a server includes, within the first line of that message, the method to be applied to the resource, the identifier of the resource, and the protocol version in use.

```
Request      = Request-Line           ; Section 5.1
               *(( general-header      ; Section 4.5
                 | request-header      ; Section 5.3
                 | entity-header ) CRLF) ; Section 7.1
               CRLF
               [ message-body ]       ; Section 4.3
```

6 Response

After receiving and interpreting a request message, a server responds with an HTTP response message.

```
Response     = Status-Line           ; Section 6.1
               *(( general-header      ; Section 4.5
                 | response-header     ; Section 6.2
                 | entity-header ) CRLF) ; Section 7.1
               CRLF
               [ message-body ]       ; Section 7.2
```



HTTP

Таким образом, сообщения состоят из трёх частей:

1. request/status line — что хотим от сервера (request), получаем ли то, что хотим (status);
2. headers — мета-данные;
3. message body — сами данные, но они могут отсутствовать.

Частей всего три, поэтому, передавать свои данные мы можем в этих трёх частях (в том числе комбинируя их).

HTTP

В рамках HTTP также определяются:

1. методы (GET, POST, PUT, DELETE) определяют логическое назначение действия + ограничения на запрос (например, у GET тело запроса должно быть пустое)
2. статус-коды ответов:
 - 1xx – информационные;
 - 2xx – успешно;
 - 3xx – перенаправление;
 - 4xx – ошибки клиента;
 - 5xx – ошибки сервера.



HTTP: Примеры

Вы уже умеете пользоваться Developer Tools из браузера.

Поэкспериментируйте с теми сервисами, которыми пользуетесь и посмотрите:

1. куда попадают данные, которые вы вводите: где-то они попадут в адресную строку (например, при поиске), а где-то в тело запроса (например, при авторизации);
2. каким методом отправляются запросы;
3. какие коды ответа приходят;
4. какие передаются заголовки.



JSON

JSON (JavaScript Object Notation) — формат данных, предназначенный для передачи информации. Наряду с XML является одним из самых распространённых форматов.

Именно JSON и XML достаточно часто используют для организации обмена данными (в нашем случае между клиентом и сервером).



Промежуточные итоги

Итого: у нас есть понимание, какой использовать транспорт и в каком формате передавать сообщения.

Осталось лишь спроектировать систему: как в Java мы проектировали набор методов у класса + нужно определить передаваемые данные на вход и получаемый результат.

API

Эта задача достаточно нетривиальная, и каждый решает её по-своему.

Например, ребята из Вк [сделали API](#), у которого в `query`-запроса передаётся необходимый метод:

https://api.vk.com/method/users.get?user_id=1&v=5.52

А GitHub сделал немного по-другому, например, для репозитория схема вот такая: `GET /users/:username/repos`.

REST API

REST — это архитектурный подход к проектированию систем, при котором вы представляете всю систему в виде набора (чаще всего иерархических) ресурсов.

Например, есть ресурс пользователи (`users`), у каждого пользователя есть репозитории (`repos`).

Пользователи и репозитории связаны иерархическими взаимоотношениями, поэтому для списка репозиториях `/users/:username/repos`.

Для конкретного репозитория будет: `/users/:username/repos/:repo`.

Для issue репозитория: `/users/:username/repos/:repo/issues`.

Для конкретного issue: `/users/:username/repos/:repo/issues/:issue`.

`:username` — это placeholder, на место которого подставляется конкретное значение.



REST API

Методы HTTP при этом имеют определённую смысловую нагрузку:

- GET — получить ресурс;
- POST — создать новый ресурс;
- POST/PUT/PATCH — обновить ресурс;
- DELETE — обновить ресурс.



REST API

Схема красивая, но в идеальном виде почти нереализуема (см. GitHub API).

Несмотря на это, в современном мире достаточно часто под REST имеют в виду систему разделения ресурсов по URI + HTTP-методы для смысла операции + передача JSON/XML.



REST-assured

Задача

Возвращаемся к нашей задаче: разработчики нам предоставили сервер в виде [запускаемого JAR-файла](#).

Чтобы запустить его, нужно выполнить в терминале команду: `java -jar app-mbank.jar` (ознакомьтесь с руководством по работе в терминале).

API

Разработчики сказали буквально следующее: "делаешь GET запрос на <http://localhost:9999/api/v1/demo/accounts>, в ответ получишь JSON":

```
[
  {
    "id": 1,
    "name": "Текущий счёт",
    "number": "•• 0668",
    "balance": 992821429,
    "currency": "RUB"
  },
  {
    "id": 2,
    "name": "Текущий счёт",
    "number": "•• 9192",
    "balance": 2877210,
    "currency": "USD"
  },
  {
    "id": 3,
    "name": "Текущий зарплатный счёт",
    "number": "•• 5257",
    "balance": 1204044,
    "currency": "RUB"
  }
]
```

REST-assured

Вы уже знакомы с Postman: инструмент, который позволяет вам делать запросы и даже производить тестирование с элементами автоматизации.

Но нам хотелось бы делать всё на Java. Для этого у нас есть [REST-assured](#):

```
dependencies {  
    testImplementation 'org.junit.jupiter:junit-jupiter:5.6.1'  
    testImplementation 'io.rest-assured:rest-assured:4.3.0'  
}  
  
test {  
    useJUnitPlatform()  
}
```

Важно: всегда читайте документацию (зависимости должны идти именно в этом порядке).

Достаточно часто автоматизаторы сначала проверяют запросы в Postman (либо аналоге), а затем переносят всё в код.

Простейший тест

```
@Test
void shouldReturnDemoAccounts() {
    // Подход: Given – When – Then
    // Предусловия
    given()
        .baseUrl("http://localhost:9999/api/v1")
    // Выполняемые действия
    .when()
        // get метод запроса GET
        // URI относительно baseUrl
        .get("/demo/accounts")
    // Проверки
    .then()
        .statusCode(200);
}
```

Всё, что делает наш тест — удостоверяться, что код ответа равен 200 (OK).



Что можно тестировать?

Вопрос к аудитории: что мы можем тестировать помимо кода 200?



Что можно тестировать?

В зависимости от постановки задачи мы можем тестировать:

- возвращаемые заголовки;
- тело ответа (если оно есть).

Заголовки

```
@Test
void shouldReturnDemoAccounts() {
    // Given – When – Then
    // Предусловия
    given()
        .baseUrl("http://localhost:9999/api/v1")
    // Выполняемые действия
    .when()
        .get("/demo/accounts")
    // Проверки
    .then()
        .statusCode(200)
        .header("Content-Type", "application/json; charset=UTF-8")
    // специализированные проверки – лучше
    .contentType(ContentType.JSON)
    ;
}
```

▼ Response Headers

[view parsed](#)

HTTP/1.1 200 OK

Content-Length: 433

Content-Type: application/json; charset=UTF-8

Connection: keep-alive

Тело ответа

Мы можем сопоставить и тело ответа целиком, но посмотрим на более интересные варианты:

```
@Test
void shouldReturnDemoAccounts() {
    ... // аналогично предыдущему коду
    .body("", hasSize(3))
    .body("[0].id", equalTo(1))
    .body("[0].currency", equalTo("RUB"))
    .body("[0].balance", greaterThanOrEqualTo(0))
    ;
}
```

Тело ответа

REST-assured использует [специальный синтаксис](#) для доступа к элементам внутри JSON:

- `" "` — корневой элемент;
- `[0]` — доступ к элементу массива по индексу;
- `.id` — доступ к свойству объекта.

Итого, получается что `[0].id` — это доступ к полю `id` первого элемента массива.

Кроме того, REST-assured предлагает использовать нам библиотеку [Hamcrest](#), которая содержит набор удобных `matcher`'ов.

Groovy GPath

На самом деле, под «капотом» REST-assured использует Groovy, а именно GPath для обработки этих выражений и если вы будете плотно работать с REST-assured, то хорошо бы его выучить:

```
@Test
void shouldReturnDemoAccounts() {
    ... // аналогично предыдущему коду
    .then()
        .statusCode(200)
        .contentType(ContentType.JSON)
        .body("every{ it.balance >= 0 }", is(true)) // все балансы неотрицательные
    ;
}
```



Тело ответа

Не всегда нам нужно проверить именно сами данные, иногда нам нужно проверить соответствие их определённой структуре и ограничениям (схеме).

Для JSON есть специальный проект [JSON Schema](#), который позволяет проверять документ на соответствие схеме.

С ним вы и познакомитесь в рамках ДЗ.



Continuous Integration



Continuous Integration

Вы уже сталкивались с термином Continuous Integration, давайте вспомним о нём и посмотрим на особенности работы с CI для задач тестирования внешнего приложения.

Вопрос к аудитории: что такое Continuous Integration?



Continuous Integration

Оригинал: Continuous Integration is a software development practice where members of a team integrate their work frequently, usually each person integrates at least daily – leading to multiple integrations per day. Each integration is verified by an automated build (including test) to detect integration errors as quickly as possible.

Martin Fowler

Вольный перевод: Continuous Integration (далее — CI) — это практика разработки ПО, при которой участники команды интегрируют изменения настолько часто, насколько это возможно (как минимум, ежедневно или несколько раз в день). Каждая интеграция верифицируется автоматической сборкой (включая тесты) для определения ошибок интеграции настолько быстро, насколько это возможно.

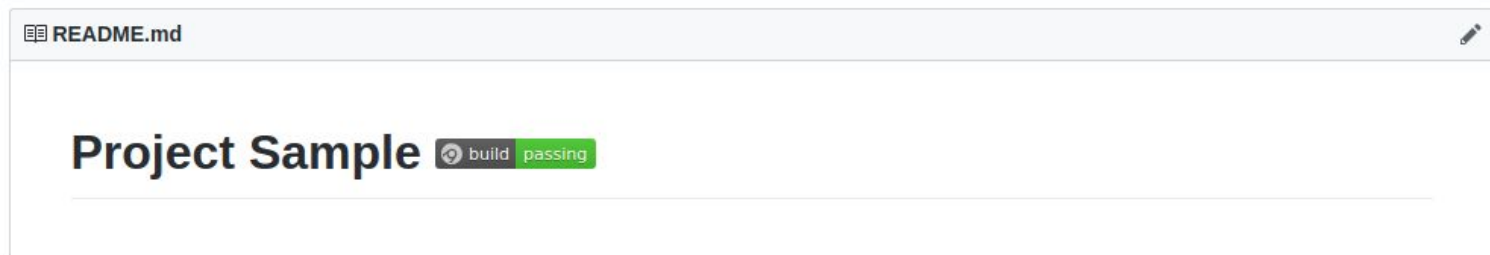
Мартин Фаулер

Continuous Integration

ISTQB в [Agile Tester Extension](#) даёт схожее определение, добавляя в него ряд нюансов.

Для нас это будет практика (подход к разработке), при которой для каждого изменения кода (git push) будет автоматически запускаться конвейер тестирования и сборки.

По результатам сборки мы будем видеть: либо сборка «Success», и код можно интегрировать; либо «Failed», и код нуждается в доработке.





CI

Теперь когда наши тесты — это клиент для сервера, умеющий сопоставлять фактические ответы с ожидаемыми, в полный рост встаёт вопрос: как же запускать сервер для тестов?

Вариантов достаточно много, мы рассмотрим несколько ключевых:

1. Делегировать эту задачу кому-то другому (SUT уже должен быть запущен).
2. Запускать сервер средствами CI.
3. Запускать сервер из тестов (Gradle или Java).

Давайте рассмотрим все три варианта подробнее.



«Не наша задача»

Достаточно удобный для нас вариант, нам нужно знать только URL SUT, на который нужно посылать запросы.

Тесты пишутся и запускаются как обычно.

Но всё ли так хорошо, как кажется на первый взгляд?

Вопрос к аудитории: какие потенциальные риски вы видите в этом варианте?

«Не наша задача»

Нам придётся столкнуться с рядом проблем, которые мы будем обсуждать на протяжении всего курса:

- периодическое падение тестов из-за недоступности сервера (например, из-за проблем сети между CI и сервером);
- сложность «управления» SUT (например, установки начального состояния, перезапуска в нужной конфигурации и т.д.);
- зависимость тестов от текущего SUT или его текущей версии (команда, отвечающая за тестовый сервер, может решить перезапустить его во время наших тестов или обновить).

Доходит до того, что иногда сервер, на котором развёрнута SUT, могут вообще временно «изъять под другие задачи». 😈



«Не наша задача»

К сожалению, это достаточно часто встречается на практике и приводит к необходимости разбирать **FAIL**'ы.

«Усталость» от разбора таких «интеграционных» проблем тестов приводит к выработке вредной привычки: если тесты падают, то, возможно, это из-за интеграции, **попробуем их просто перезапустить**.

Это очень плохо! Подобный подход приводит к тому, что доверие к тестам пропадает => тесты перестают использоваться (их результат не учитывается) => смысла в тестах нет.

Конечно же, мы описываем то, как должно быть, и в реальной практике вы достаточно часто встретитесь с подходом «перезапустим тесты».

Запуск из CI или из тестов

И CI, и Gradle, и Java позволяют нам запускать помимо тестов другие приложения (речь о процессах*).

Поскольку CI предоставляет возможность запуска произвольных команд, то мы вполне можем этим воспользоваться исходя из того, какую командную оболочку (CMD, PowerShell, Bash и т.д.) и операционную систему предоставляет CI.

В Gradle мы можем использовать [Exec](#).

В Java для запуска команд существуют инструменты `Runtime.exec()` и `ProcessBuilder`.



Запуск из CI или из тестов

Достаточно удобный для нас вариант: мы сами запускаем SUT; проблем сети, перезагрузки сервера с SUT или обновления SUT другой командой не будет.

Но всё ли так хорошо, как кажется на первый взгляд?

Вопрос к аудитории: какие потенциальные риски вы видите в этом варианте?



Запуск из CI или из тестов

Эти варианты также обладают рядом проблем:

- SUT должна быть спроектирована для подобного запуска (установка и запуск SUT могут представлять из себя комплексную процедуру);
- зависимости SUT (SUT может требовать баз данных, систем очередей или интеграции с реальными системами);
- у вас должен быть актуальная версия SUT (тестировщики и разработчики достаточно часто работают в разных подразделениях);
- требования SUT должны быть соизмеримы с техническими характеристиками CI.

Q & A

Q: Почему мы всё время говорим о разных вариантах, у каждого из которых есть куча недостатков? Почему мы не изучаем **одно правильное и работающее всегда решение?**

A: Потому что его нет. Вам всегда придётся выбирать из нескольких вариантов исходя из того, какой у вас проект, продукт, команда, сроки, бюджет и цели.

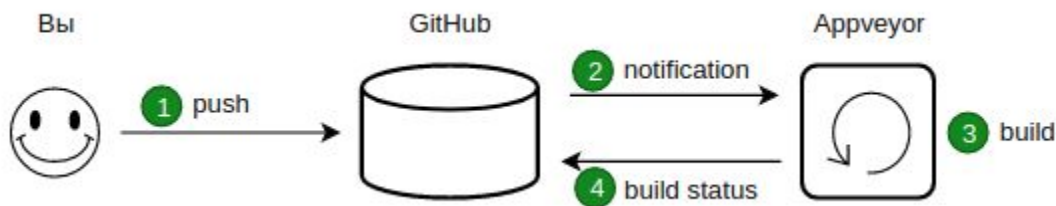
К этому нужно привыкнуть: идеального решения (как и идеального процесса) нет и не будет. Наша задача — показать вам возможные варианты, их плюсы и минусы, чтобы могли принимать решения **осознанно**.

Мы отработаем все варианты запуска в рамках ваших ДЗ (часть в ДЗ к этой лекции, часть — в последующих).

CI

Настройка CI будет пошагово описана в вашем ДЗ.

Вы настроите полную связку из GitHub и Appveyor. Обычно, общий процесс выглядит в простейшем варианте следующим образом:



Q & A

Q: Почему Appveyor а не GitHub Actions?

A: Потому что вы должны познакомиться и с другими системами CI. GitHub Actions "скрывает" от нас с вами некоторые вещи. Работая с Appveyor, вы научитесь настраивать целый класс облачных CI: Travis CI, Circle CI и т.д. (т.к. они аналогично интегрируются с GitHub).



Итоги



Итоги

Итак, мы рассмотрели достаточно много важных тем на сегодня:

- Что такое API
- HTTP, REST, JSON
- Тестирование API и использование REST-assured
- CI



Домашнее задание

Давайте посмотрим ваше [домашнее задание](#).

- Вопросы по домашней работе задавайте **в чате** мессенджера Slack.
- Задачи можно сдавать **по частям**.
- Зачёт по домашней работе проставляется после того, как **приняты все задачи**.

**Задавайте вопросы и
пишите отзыв о лекции!**

Артем Романов



[romanov-artyom](https://romanov-artyom.netology.ru)



[Романов Артём](#)



[@Artem_Telegram](#)