API TEST AUTOMATION

DEEP DIVE INTO PROTOCOL STACK

About ☺

Anton Semenchenko

- *Co-founder \ activist of COMAQA.BY, CoreHard.BY and InterIT communities, co-founder of DPI.Solutions, manager at EPAM Systems. More than 16 years of experience in IT. Specializes in low-level development, QA automation, management, sales.*

COMAQA.BY
QA automation community

COREHARD
C++ COMMUNITY

Agenda ☺

COMAQA.BY
QA automation community

1. OSI Model
2. Web service testing
3. Tools

OSI model

7 APPLICATION	DATA	
6 PRESENTATION		
5 SESSION		
4 TRANSPORT	SEGMENT	PORT
3 NETWORK	PACKET	IP
2 DATA LINK	FRAME	MAC
1 PHYSICAL	BITS	

COMAQA.BY
QA automation community

7 layer OSI model:

1. It is not strict representation of actual architecture, but rather **conception** that helps to understand interaction over the network. I believe almost all of you already saw this model, or at least heard about it. OSI stands for Open System Interconnection

COMAQA.BY
QA automation community

7 layer OSI model:

2. **The first layer is the physical layer**. This is **how do devices connect**. Are they connecting across wires? Or maybe with fiber?
3. **The second layer** up here is referred to as the **DataLink Layer**. The DataLink layer identifies **how the electronics signal should be translated**. Is it Ethernet or wireless. It could be 802.11 for wireless or 802.3 for Ethernet standards. DataLink layer is saying **how the bits of data are packaged**.
4. **The third level** up here is the **network layer**. The network layer is where we introduce the concept of addressing. **From address, and to address**.
5. **These first three layers** are defined under the OSI model to be **controlled by the network**. The next four are the system or software controlled components.

COMAQA.BY
QA automation community

7 layer OSI model:

6. **Layer number** four is called the **Transport Layer**. Transport is telling us how we are going to **package the data because we don't send all of the content at one time**. We **send it in packets**. They are small pieces of data that are taken from a large bundle of data on the system here … and chopped into smaller pieces. …. and then sent across the internet and reassembled at the far side. So transport mechanisms disassemble these pieces, send it across the internet and reassemble it at the other side. Transport Layer is very important for reliability of the data. It ensure that I actually get all of the message. If I get part of the message from one source and part from another, if I can properly put them back together, I have the complete message. The transport tells me, how I break down and put back together the data.

COMAQA.BY
QA automation community

7 layer OSI model:

7. Next level up here is Session. **This is where sessions are handled**. It **opens, close, and able to recover a session**. And it allows to synchronize information from different streams.
8. All right, so one layer up is **layer 6 is the presentation layer**. **It handles the language or format that am I using**? And so, when I'm talking about the communication that's going across a particular session, across a particular protocol, I could use multiple languages. If I use HTTP and Port 80 the language might be **HTML**. Or I could use other formats. I could use **XML**. There's a lot of other languages that might be used across HTTP. Also some sources claims that SSL leaves here, in presentation layer.

COMAQA.BY
QA automation community

7 layer OSI model:

9. When we move up here to the **seventh layer** here, we're gonna talk about the **Application Layer**. The Application Layer is the actual application that's going to manage the data and converts information to data and backwards. When **I say Application Layer, our example here would be browsers, at least for the case we're talking about HTTP. Or HTTP client for API**.

API test automation

GET http://www.example.com/customers/33245

200 Ok + Payload

CLIENT

7 APPLICATION	Request data
6 PRESENTATION	Data format
5 SESSION	:80/:443
4 TRANSPORT	UDP/TCP
3 NETWORK	IP
2 DATA LINK	MAC
1 PHYSICAL	

SERVICE

COMAQA.BY
QA automation community

The **OSI model** describes **how to encapsulate the pieces of data and send trough the network**. Well, here's a packet of data. The blue column. Again, this is by example, I am not trying to describe precisely standards of an IP packets content, because this if far from this workshop scope.

I am gonna start here at layer seven, the Application Layer. Here in the packet, the first thing we're gonna do in layer seven's gonna put data in here, some payload. **This data could be encrypted**. I could put a flag that says that encryption is used. **As an example VPN**.
The next layer of the model is six and this is where I would put the format. I need a way to identify what the format of the data is.

I have layer five and layer five is about my port or my protocol. In here, I'm identifying what mechanism I'm using. I've got Port 80, if it's HTTP for example.

My layer four here is the Transport Layer. The two most common forms of transport are **UDP**, **User Datagram Protocol** and **TCP**, **Transmission Control Protocol**.

In UDP sequence of delivery, sequence of transmission doesn't mean as much. The other option here is TCP, Transmission Control Protocol. When I send content via TCP sequence matters. TCP is more important that it be delivered and reassembled sequentially.
Let's just say I have a 20-word message. And I receive part one, part two, part three, part five. Wait I need part four, Retransmit part four so that I can reassemble the message. If I'm doing UDP, it might not bee needed.

So in layer four I have information that allows me to recover the content of multiple packets. I don't send all of my data in one packet. I chop it into pieces.

The bottom three network layers. Layer two is where the MAC address exists. This is the physical ID of a network card in a Pcso I will have the FROM and TO MAC addresses.

And finally layer 3 is for IP address. This is the very last piece of data added to packet and now, I can send this packet using the OSI model back and forth between the two devices.

The OSI model is an idealized abstract model, and there are no protocols in use today that follow it. HTTP was created without regard to the OSI model, so there's no point to trying to make it absolutely fit. It just should help to imagine what's going on when you send request somewhere.

Request - response structure and types

```
POST /maps/api/place/add/json HTTP/1.1

Host: maps.googleapis.com
Content-Type: application/json
Accept: application/json

{"location": {
   "lat": -33.8669710,
   "lng": 151.1958750
  },
  "name": "Andrew was here",
  "phone_number": “+375(29)3704858",
  "types": ["bar"],
}
```

Request line: METHOD|URI|HTTP version

HEADERS

MESSAGE BODY

COMAQA.BY

QA automation community

COMAQA.BY
QA automation community

So I will go ahead and move clothe to main theme of this workshop. In most cases when you are working with API, **it means that you will send some request and check some response**. This is how it usually works.
So this is request example. It consists from request line, headers and message body. Message body is optional. For Get it optional, and HEAD request should not have this part.
Ok, **request line** contains **Request method**, **URI of resource**, and **version of protocol**. Header section contains all information about request itself. The only one header that should be presented in any case is host. This is address of server that we are talking to. Cookies, content types, session information will be stored in this section.
And request body itself. This part contains the data, that should be sent to the server.

Request - response structure and types

COMAQA.BY
QA automation community

```
HTTP/1.1 200 OK
```

```
Content-Type: application/json;
Content-Length: 352
```

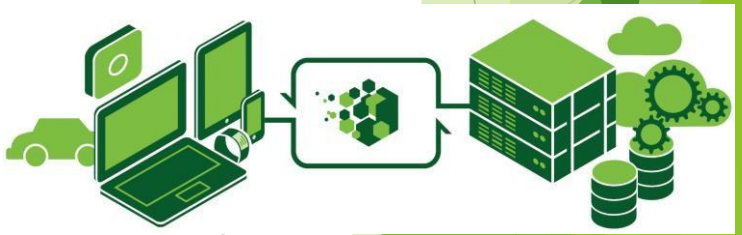
```

```

```
{
  "id" : "50ea6150666e0de4b097",
  "place_id" : "qgYvCi0jlhZTgzNGI",
  "scope" : "APP",
  "status" : "OK"
}
```

Status-Line:
HTTP-Version|Status-Code|Reason-Phrase
HEADERS

MESSAGE BODY



COMAQA.BY
QA automation community

On each request server replies with response. Typically **response will contain Status line, headers and message body**.

Status line shows execution status of request. Are there any errors, or request processed successfully. There are pretty big amount of status codes, I'll describe them better in next slide.

Headers part is the same with request. It contains information about response, session data, cookies, and size of message body.

And a message body itself. It could be JSON, XML, HTML, mime types...

Request - response structure and types

COMAQA.BY
QA automation community

2xx: Success	200 OK 201 Created
4xx: Client Error	400 Bad Request 403 Forbidden 404 Not Found 405 Method Not Allowed 410 Gone
5xx: Server Error	500 Internal Server Error

WEB SERVICE

About status codes. They are represented and described in RFC 7231.

Value of status code represent result of response execution. There are five subgroups. One hundred to five hundred.

The most common are **two hundred**, server should return this code in case of successful request processing.

Four hundred status codes should be returned in case if something went wrong on requestors side.

And the **500** (Internal Server Error) status code indicates that the server encountered an unexpected condition that prevented it from fulfilling the request.

REST and CRUD

http://www.awesomeportal.com:1234/path/version/resource?foo=foo%20bar

protocol | address | port | resource path | query

CREATE

POST http://www.example.com/customers + *payload*

201 Created + Location

READ

GET http://www.example.com/customers/33245

200 Ok + Payload

UPDATE

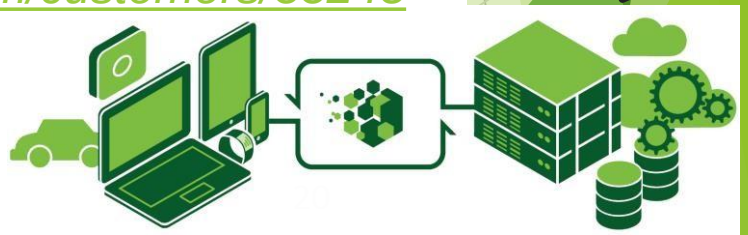
PUT http://www.example.com/customers/33245

200 Ok

DELETE

DELETE http://www.example.com/customers/33245

200 Ok



COMAQA.BY
QA automation community

COMAQA.BY
QA automation community

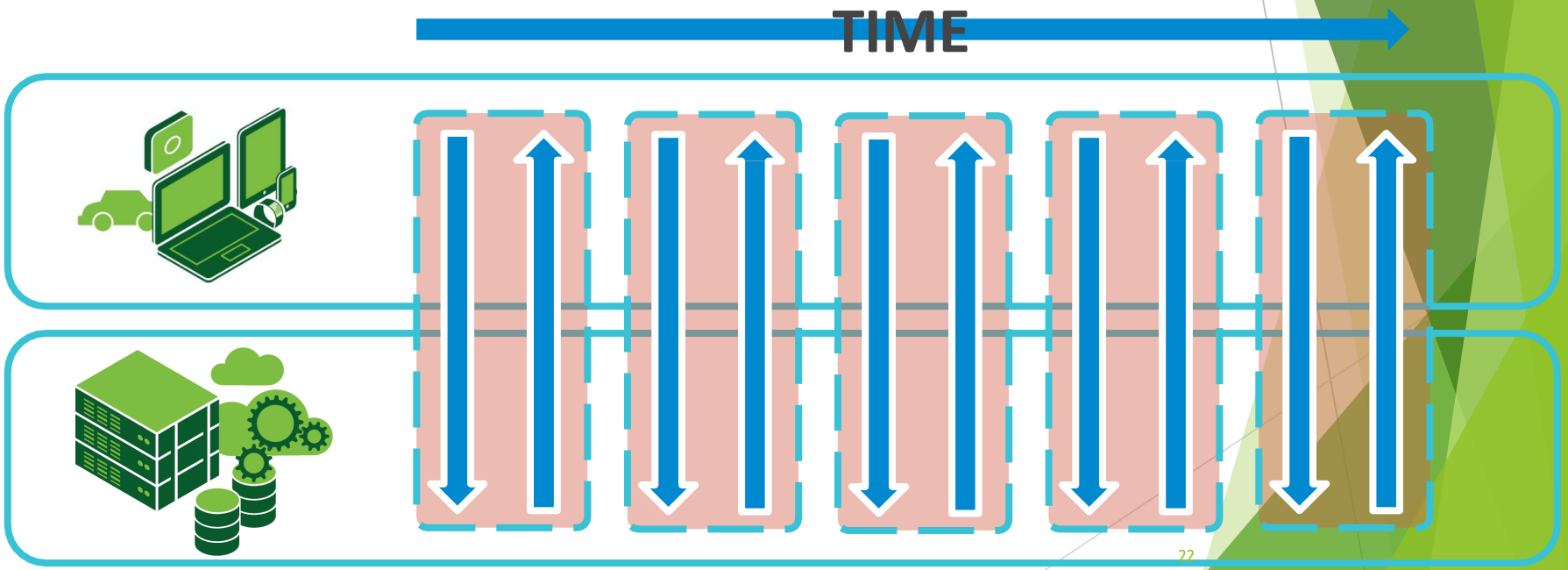
And a few words about REST:

1. **REST is architectural style of building web services**. Rest service usually represents all data as resources, and to interact with this resources we use HTTP methods. So to send request to interact with some data, request will looks like: type of protocol, then will be the host, or address, or maybe you will wont to use IP and port. The next part is resource path, and here could be version of endpoint. And last part is query, it contains parameters that you want to send to resource.
2. Each resource could be **created**, **read**, **updated** or **deleted**. Well it obviously depends on access and what logic is implemented in web service? But in general: this is called **crud** operations. The main idea is that **data should be created with post method**, **read with GET method**, **to DELETE data, method DELETE should be used**, and to **update, method update should be used**.

Client – service interaction

COMAQA.BY
QA automation community

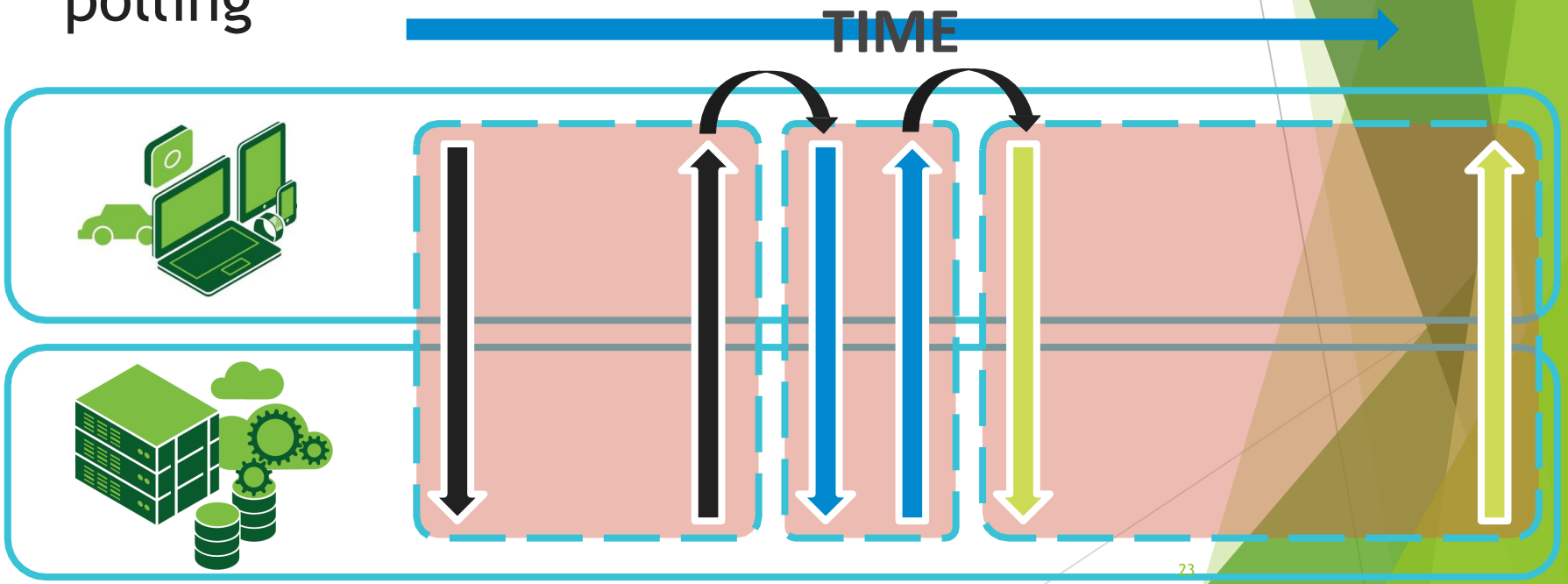
Polling



Client – service interaction

COMAQA.BY
QA automation community

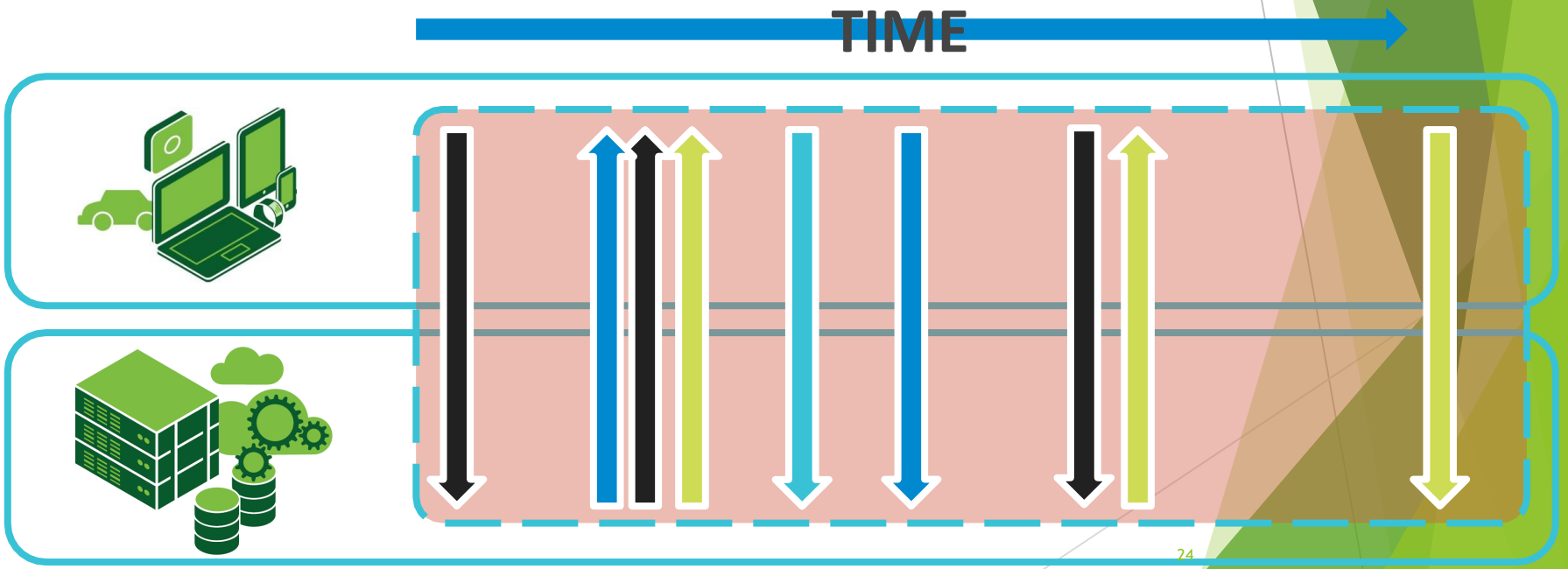
Long-polling



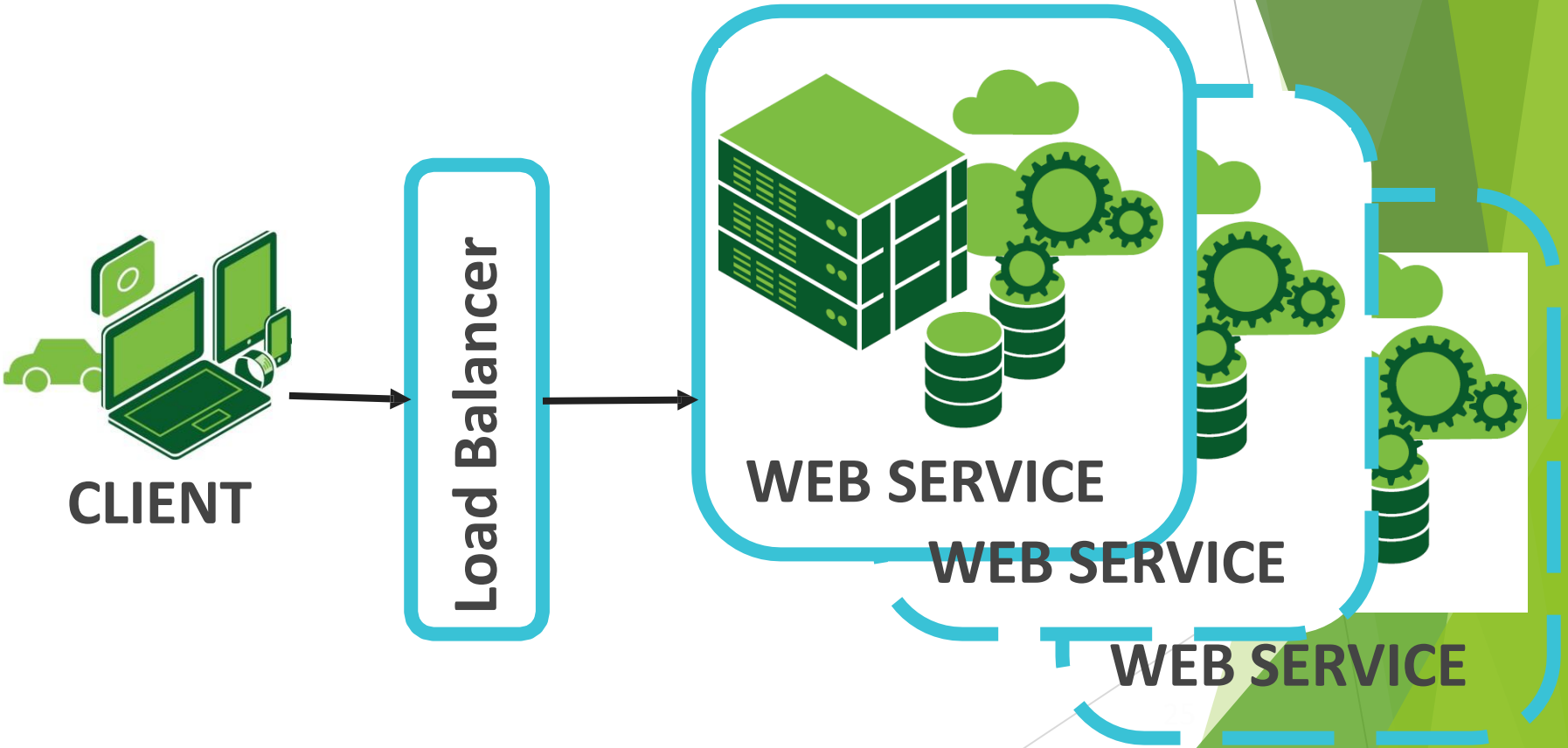
Client – service interaction

COMAQA.BY
QA automation community

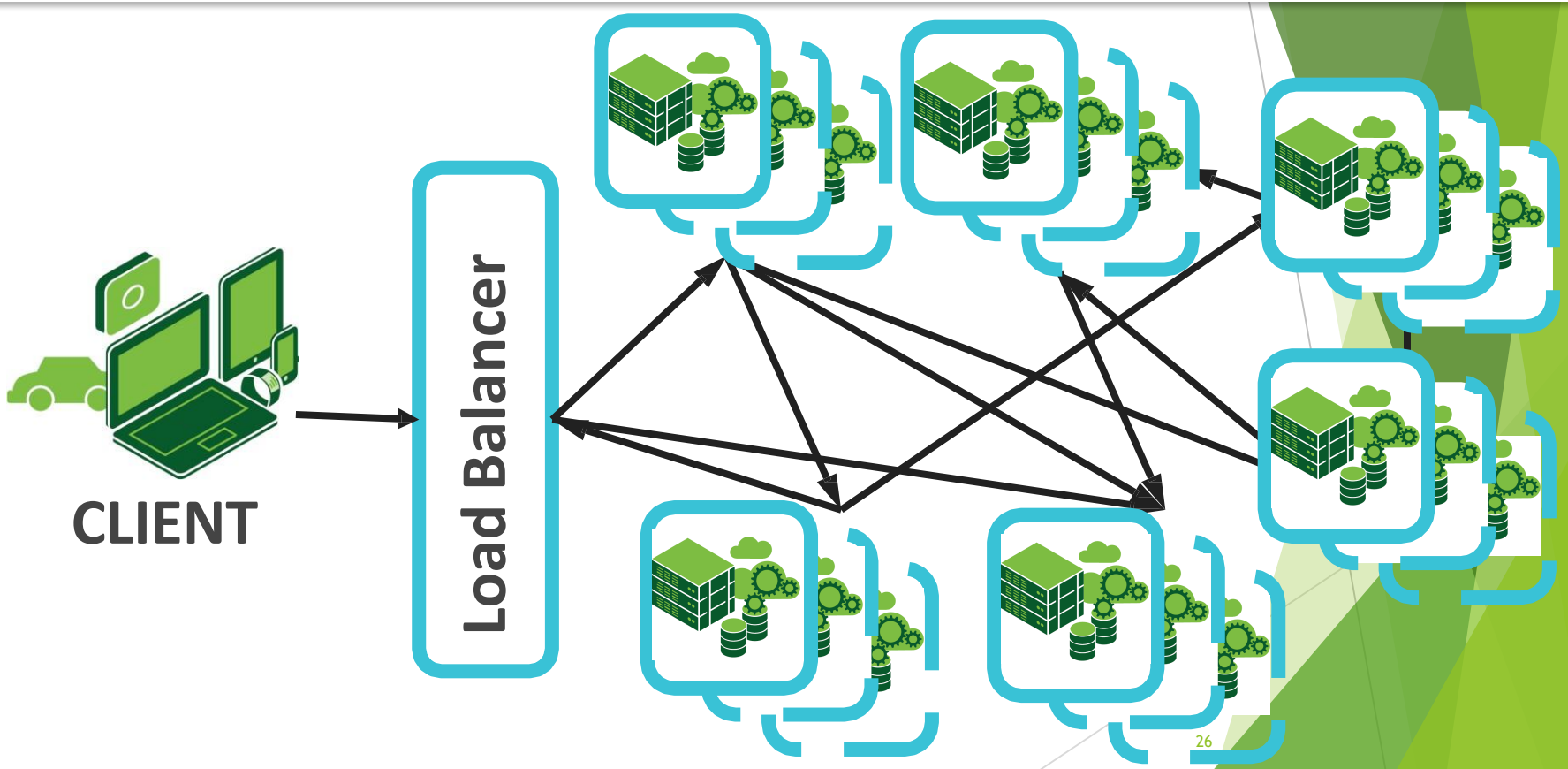
Web Socket



Client – service interaction

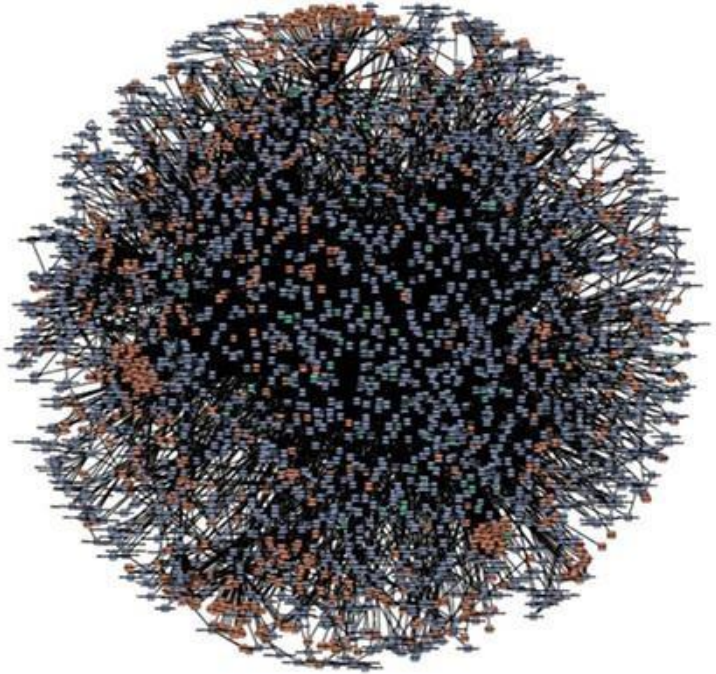


Client – service interaction



Client – service interaction

COMAQA.BY
QA automation community



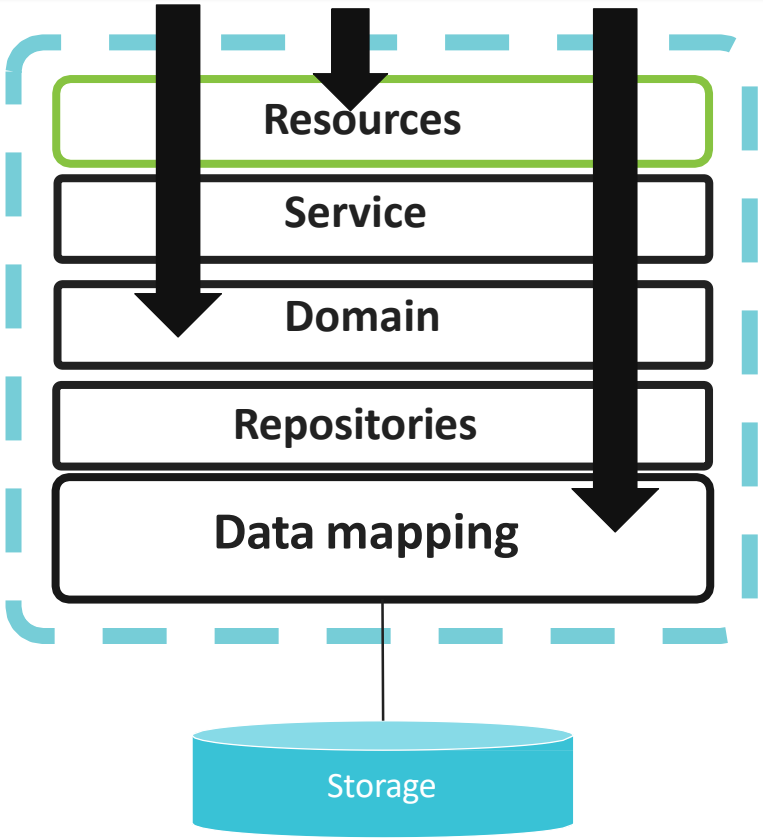
amazon.com

NETFLIX

And in big projects each module could have a lot of external connections. **Be careful with estimations about how much time you need to mock dependencies**.

This in nice example. This is scheme of Amazon and Netflix micro services structure. This picture is at least few years old, so I am sure that in present time it even more impressive.
By the way, **this is called “Death star” diagram**. **Pretty nice way to show that system structure is really complicated**.

Web service testing



WEB SERVICE

It is nice and helpful if you **have access to web service code and understand its structure and how it works**. But in case if you don't **it could be tricky to figure out how and what to check with backend tests**. In general most web services could be present as following structure. It is doesn't represent real structure, but just very-very high level conception.

So web service will have some **resources or some endpoints, that are exposed to client**.
Also there are will be some **layer that represents main logic, something that service is responsible for**.
And there are will be some **layer that interacts with data storage**. **Back end is usually about data**. Maybe in your case it will read from database, or store some data, or executes some logic if data changed. This is what service usually responsible for.

COMAQA.BY
QA automation community

So taking this structure into account, you cold plan different sets of test, and you could **be sure that all layers will be covered**.

For resources there will be tests that check web service contract. Obviously **positive tests**. **But also how backend process requests without mandatory fields, with wrong data type, and which data is optional**. And so on. **Don't expect that only valid requests will be sent to backend**. **Check the contract, that backend really returns what is claimed in documentation**.

The next set of tests will be **tests that check logic (?)**, implemented in web service. This part of tests fully depends on what's going on in your project. **This is hard to give some general idea what should be checked here. But usually this is data itself, like does back end return proper values**.

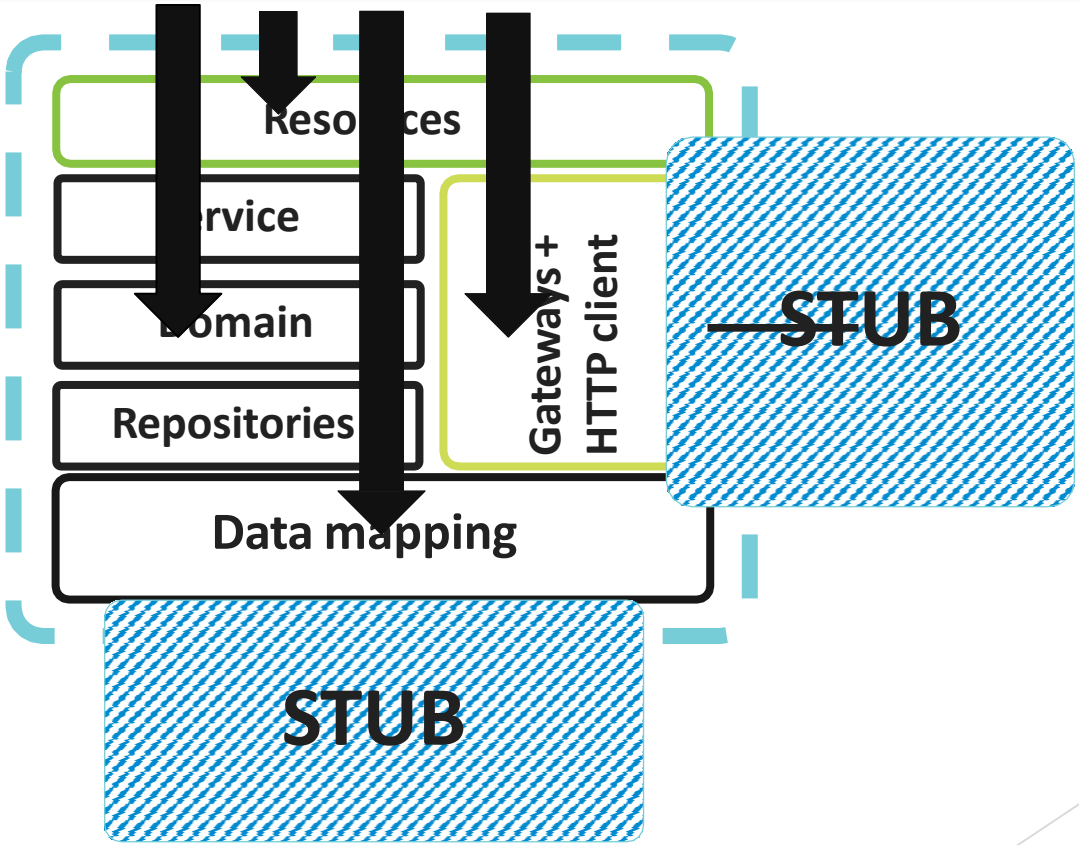
COMAQA.BY
QA automation community

Actually in some cases you can't even validate anything. Like if you have backend that sends you financial market data. You can't predict what you will receive. So you will have to **find workarounds, or maybe customer will decide that data values itself shouldn't be checked, just check that it has proper structure**. This is possible to.

And the last set **is tests to check interaction with data storage (?!)**. Try to create some data with backend, retrieve, update. What your service allows to do. **Don't expect that there are will be only valid data in database**. If field is nullable, add test that will check that backend could sussessfully process this kind of data with nulls in database.
And again this is high level idea. Because this is not possible to strictly say that you should do like this and this, because there is a big variety of approaches and architectures of back ends. So each case will have specific tests but you could use this conception like check list, to verify that you are not missing something.

Web service testing

COMAQA.BY
QA automation community



WEB SERVICE

COMAQA.BY
QA automation community

In case of **micro services** the main idea is the same but because it interacts with another micro services it could have **a lot of external dependencies**. So we should take into account that it will have its own client to interact with another parts of back end. Yes, monolithic backend will probably have it, but this is more important in this case. **You still can test this kind of backend in the same way as monolithic. But ALSO… you will probably want to test micro services separately**. You saw death star diagram. It will be tooooo complicated to handle whole system together. **Also one of pros of this architecture, that you could redeploy modules separately**. **So it will be nice to be able to check modules separately before they will be deployed**. **This could be achieved by creating hermetic environments for each module**. **All external dependencies should be replaced by mocks or stubs. And then you could test module in isolation.**

COMAQA.BY
QA automation community

Pros of this approach are obvious: you could fully control data that is coming from external dependencies and you could use mocked database or in memory database with some test data.
But also there are some aspects that you should **take into account**:

1. The first one is **stubs** itself. You should **invest some amount of time to implement stubs, or fake services, or fake http client**. It could be not so complicated for one two simple external dependencies. But **it could become disaster if service has bunch of dependencies and each of them has complicated behavior**. And it will be even more complex if this **module is in development**. Team will add dependencies faster then you will be able to mock them.
2. Also you should decide **how you will implement mocks**. As an example: **Is it more convenient to deploy module with fake http client, or replace some service class with mock. Or the second option is to run module with fake hosts parameters and start fake services as separate web services**. So it will send requests to fake services instead real ones.

COMAQA.BY
QA automation community

So there is a lot of staff to think about before you will start back end automation on your project. *As you see it could be pretty simple, and you will be able to deliver hundreds of tests in three months, or you could spend half year on mocks and even not start with tests*.

You should ask yourself:

1. How my **tests will get access to backend**?
2. Am I **able to authenticate**?
3. Can I **create or request enough test users**?
4. Am I **able to manage test data, or test DB should be added**?
5. Can I **use test environment**, or I **should deploy application form test framework**?
6. Should I **create any mocks**? As an example module could interact with prod data, so you could not use real dependencies in tests because of security concerns.

COMAQA.BY
QA automation community

The most important conceptions:

1. **Dependency Injection DI \ Inversion of control IoC**
2. **Mock Framework**
3. Domain Specific Languages development
4. Code-generation
5. *BDD ..?*
6. *The Single Responsibility Principle, SRP*

COMAQA.BY
QA automation community

The most important Design Patterns:

1. Builder
2. Fluent interface
3. Singleton
4. Abstract Factory
5. Factory Method
6. Creation Method
7. Decorator

COMAQA.BY
QA automation community

The most important idioms:

1. Generic
2. Reflection
3. Lambda

COMAQA.BY
QA automation community

Dependency injection is a technique whereby one object (or static method) supplies the dependencies of another object. A dependency is an object that can be used (a service). **An injection is the passing of a dependency to a dependent object** (a client) **that would use it**. The service is made part of the client's state. **Passing the service to the client, rather than allowing a client to build or find the service, is the fundamental requirement of the pattern.**

The intent behind dependency injection is to **decouple objects** to the extent that **no client code has to be changed simply because an object it depends on needs to be changed to a different one**. This permits **following the Open / Closed principle**.

COMAQA.BY
QA automation community

Dependency injection is one form of the broader technique of inversion of control. As with other forms of inversion of control, dependency injection supports the **dependency inversion principle**. **The client delegates the responsibility of providing its dependencies to external code (the injector)**. The client is not allowed to call the injector code; it is the injecting code that constructs the services and calls the client to inject them. This means the client code does not need to know about the injecting code, how to construct the services or even which actual services it is using; **the client only needs to know about the intrinsic interfaces of the services because these define how the client may use the services**. **This separates the responsibilities of use and construction**.

There are three common means for a client to accept a dependency injection: **setter-, interface- and constructor-based injection**. Setter and constructor injection differ mainly by when they can be used. Interface injection differs in that the dependency is given a chance to control its own injection. **Each requires that separate construction code (the injector) takes responsibility for introducing a client and its dependencies to each other**.

COMAQA.BY
QA automation community

The **Dependency Injection design pattern solves problems** like:

1. How can **an application or class be independent of how its objects are created**?
2. How can the **way objects are created be specified in separate configuration files**?
3. How can **an application support different configurations**?

Creating objects directly within the class that requires the objects is inflexible because it commits the class to particular objects and makes it impossible to change the instantiation later independently from (without having to change) the class. **It stops the class from being reusable if other objects are required**, and **it makes the class hard to test because real objects can't be replaced with mock objects**.

A class is no longer responsible for creating the objects it requires, and it doesn't have to delegate instantiation to a factory object as in the Abstract Factory design pattern.

COMAQA.BY
QA automation community

Mock-object, mocking:

In object-oriented programming, **mock objects are simulated objects that mimic the behavior of real objects in controlled ways**, most often as part of a software **testing initiative**. A programmer typically creates a mock object **to test the behavior of some other object**.

COMAQA.BY
QA automation community

Mock objects can simulate the behavior of complex, real objects and are therefore useful when a **real object is impractical or impossible** to incorporate into a test**. If an object has any of the following characteristics, it may be useful to use a mock object in its place**:

1. the object **supplies non-deterministic results** (e.g. the current time or the current temperature);
2. it has **states that are difficult to create or reproduce** (e.g. a network error);
3. it is **slow** (e.g. a complete database, which would have to be initialized before the test);
4. it **does not yet exist or may change behavior**;
5. it **would have to include information and methods exclusively for testing purposes** (and not for its actual task).
6. uses **third-party services**

For **example**, an **alarm clock program** which causes a bell to ring at a certain time might **get the current time from a time service**. To test this, the test must wait until the alarm time to know whether it has rung the bell correctly. If a mock time service is used in place of the real time service, it can be programmed to provide the bell-ringing time (or any other time) regardless of the real time, so that the alarm clock program can be tested in isolation.

COMAQA.BY
QA automation community

«**The Pragmatic Programmer: From Journeyman to Master**»

1. Domain Languages
2. The Power of Plain Text
3. Code Generators
    a. Passive
    b. Active
4. Code That’s Easy to Test
5. Ubiquitous Automation
6. Ruthless Testing

COMAQA.BY
QA automation community

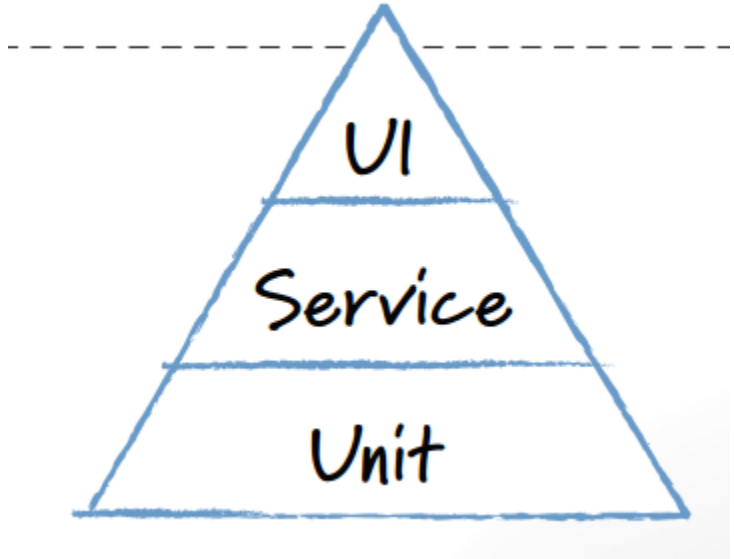
«**Программист-прагматик путь от подмастерья к мастеру**»

1. Языки, отражающие специфику предметной области
2. Преимущество простого текста
3. Генераторы текстов программ
    a. Пассивные
        i. *Приемлемая точность (правка сгенерированного кода руками)*
    b. Активные
        i. *Часть процесса сборки*
4. Программа, которую легко тестировать
5. Вездесущая автоматизация
6. Безжалостное тестирование

COMAQA.BY
QA automation community

Test Pyramid: definition

«An effective test automation strategy calls for automating tests at three different levels, as shown in Figure, which depicts **the test automation pyramid**»



COMAQA.BY
QA automation community

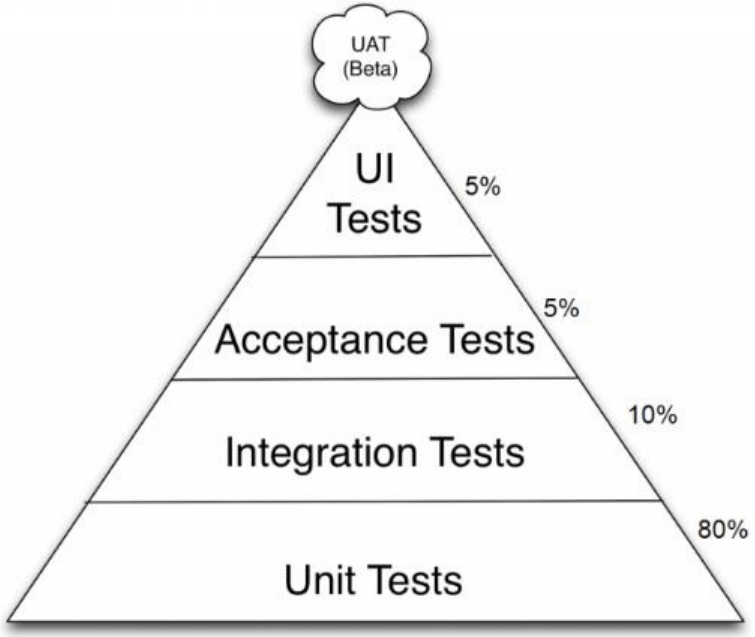
Test Pyramid with percentage #1

UI - 5%

Acceptance – 5%

Integration – 10%

Unit - 80%



COMAQA.BY
QA automation community

Test Pyramid with percentage #2

UI – 1%

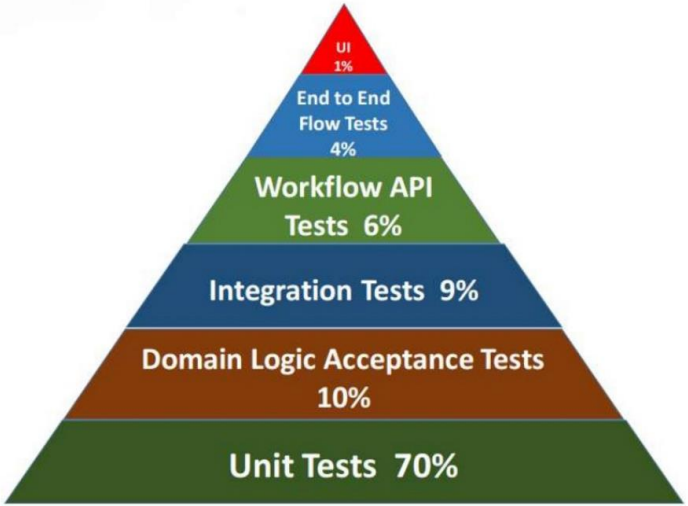
End to End Flow – 4%

Workflow API – 6%

Integration – 9%

Domain Logic Acceptance – 10%

Unit – 70%

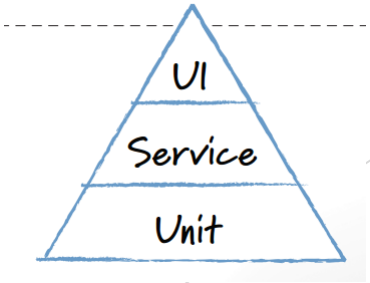


COMAQA.BY
QA automation community

Test Pyramid: Unit Testing

- «**Unit testing should be the foundation of a solid test automation strategy** and as such represents the **largest part of the pyramid**. Automated unit tests are wonderful because they **give specific data to a programmer—there is a bug and it's on line 47**.»

- «Unit tests are usually written in **the same language as the system**, programmers are often most comfortable writing them.»

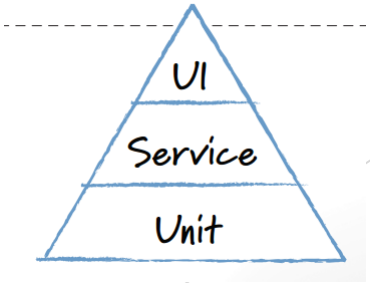


COMAQA.BY
QA automation community

Test Pyramid: Service

- «Although I refer to the **middle layer of the test automation pyramid as the service layer**, I am not restricting us to using only a service-oriented architecture. All applications are made up of various services. In the way I'm using it, a service is something the application does in response to some input or set of inputs.»

- «Service-level testing is about testing the services of an application separately from its user interface.»



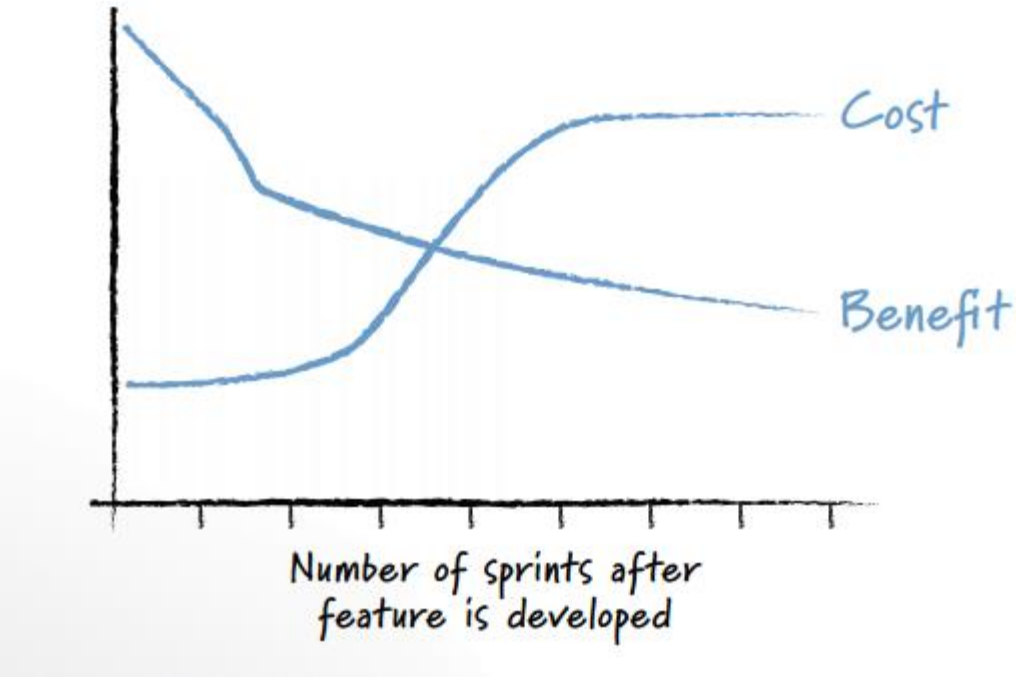
COMAQA.BY
QA automation community

The remaining role of user Interface Tests

- “But don’t we need to do some user interface testing? Absolutely, but far less of it than any other test type.
- Instead, we run the majority of tests (such as boundary tests) through the service layer, invoking the methods (services) directly to confirm that the functionality is working properly. At the user interface level what’s left is testing to confirm that the services are hooked up to the right buttons and that the values are displaying properly in the result field. To do this we need a much smaller set of tests to run through the user interface layer.
- Where many organizations have gone wrong in their test automation efforts over the years has been in ignoring this whole middle layer of service testing. Although automated unit testing is wonderful, it can cover only so much of an application’s testing needs. Without service-level testing to fill the gap between unit and user interface testing, all other testing ends up being performed through the user interface, resulting in tests that are **expensive to run, expensive to write, and brittle**”

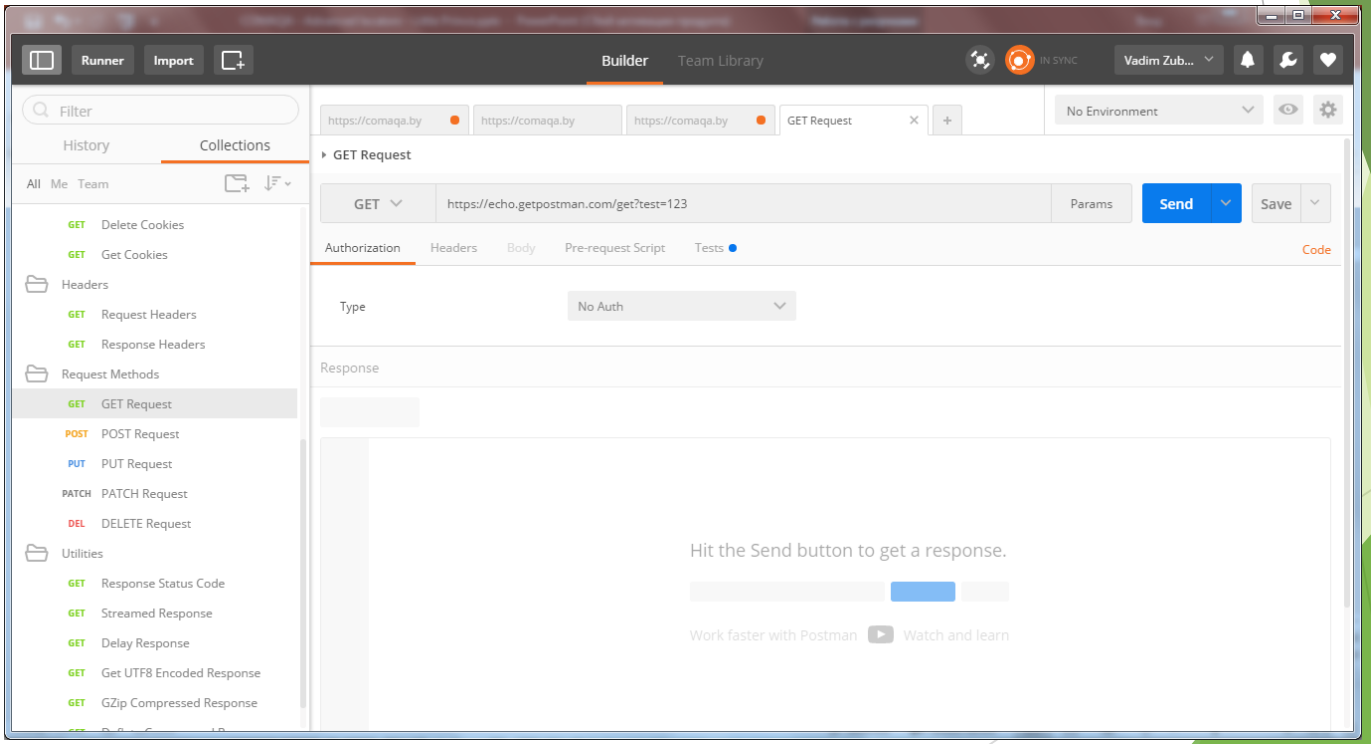
COMAQA.BY
QA automation community

Find a balance ☺



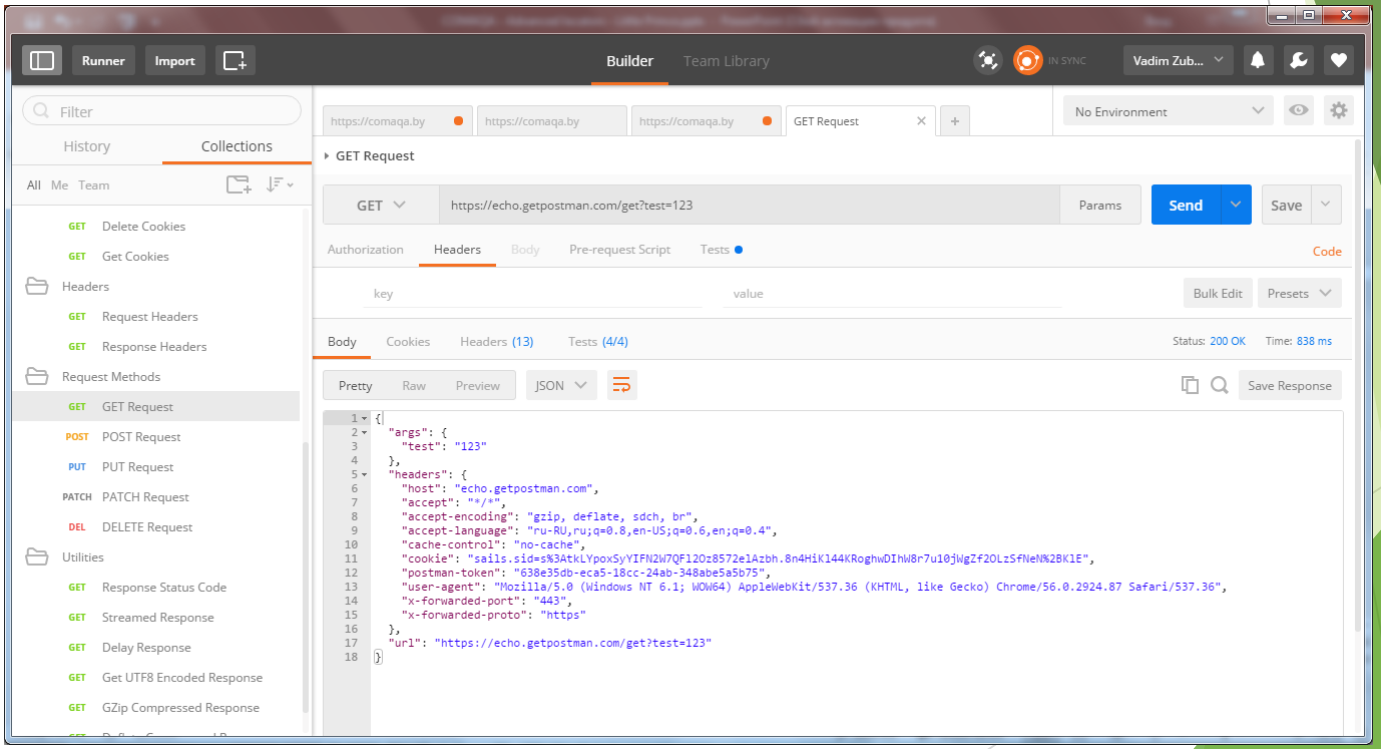
COMAQA.BY
QA automation community

POSTMAN



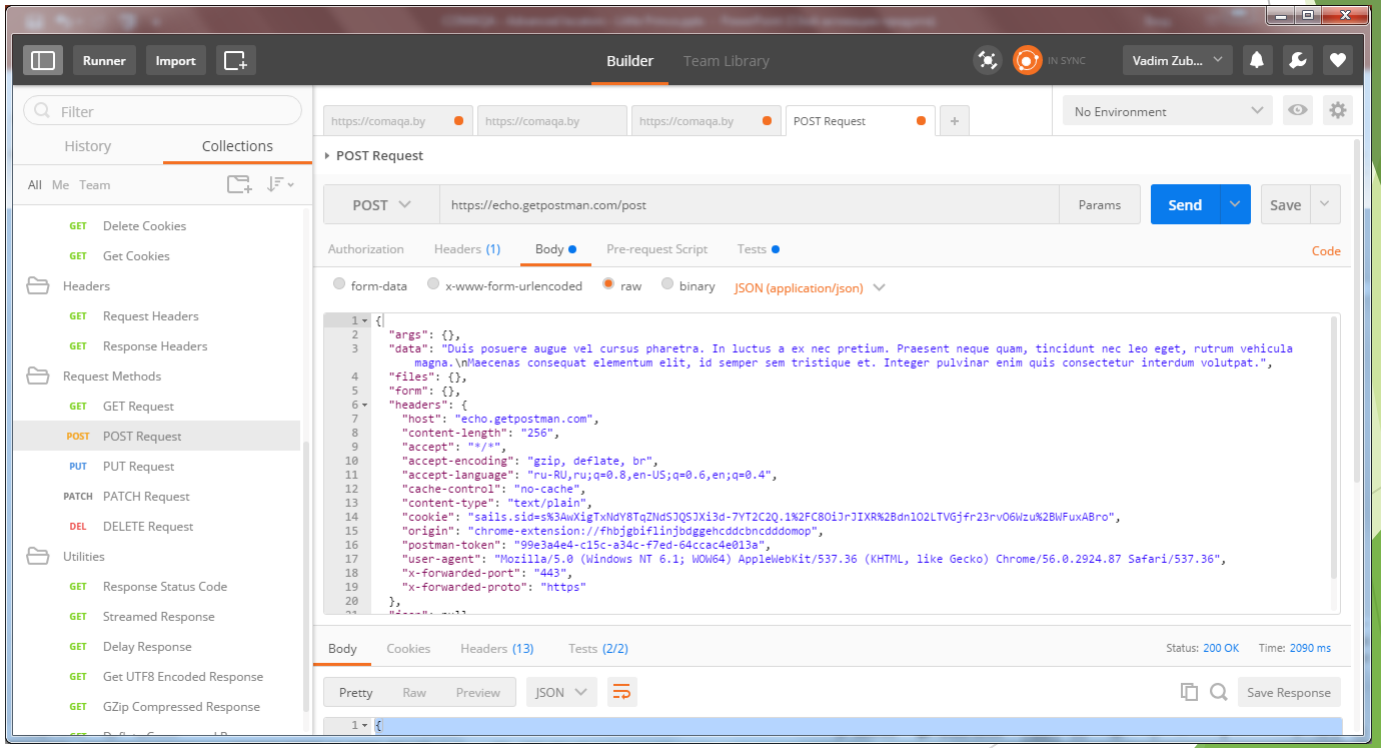
COMAQA.BY
QA automation community

POSTMAN



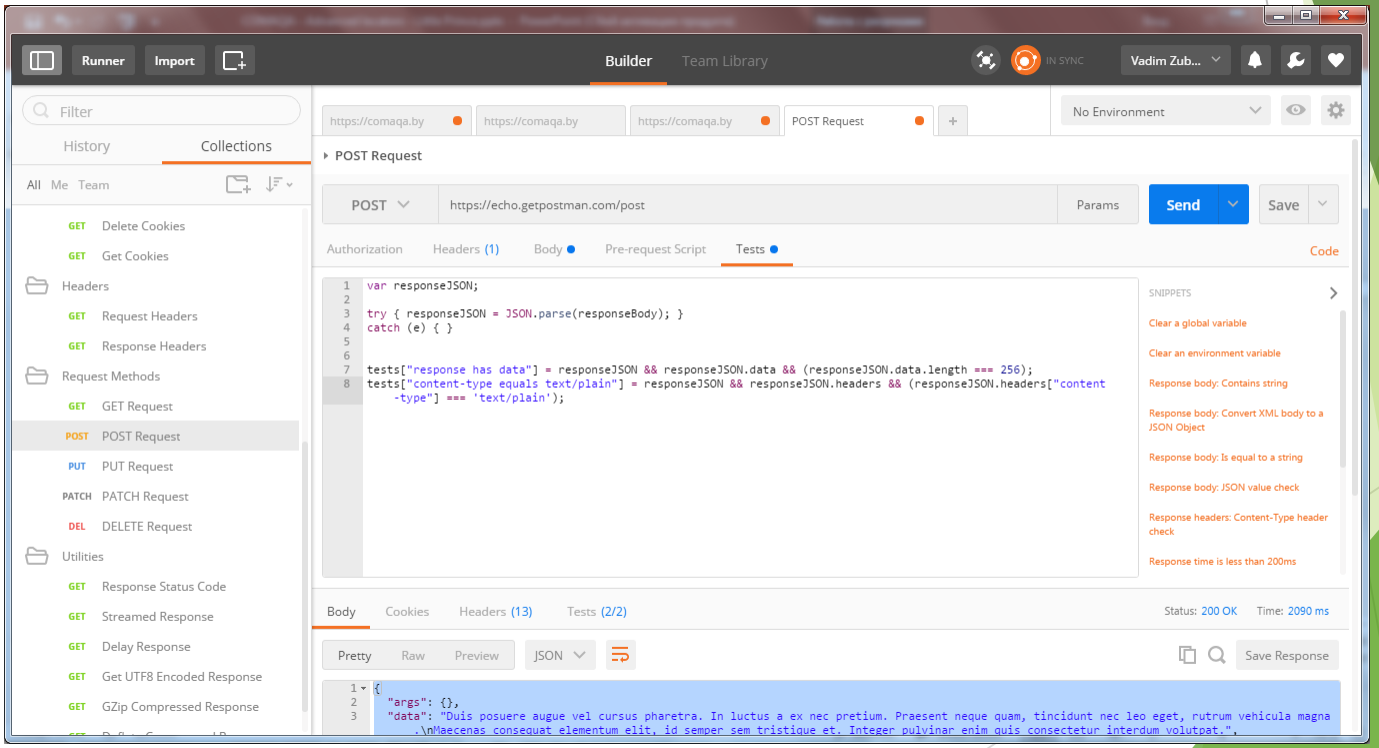
COMAQA.BY
QA automation community

POSTMAN



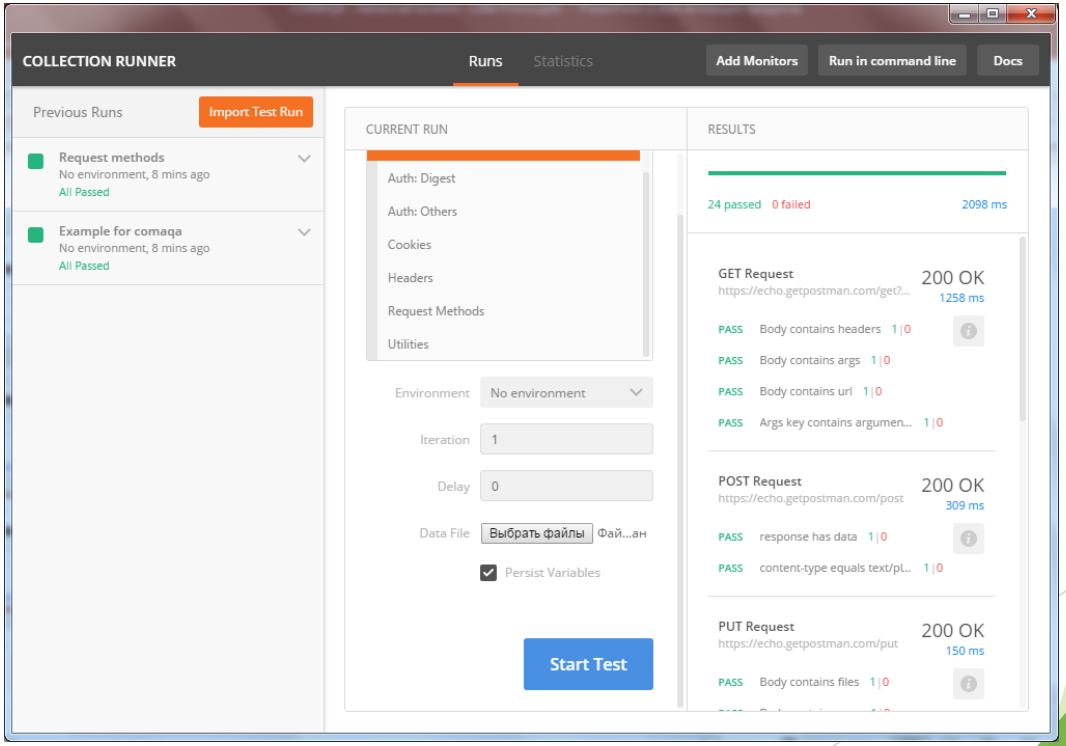
COMAQA.BY
QA automation community

POSTMAN



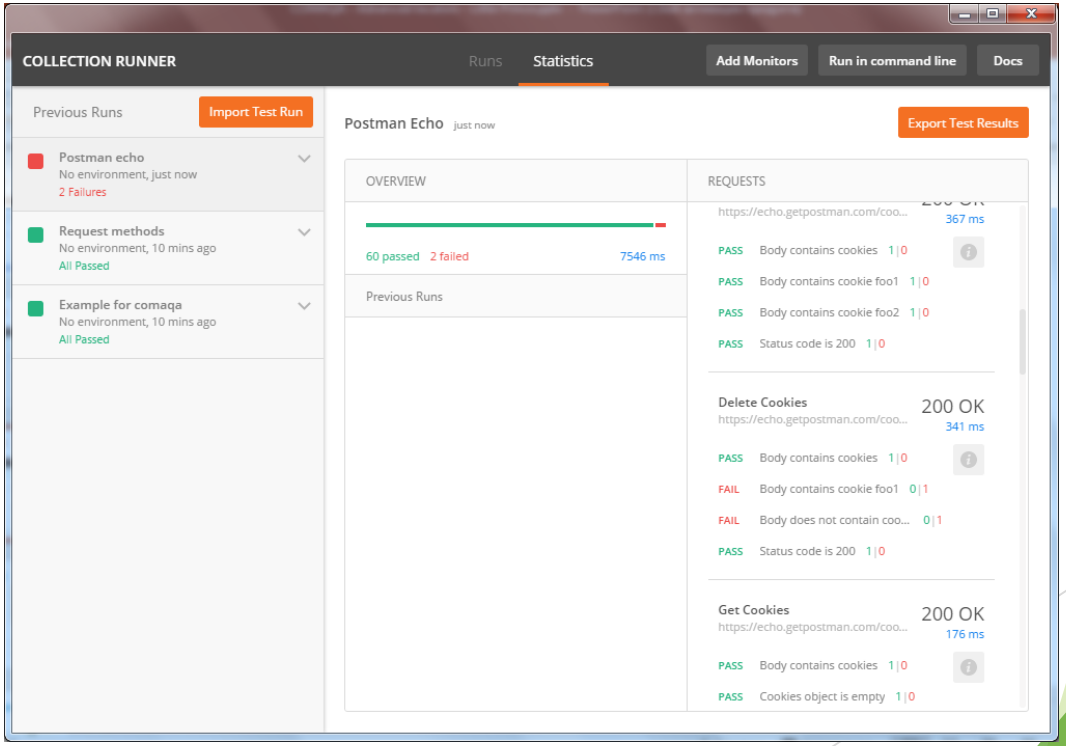
COMAQA.BY
QA automation community

POSTMAN



COMAQA.BY
QA automation community

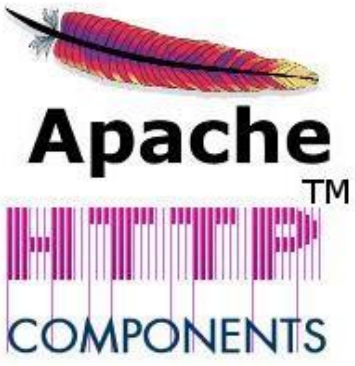
POSTMAN



TEST FRAMEWORK

COMAQA.BY
QA automation community

HTTP Client



XML/JSON parser

Test framework

COMAQA.BY
QA automation community

Lets take a look on tools that you could use in the framework. You will probably use some **HTTP client**. There are pretty big variety of clients. If you need you could use ***Apache HTTP client***. It will take, maybe more effort, to use it, and you will need to wright more code, but you will be able to fully control what's going on.
Another option is ***Spring RestTemplate*** to interact with web service. This is a bit different level of abstraction but under the hood you could still use Apache client..

COMAQA.BY
QA automation community

You will definitely need some **unit testing framework**. ***JUnit***, ***TestsNG***, it doesn't really matter. And you already know how to use it.

If you work with service that send data in **JSON** format, this is really nice library "***google-gson***". You could use this library to convert jsons into data transfer objects. Or create json from string or dto. If you use xml, you could use faster xml library.

To **compare JSON's** I suggest to use ***JSONAssert*** library. It could compare JSONs with different sorting order, or you could skip fields from comparison. As an example you create new user, and want to validate that web service returns exactly the same response that you need, but you can not predict some generated value in response, like user id. So you could skip particular value from comparison and check separately, that it has proper format or check it with regular expression.

If you are going to **mock** some parts of modules, you could use ***Mockito***. This is really nice tool.

So what also I have here. **Fiddler**. This is really not the part of the framework, but separate tool, that works as proxy, and it captures sessions and helps you to debug requests /responses during framework implementation. Also this is really useful to be able to save captured sessions. So if something works not as expected you could just save session and sent it to developers. This is really convenient. So I would say that fiddler is just must have if you work with backend.

Postman, it could be useful too, this is nice application with friendly UI where you could create request, execute it and check response.

And the last one that I would mention is ***REST Assured***. This is test automation framework, that has http client and a lot of syntax sugar, that allows you to create easy readable tests in Gherkin style.

COMAQA.BY
QA automation community

Main tools

- HTTP-client

spring by Pivotal

REST-assured

- Unit-Test*ing* Framework

JUnit 5

TestNG

- XML / JSON parser

google-gson
A Java library to convert JSON to Java objects and vice-versa

FasterXML, LLC
JSON and XML for the JVM.
Faster is better.

COMAQA.BY
QA automation community

Additional tools

- “Query” tool

```
curl https://api.github.com/zen
```

- Mapping tool

```
package models;

public class User {
    private String login;
    private String password;

    public User(String login, String password) {
        this.login = login;
        this.password = password;
    }

    public String getLogin() { return login; }

    public String getPassword() { return password; }
}
```

SPRING

COMAQA.BY
QA automation community

```java
import org.springframework.web.client.RestTemplate;
import org.testng.Assert;
import org.testng.annotations.Test;

public class SpringTest {
    private static final String USER_LOGIN = "USER32";
    private static final String SUCCESSFUL_RESPONSE = "Something here";

    private RestTemplate restTemplate = new RestTemplate();

    @Test
    public void postUserTest(){
        User user = new User();

        String response = restTemplate.postForObject("http://camaqa.by/users/", user, String.class);

        Assert.assertEquals(response, SUCCESSFUL_RESPONSE);
    }

    @Test
    public void getUserTest(){
        User user = restTemplate.getForObject("http://camaqa.by/users/32", User.class);

        Assert.assertEquals(user.getUserLogin(), USER_LOGIN);
    }

}
```

spring by Pivotal™

COMAQA.BY
QA automation community

Lets take a closer look. **Rest template**. To use it you even don't need to bring all Spring stuff into the framework just create RestTemplate() instance. And you are good to go.

You could specify in what type you want to map response body. **To send post request method postForObject could be used**. Just specify **URI**, add **payload** and select in what **type you want to receive the response**. You could map it to some DTO object and than check some fields.

SPRING

COMAQA.BY
QA automation community


```
public class ExchangeDemo {
    public static void main(String args[]) {
        RestTemplate restTemplate = new RestTemplate();

        String uri = "http://localhost:8080/HolloWorldApp/{id}";

        HttpHeaders headers = new HttpHeaders();
        headers.setContentType(MediaType.APPLICATION_JSON);

        HttpEntity<String> entity = new HttpEntity<String>( headers);

        ResponseEntity<Person> personEntity = restTemplate.
                    exchange(uri, HttpMethod.GET, entity, Person.class, 100);
    }
}
```

COMAQA.BY
QA automation community

Let make it a bit more complicated, you could use **HTTP entity to add headers, or you could pass some variables to uri**. Like in this example: I have **ID parameter** here, and rest template will **replace it with parameter from method call**. Here ID will be replaced with one hundred.

COMAQA.BY
QA automation community

spring
by Pivotal

Usage

```
RestTemplate restTemplate = new RestTemplate();
User user = restTemplate.getForObject( url: "http://comaqa.by/users", User.class);
```

COMAQA.BY
QA automation community

spring by Pivotal

Core

```
package core;

import org.springframework.web.client.RestTemplate;
import org.testng.annotations.BeforeTest;

public class TestBase {

    protected RestTemplate restTemplate;
    protected final String BASE_URL = "http://comaqa.by/";

    @BeforeTest
    public void setup() {
        restTemplate = new RestTemplate();
    }
}
```

COMAQA.BY
QA automation community

spring by Pivotal

Data Generators


```
package data;

import models.User;

public class UserGenerator {

    public static User generate() {
        return new User( login: "admin",  password: "admin");
    }
}
```

COMAQA.BY
QA automation community

spring by Pivotal

Tests


```java
public class UsersTest extends TestBase {

    private final String USERS_URL = BASE_URL + "users";

    @Test
    public void userResponseTest() {

        User user = restTemplate.getForObject(USERS_URL, User.class);

        assertTrue(user.getLogin().contains("admin"));

    }

    @Test
    public void postToUserTest() {
        User user = UserGenerator.generate();

        List<String> resp = new ArrayList<String>();
        resp.add("Response Type");

        String response = restTemplate.postForObject(USERS_URL, user, String.class);

        assertEquals( expected: "Something we expect here", response);
    }
}
```

TEST FRAMEWORK

COMAQA.BY
QA automation community

```
@Test
public void httpClientTest() throws IOException {
    CloseableHttpClient httpclient = HttpClients.createDefault();
    HttpGet httpGet = new HttpGet("http://targethost/homepage");
    CloseableHttpResponse response = httpclient.execute(httpGet);

    try {
        System.out.println response.getStatusLine().getStatusCode());

        HttpEntity entity1 = response.getEntity();

        System.out.println( EntityUtils.toString(entity1));

        EntityUtils.consume(entity1);
    } finally {
        response.close();
    }
```


```
public class ResponseInfo<T> {
    private int exitCode;
    private String responseBody;
    private long elapsedTime;
    private Exception exception;
    private T referenceObject;
```

COMAQA.BY
QA automation community

Working with **Apache HTTP** is pretty simple too. You should have **instance of client** itself, and create **instance of request**. Than you just **retrieve response and checks that the entity content is fully consumed and the content stream, if exists, is closed**.

It could be useful to have **response wrapper in this case that will have exit code, response body as String**, Response body as object, Exception if any occurred during request execution, and if you want to check that request processing time, you could have this parameter too.

TEST FRAMEWORK

COMAQA.BY
QA automation community

google-gson

A Java library to convert JSON to Java objects and vice-versa

POJO→JSON

```
Gson gson = new Gson();
User user = new User();

String jsonInString = gson.toJson(user);
```

JSON→POJO

```
String jsonString="{'user_login':'foo', 'user_password':'bar'}";
Gson gson = new Gson();

User userFromJson = gson.fromJson(jsonString, User.class);
```

Custom DTO or com.google.gson.JsonElement

```
public class User {

    @SerializedName("user_login")
    private String userLogin;

    @SerializedName("user_password")
    private String userPassword;

    public String getUserLogin() {
        return userLogin;
    }

    public void setUserLogin(String userLogin) {
        this.userLogin = userLogin;
    }

    public String getUserPassword() {
        return userPassword;
    }

    public void setUserPassword(String userPassword) {
        this.userPassword = userPassword;
    }
}
```

COMAQA.BY
QA automation community

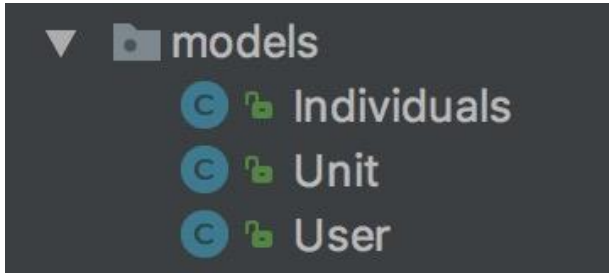
In most cases you will work with **JSONs** and **Google-GSON** is really good option here. There at least there approaches how to validate responses.
The first one is to convert JSON in to DTO, or POJO object. POJO is kind of object that doesn't have any logic, it contains only data, and getters with setters.
In current example you could see, that I expect field user_login with underscore. So I should specify which value should be mapped to user login parameter. In case if values are the same, as an example you have camel case in gson, so you even could get rid of this annotation, and google gson will automatically map json fields to proper values.
Converting objects from POJOS to JSON strings is really easy. I just need two methods to json and from json.

COMAQA.BY
QA automation community

And one more thing. If you don’t want to create or generate DTO classes, **you could map you class to JSON element**. **It will recursively go trout JSON and instantiate object that represents your JSON**. **So then it could be used with JSON assert library to compare it with another JSON**. This is much more convenient than compare them as Strings, and doesn’t require generation of POJO classes.

COMAQA.BY
QA automation community

Mapping




```
package models;

public class User {
    private String login;
    private String password;

    public User(String login, String password) {
        this.login = login;
        this.password = password;
    }

    public String getLogin() { return login; }

    public String getPassword() { return password; }
}
```

REST ASSURED

COMAQA.BY
QA automation community

REST-assured

```
import org.testng.annotations.Test;
import static io.restassured.RestAssured.*;
import static org.hamcrest.Matchers.*;

@Test
public void getUserTest() {
    Response response =
            given()
                    .auth().basic("login", "password")
            .when()
                    .get("http://camaqa.by/users/32")
            .then()
                    .statusCode(200)
                    .extract().response();
Assert.assertEquals(response.path("user_login"), USER_LOGIN);
Assert.assertEquals(response.path("user_password"), USER_PASSWORD);
}
```

COMAQA.BY
QA automation community

And I will finish this workshop with short description of rest assured framework. **It pretty easy to use and flexible enough**. **Rest Assured provide tons of syntax sugar, which helps to create simple and easy understandable tests**.

To use rest assured you should **statically import RestAssured and Matchers classes**. **To send get request you … just call "get" method and specify uri**.

From one side Rest assured mimics **Gherkin** syntax, with it's given when then constructions. From another it looks like **builder**, where all methods go one after another separated by comma.

In this example I set basic authentication, then send GET request and validate status code. Word when is syntax sugar, so I even can skip it in this case. Also rest assured provide **few strategies of request validation**.

In this example code sends GET request which returns response body, and then validate it.

REST ASSURED

COMAQA.BY
QA automation community

```
public class GivenWhenThen {
    @Test
    public void givenWhenThenExampleTest() {
        Response response =
        given().
                param("a", "b").
                header("x", "y").
                header("x1", "y1").
        when().
                get("http://postman-echo.com/get?test=123").
        then().
                statusCode(200).
                body("x.y", equalTo("z")).
        extract().
                response();
    }
}
```

with().

SETTING PARAMETERS

REQUEST EXECUTION

RESPONSE VALIDATIONS

RESPONSE OBJECT

Full **RestAssured test structure** looks like this. **Like in Cucumber RestAssured has Given When Then keywords**.
GIVEN section is responsible for configuring the request. *In this section we could specify headers, request body or any RestAssured parameters like proxy, ssl security staff and so on*.
WHEN section represents HTTP method and URI.
In THEN section goes all validation. *Also you could extract response body, and perform validation with test framework, instead validating in rest Assured.* I will describe each section in more detail.

REST ASSURED

COMAQA.BY
QA automation community

```
given().
        queryParam("QueryParamName", "QueryParamValue").
        pathParam("methodName", "post").
        cookie("customCookie", "customCookieValue").
        header("CustomHeader", "customHeaderValue").
        contentType("application/json").
        accept(ContentType.JSON).
        body(new User("User","Login")).
when().
        post("http://postman-echo.com/{methodName}").
```

Set query parameters
Set path parameters

Headers section

Add body

```
POST http://postman-echo.com/post?QueryParamName=QueryParamValue HTTP/1.1
Host: postman-echo.com
Cookie: customCookie=customCookieValue
CustomHeader: customHeaderValue
Accept: application/json, application/javascript, text/javascript, text/json
Content-Type: application/json; charset=UTF-8
Content-Length: 43

{"userLogin":"User","userPassword":"Login"}
```

COMAQA.BY
QA automation community

So, what about Given section.

In Given Section you can specify parameters for URI or query parameters, it makes tests more flexible, and code more reusable.

*Also, things like **Cookies** and **Headers** should be specified in given section.*

*Next one is **BODY**, where request body could be specified. This is most widely used methods from given section.*

In general, given method allows to set all needed request data, ant set framework parameters before request will be issued.

REST ASSURED. URI

COMAQA.BY
QA automation community

By default REST assured assumes host localhost and port 8080 when doing a request.

```
when().
        get().
then().
        statusCode(200);
```

```
GET http://localhost:8080/ HTTP/1.1
Accept: */*
Content-Length: 0
Host: localhost:8080
Connection: Keep-Alive
User-Agent: Apache-HttpClient/4.5.3
Accept-Encoding: gzip,deflate
```

You can change the default base URI, base path, port for all subsequent requests:

```
//In BaseTest
RestAssured.baseURI = "http://postman-echo.com";
RestAssured.basePath = "/get";

//In tests
given().
when().
        get("/resourceName").
then().
        statusCode(200);

RestAssured.reset();
```

```
GET http://postman-echo.com/get/resourceName HTTP/1.1
Accept: */*
Content-Length: 0
Host: postman-echo.com
Connection: Keep-Alive
User-Agent: Apache-HttpClient/4.5.3 (Java/1.8.0_131)
Accept-Encoding: gzip,deflate
```

COMAQA.BY
QA automation community

By default the RestAssured sends call to localhost if no parameters ware provided.
So in this example rest assured will send request to locallhost with port 8080, which is actually default tomcat port.

You can change default base uri, path and port by setting these values in base class, and then use in all tests, and you will have a possibility to change URI for all test from one place, which is good if you testing on different environments which have different URI.
The same with basePath.
In provided example, if I specify baseURI and base Path, the request will be sent to BaseURI + BasePath + UriParameter.

REST ASSURED. URI

COMAQA.BY
QA automation community

Path parameters makes it easier to read the request path as well as enabling the request path to easily be re-usable in many tests with different parameter values.

```
RestAssured.baseURI = "http://postman-echo.com";

given().
        pathParam("firstParameter", "bar1").
when().
        get("/get?foo1={firstParameter}&foo2={secondParameter}", "bar2").
then().
        statusCode(200);
```

Named parameters

Unnamed parameters

```
GET http://postman-echo.com/get?foo1=bar1&foo2=bar2 HTTP/1.1
Accept: */*
Content-Length: 0
Host: postman-echo.com
Connection: Keep-Alive
User-Agent: Apache-HttpClient/4.5.3 (Java/1.8.0_131)
Accept-Encoding: gzip,deflate
```

```
when().
        get("/get?foo1={firstParameter}&foo2={secondParameter}","foo1", "bar2").
```

Like String.format

COMAQA.BY
QA automation community

There few ways how to make URI value more flexible or configurable. You could use “**named parameters**”. So rest assured will check parameter name and replace it with corresponding value.

Or you could specify parameters directly in call method. They call it **unnamedParameters**.

You could mix this two types of parameters.

So In this example: framework will check that: ok I have in request specification this parameter, so I will replace it with specified value. Then it checks, ok I don’t have such parameter name in request specification, but I have one extra parameter in the request call, so I will replace it with this parameter.

And also parameters could be passed the same as in string format and RestAssured will replace them one by one.

REST ASSURED. HEADERS.

COMAQA.BY
QA automation community

```
given().
        header("MyHeader", "Something").
        header("MyOtherHeader", "SomethingElse").

        headers("MySecondHeader", "Something", "MyOtherSecondHeader", "SomethingElse").

        header("MyMultyValuedHeader", "Something", "SomethingElse").

        contentType(ContentType.JSON).

        accept("application/json").

when().
        get("/get").
then().
        statusCode(200);
```

Headers

Multi-value headers

Headers shortcuts

```
GET http://postman-echo.com/get HTTP/1.1
Host: postman-echo.com
Content-Type: application/json; charset=UTF-8
Accept: application/json
MyOtherSecondHeader: SomethingElse
MyMultyValuedHeader: Something
MyMultyValuedHeader: SomethingElse
MySecondHeader: Something
MyHeader: Something
MyOtherHeader: SomethingElse
```

Then you will probably need to set **headers and cookies**.

Method header will create single header if it was called wit**h two parameters** and **multiple headers** with the same name if you pass **three or more parameters**.

Method headers creates headers from pairs of parameters.
Also there are **shortcuts for frequently used headers**. Actually under the hood they are calling header() method. But well, these guys really want to make our code simple.

REST ASSURED. HEADERS.

COMAQA.BY
QA automation community

```
RestAssured.baseURI = "http://postman-echo.com";

Cookie someCookie = new Cookie.Builder("some_cookie", "some_value").
                            setExpiryDate(date).
                            setSecured(true).
                            setComment("some comment").build();

given().
        cookie(someCookie).

        cookie("username", "John").

        cookie("cookieName", "value1", "value2").

when().
        get("/get").
then().
        statusCode(200);
```

Detailed cookie builder

Just like headers

```
GET https://postman-echo.com/get HTTP/1.1
Cookie: some_cookie=some_value; username=John; cookieName=value1;
Accept: */*
Content-Length: 0
Host: postman-echo.com
Connection: Keep-Alive
User-Agent: Apache-HttpClient/4.5.3 (Java/1.8.0_131)
Accept-Encoding: gzip,deflate
```

COMAQA.BY
QA automation community

Pretty the **same idea with cookies**. You could set single cookie, or multi valued cookie. In this case rest assured will create **multiple cookies with the same name and different value**s.

Also if you need **more flexibility in cookies**, you could use **cookieBuilder**, which allows you **to set all cookie parameters**.

REST ASSURED. SPEC BUILDER.

COMAQA.BY
QA automation community

```
public RequestSpecification getCommonSpec() {
    RequestSpecBuilder builder = new RequestSpecBuilder();
    builder.addHeader("MyHeader", "Something");
    builder.addCookie("Key", "value");
    builder.setAccept(ContentType.ANY);
    RequestSpecification requestSpec = builder.build();
    return requestSpec;
}

@Test
public void specificationExample() {
    RestAssured.proxy(8888);
    RequestSpecification requestSpec =getCommonSpec();

    given().
            spec(requestSpec).
            param("parameter2", "paramValue").
    when().
            get("/something").
    then().
            body("x.y.z", equalTo("something"));
}
```

Instead duplicating response expectations and/or request parameters for different tests you can re-use an entire specification.

To do this you define a specification using either the RequestSpecBuilder or ResponseSpecBuilder.

COMAQA.BY
QA automation community

Setting the same values for multiple requests doesn't have any sense. To reduce code duplication, RestAssured framework has Request specification.

How it works: **you create request specification builder and add all parameters like headers and cookies. And this specification can be used in GIVEN section**.

But you **can add additional parameters in given section, or even override values from request specification**.

The same staff with Response specification. It could be used to apply some set of validations for all responses.

REST ASSURED. OBJECT MAPPING.

COMAQA.BY
QA automation community

Content-Type based Serialization

```
Message message = new Message();
message.setMessage("My messagee");
given().
       contentType("application/json").
       body(message).
when().
      post("/message");
```

```
public class Message {
    private String message;

    public String getMessage() {
        return message;
    }

    public void setMessage(String message) {
        this.message = message;
    }
}
```

Create JSON from a HashMap

You can also create a JSON document by supplying a Map to REST Assured.

```
Map<String, Object>  jsonAsMap = new HashMap<>();
jsonAsMap.put("firstName", "John");
jsonAsMap.put("lastName", "Doe");

given().
        contentType(JSON).
        body(jsonAsMap).
when().
        post("/somewhere").
```

1. JSON using Jackson 2 (Faster Jackson (databind))
2. JSON using Jackson (databind)
3. JSON using Gson
4. XML using JAXB

COMAQA.BY
QA automation community

And few words about **request/response objects mapping**.

In rest assured it is really easy. Lets assume that I have class Message with request data. So when I pass it in given section with body message, RestAssured **will serialize the object to JSON** since the request content-type is set to "**application/json**".

It will first try to use Jackson if found in classpath and if not Gson will be used. If you change the content-type to "**application/xml**" REST Assured will serialize to XML using JAXB. If no content-type is defined REST Assured will try to serialize in the following order.

You can also create a JSON document by **supplying a Map** to REST Assured.

REST ASSURED. RESPONSE VALIDATION.

COMAQA.BY
QA automation community

```
given().
        body(new User("User","Login")).
when().
        post("http://postman-echo.com/post").
then().
        statusCode(200).
        body("data.userLogin", equalTo("login")).
        body("data.userPassword", equalTo("password"));
```

```
Response response =
given().
        body(new User("User","Login")).
when().
        post("http://postman-echo.com/post").
then().
        statusCode(200).
        extract().response();

Assert.assertEquals(response.path("data.userLogin"), "login");
Assert.assertEquals(response.path("data.userPassword"), "password");
```

```
Message message = get("/message").as(Message.class);
```

COMAQA.BY
QA automation community

And then response validation. There are few ways. It could be validated with RestAssured:

StatusCode obviously validates status code, and **body method to validates response body**. First parameter is JSON path, and second is assert.

OR … **you could extract response object and validate with it**. Extract method will return Response object, which can be passed to TestNg asserts, and validated here.

And the **third way is map response to DTO object**, and work with it.

REST ASSURED. JSON SCHEMA.

COMAQA.BY
QA automation community

```
import static io.restassured.module.jsv.JsonSchemaValidator.*;
```

```
when().
        get("/products").
then().
        statusCode(200).
        assertThat().body(matchesJsonSchemaInClasspath("products-schema.json"));
```

```
<dependency>
    <groupId>io.rest-assured</groupId>
    <artifactId>json-schema-validator</artifactId>
    <version>3.0.5</version>
</dependency>
```

And even if you want to **validate with JSON schema**, it could be done in RestAssured. Just statically import **JsonSchemaValidatorClass**, and add new dependency in pom.xml and you are good to go.

REST ASSURED. PROXY. SSL. LOGGING.

```
RestAssured.proxy("localhost", 8888);
RestAssured.proxy(host("http://myhost.org").withAuth("username", "password"));

RestAssured.authentication = basic("username", "password");

given().
        auth().
        preemptive().basic("username", "password").
        relaxedHTTPSValidation().
        proxy(8888).
when().
        get("http://postman-echo.com/get?test=123").
then().
        body("args.foo2", equalTo("bar2")).
log().
        ifValidationFails();
```

```
given().log().all();
               .params();
               .body();
               .headers();
               .cookies();
               .method();
```

COMAQA.BY
QA automation community

At the end: few really important things that I haven't mentioned yet.

RestAssured has built in methods to perform authorization. If you will specify method basic in given section, RestAssured will try to authenticate with provided login/password. Preemptive method shows that RestAssured will try to authenticate even before it will be requested for authentication.

Also **if you want to see you requests in fiddler, and I assume you want, you should set proxy value**. By default Fiddler works on eighty-eight on eighty-eight port. Also, it uses self signed certificate, which will be recognized as unsafe. To deal with it. You could create custom key store and use it in your tests, or you could just turn off certificate validation when you work with fiddler.

Also **RestAssured has pretty good logging functionality**.

COMAQA.BY
QA automation community

spring
by Pivotal

Maven dependency

```
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-web</artifactId>
    <version>4.1.7.RELEASE</version>
</dependency>
```

COMAQA.BY
QA automation community

REST-assured

Dependency

```
<dependency>
    <groupId>com.jayway.restassured</groupId>
    <artifactId>rest-assured</artifactId>
    <version>2.9.0</version>
</dependency>
```

COMAQA.BY
QA automation community

REST-assured

Usage

```
@Test
public void userResponseTestAssured() {
    Response response =
    given()
            .authentication().basic( s: "admin",  s1: "admin").
    when()
            .get(USERS_URL).
    then()
            .statusCode(200)
            .extract().response();

    assertEquals(response.path( s: "login"),  actual: "admin");

}
```

COMAQA.BY
QA automation community

REST-assured

Usage

```
@Test
public void postUserTestAssured() {
    String user = UserGenerator.generateJsonString();

    given()
            .authentication().basic( s: "admin",  s1: "admin").
    body(user).
    when()
            .post(USERS_URL).
    then()
            .statusCode(200)
            .body(containsString( substring: "Success"));
}
```

COMAQA.BY
QA automation community

Rest Assured – please pay special attention to next features:

a. **Complex parsing and validation**

i. Groovy's collection API

ii. body("shopping.category.find { it.@type == 'groceries' }.item", hasItems("Chocolate", "Coffee"));

iii. from(response).getList("shopping.category.find { it.@type == 'groceries' }.item");

iv. get("/shopping").path("shopping.category.find { it.@type == 'groceries' }.item");

COMAQA.BY
QA automation community

b.Deserialization with Generics

i. List<Map<String, Object>>

ii. io.restassured.mapper.**TypeRef**

c. **Custom parsers**

i. RestAssured.**registerParser**(<content-type>, <parser>);

ii. RestAssured.**registerParser**("application/vnd.uoml+xml", Parser.XML);

iii. RestAssured.**unregisterParser**("application/vnd.uoml+xml");

COMAQA.BY
QA automation community

d. **Specification Re-use**
    i. ***RequestSpecBuilder***
    ii. ***ResponseSpecBuilder***
e. **Filters**
    i. io.restassured.filter.Filter
    ii. io.restassured.filter.log.RequestLoggingFilter
    iii. io.restassured.filter.log.ResponseLoggingFilter
    iv. io.restassured.filter.log.ErrorLoggingFilter
f. **Session support**

COMAQA.BY
QA automation community

g. **Spring Support**
    i. Spring Mock Mvc Module
    ii. Bootstrapping RestAssuredMockMvc
    iii. RestAssuredMockMvc
    iv. Asynchronous Requests
    v. Spring MVC Authentication
    iv. RestAssuredWebTestClient
h. Scala
i. Kotlin

Summary

COMAQA.BY
QA automation community

If you want to start automation – start from API.

- A lot of logic can be validated trough API without being dependent upon the UI.
- Web service is not tied to any one operating system or programming language, so why should the framework be written in the same language?
- API tests are **EASY TO AUTOMATE**, **CHEAP** and **FAST**.