

Web Automation:
Java + TestNG +
Selenium
Webdriver

REST API with Postman



Aleksei Barabash

Software Engineer

- Java
- Web And Mobile Automation
- Backend Automated Testing



Aleksei Barabash

Software Engineer

- Ruby, Webdriver, Cucumber, API
- Java, Appium, TestNG
- Java, Espresso, Jenkins, python, AWS



- Web Automation
- Selenium WebDriver
- Locating web elements
- Assertions
- Page Object - OOP architecture




- TestNG
- Data Providers
- Reporting
- Debugging



- REST API
- Postman

Rules and Recommendations

- Mute microphone 
- Use Questions/Chat section 
- Keep Calm! 
- Watch videos and code after class 
- No homework - no result
- Google it! 
- Use Skype 
- Use www.teamviewer.com 



Questions

First and last name

Country and city

Is working?

Company name

Skills

Is attending any courses now?

Why automation?



Why Java?

<https://fossbytes.com/most-popular-programming-languages/>



ROI for Test Automation

- Decreasing test regression time
- Reduction of repetitive testing
- Increasing efficiency of manual testers
- Faster releases
- Etc.



Risks and challenges

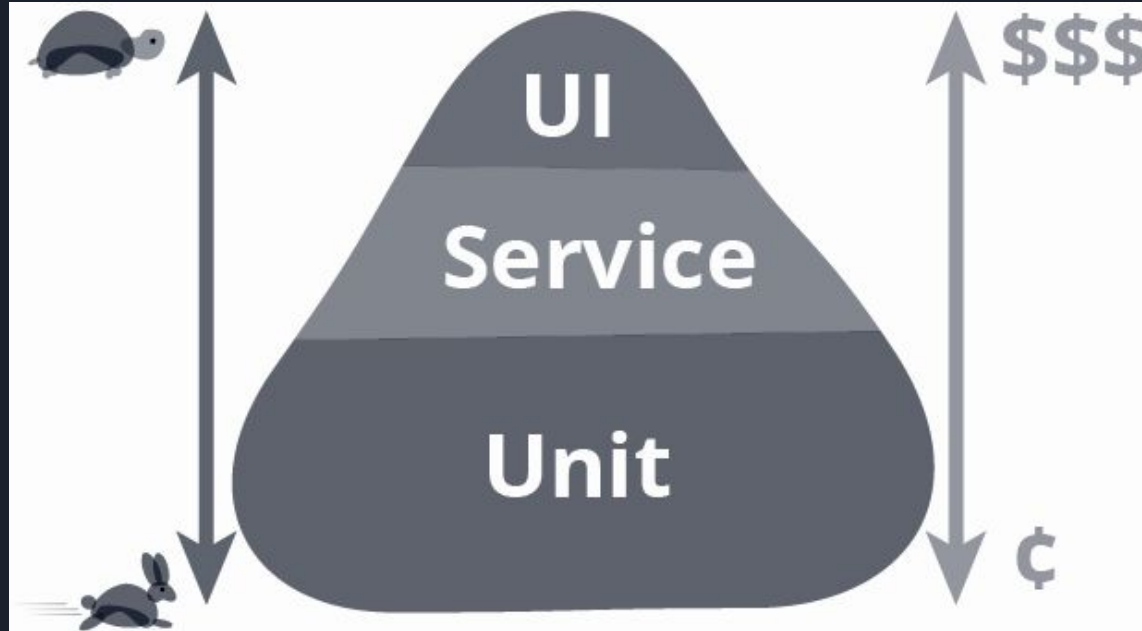
- Additional skills required
- Usually longer debugging
- Cost of maintenance
- Cost of infrastructure



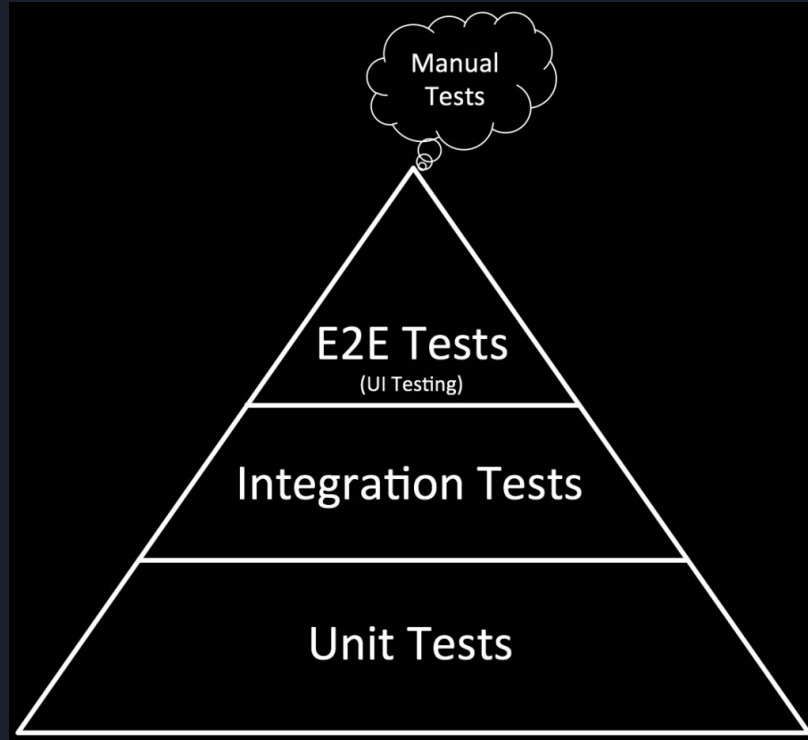
Manual vs. Automation

- Automated tests are not a substitute for manual testing
- Testing will remain as an “exploration exercise” which require domain knowledge and applying proper test techniques to be able to spot anomalies in the software
- Automated test is a set of predefined test steps and comparing the actual results with expected results - Automated checks

Automated tests Pyramid



<https://martinfowler.com/bliki/TestPyramid.html>





TestNG

Java

WebDriver
APIs



TestNG

- Annotations
- Test grouping
- Test parallelization
- Reports
- Exception handling
- Supported by CI/CD tools



WebDriver

- Supports all major browsers
- Supports multiple programming languages
- Works on Mac, Windows, Linux
- Local, remote and cloud



Test Stateless

- Start in clean state
- TestMethod independancy
- Finish in clean state



Postman

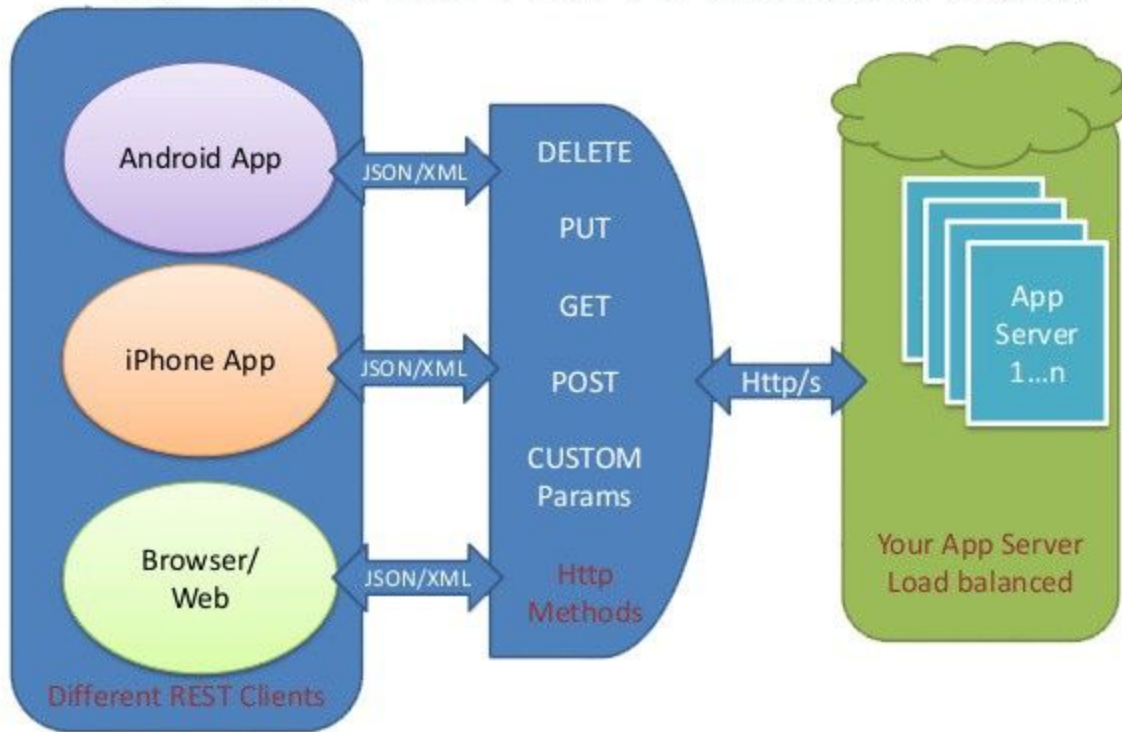
- Complete API development and testing environment
- Manual and Automated Testing
- Documentation
- Mocking
- Monitoring
- Workspaces, collections, tools, etc.
- Easy to use
- Great interface

API

Application Programming Interface



REST API Architecture





API - “window” to the code

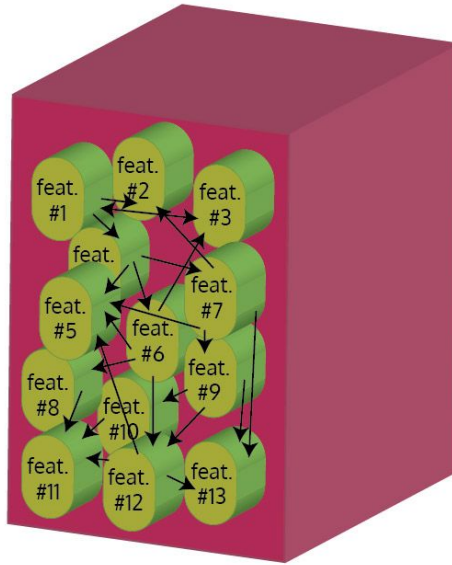
How the applications ‘talk’ to each other:

- System-level
- Network

Facebook, Google Maps, Yelp, Weather

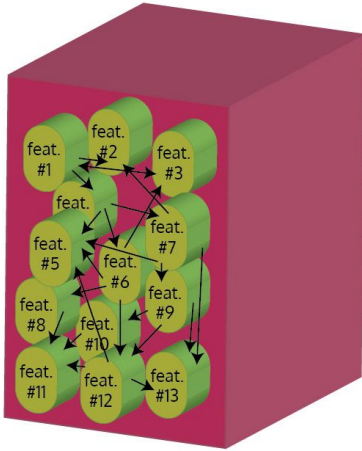
Monolith vs Services

Monolith

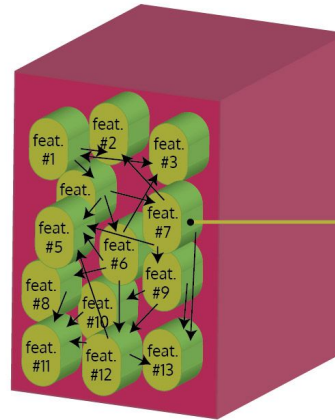


Monolith vs Services

Monolith

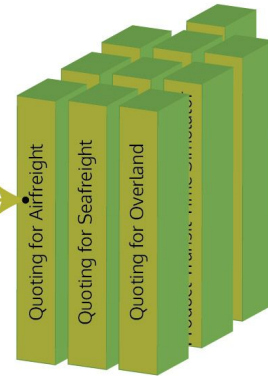


Monolith

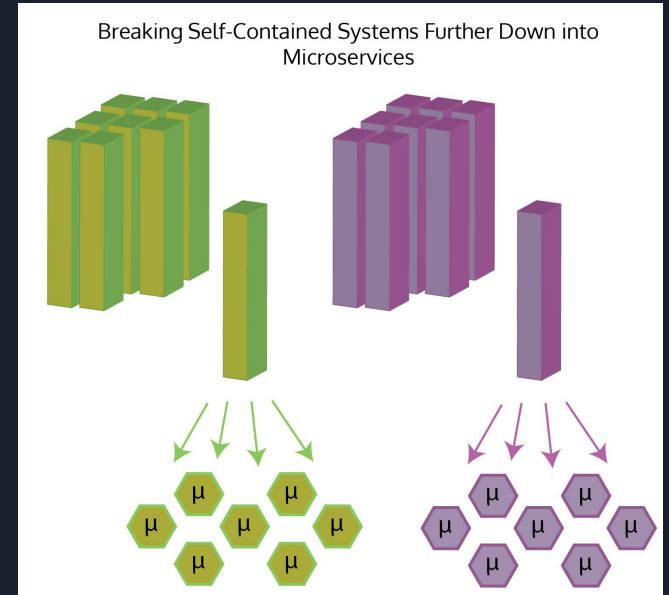
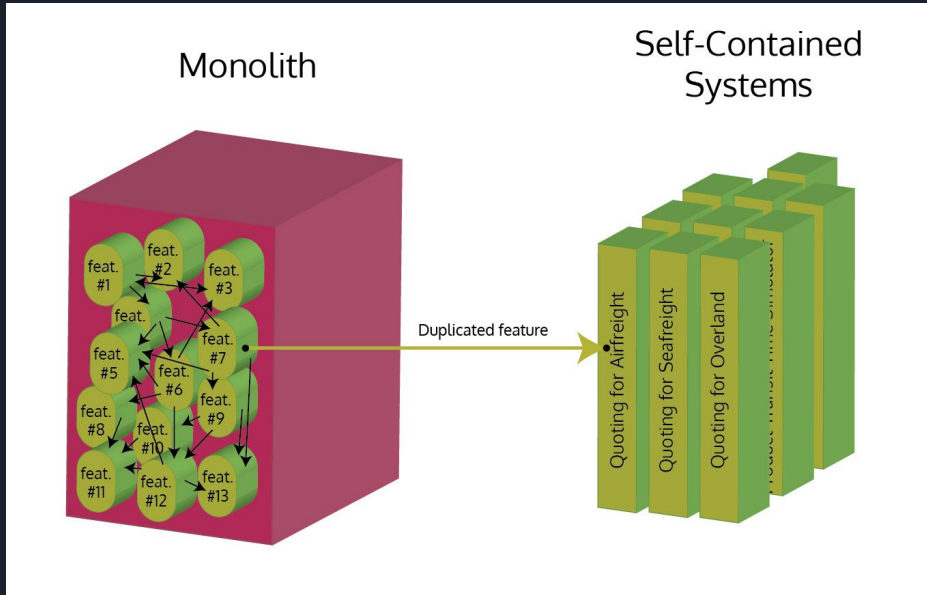


Duplicated feature

Self-Contained Systems

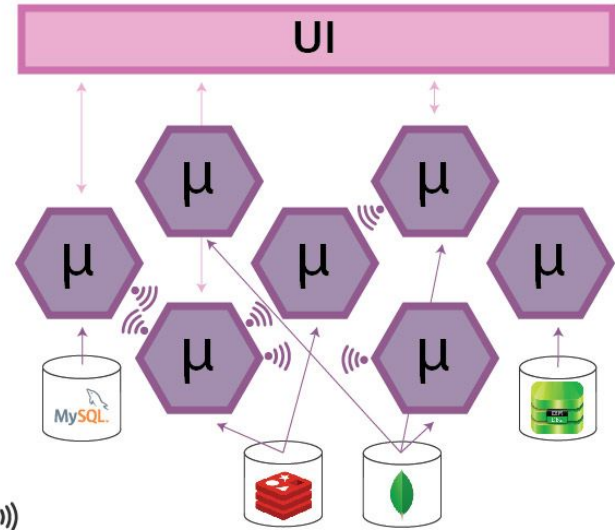
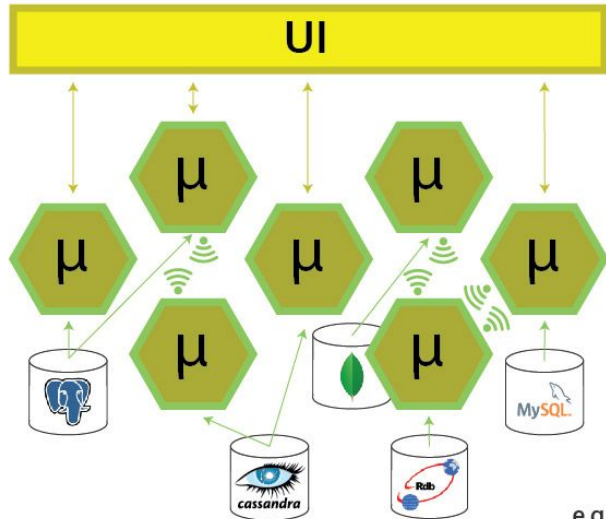


Monolith vs Services



Monolith vs Microservices

Microservices



e.g. API-based ☺))



References:

<https://dzone.com/articles/breaking-down-a-monolithic-software-a-case-for-mic>



API - “window” to the code

- Make internal function of the app public
- Only needed functionality exposed
- The way application interacts with outside world
- Standard “language” (java app talks with python app)

Analogy



Analogy





Facebook, Google Maps, Yelp, Weather

- <https://developers.facebook.com/docs/graph-api>
- <https://developers.google.com/apis-explorer/#p/>
- <https://www.yelp.com/developers>
- <https://openweathermap.org/api>



REST API

- Representational State Transfer
- Type of architecture of networked applications
- **Stateless** (server not saving any data, client is)
- Client-server
- HTTP/HTTPS
- Server object are resources (CRUD)
- Supports any programming language



Application vs Resource

- Application state is server-side data
 - identify incoming client requests
 - previous interaction details
 - current context information
- Resource state is current state of a resource on server
 - No information about communication
 - API response as resource representation

Stateless = free of application state



Advantages

- REST APIs are less complex
- No synchronization logic
- Easy to cache
- Improved performance
- Less connectivity problems



REST API Methods

- GET
- POST
- PUT
- DELETE



REST API Methods

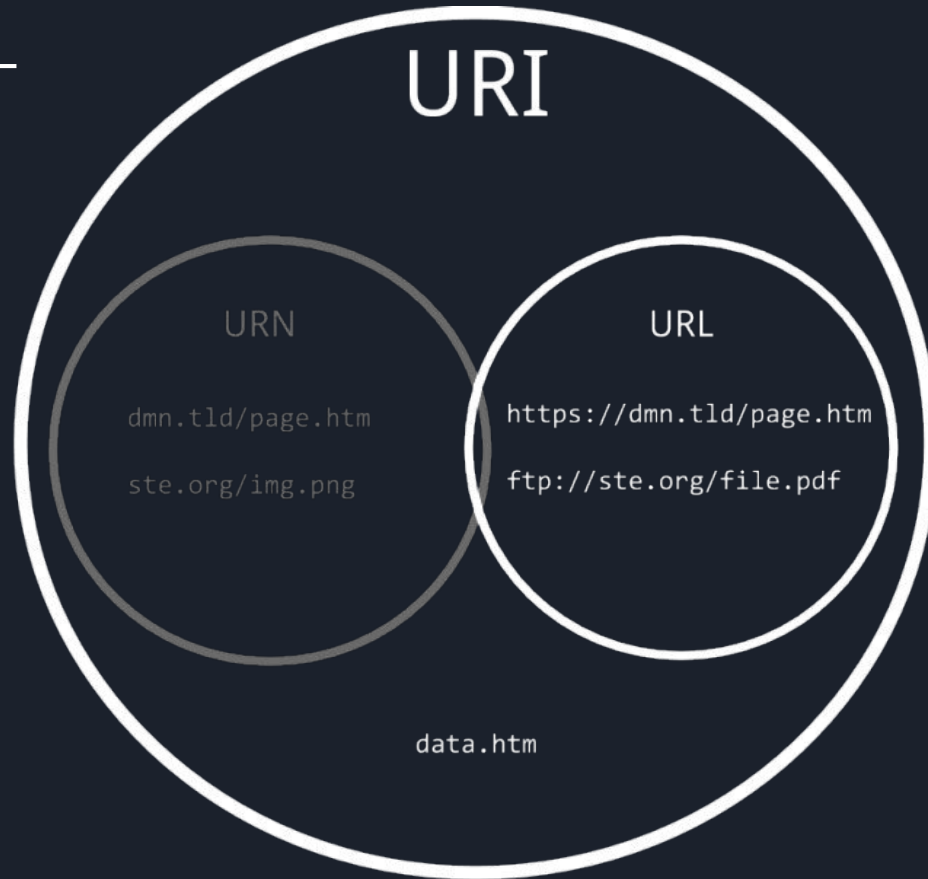
- GET
Retrieve data
- POST
Submit data
- PUT
Update data
- DELETE
Destroy data

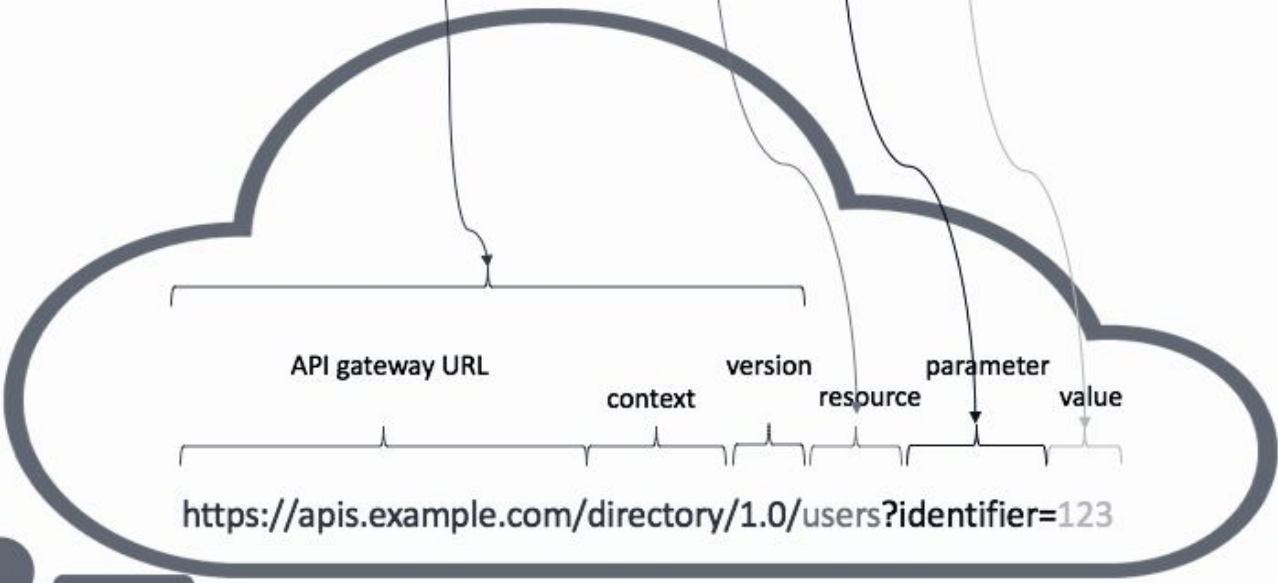
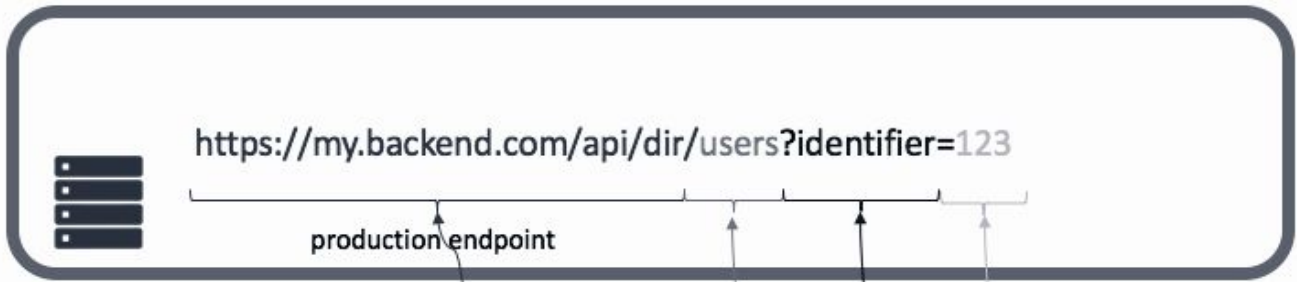


REST API Methods

- GET
Retrieve data
- POST
Submit data
- PUT
Update data
- DELETE
Destroy data
- HEAD
(no body)
- OPTIONS
Return methods
- PATCH
Update partial

URI vs. URL



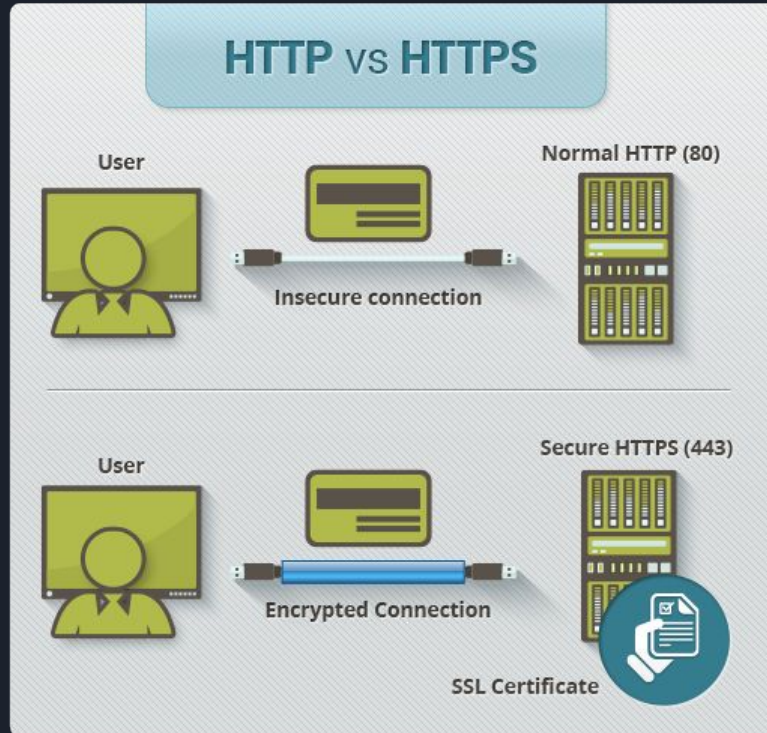




HTTP vs HTTPS

- The 'S' at the end of HTTPS stands for 'Secure'
- HTTPS using SSL (Secure Socket Layer) for sending and receiving information
- HTTPS using SSL certificate for encryption

HTTP vs HTTPS



HTTP vs HTTPS





Endpoints

GET <https://mysite.com/api/users>

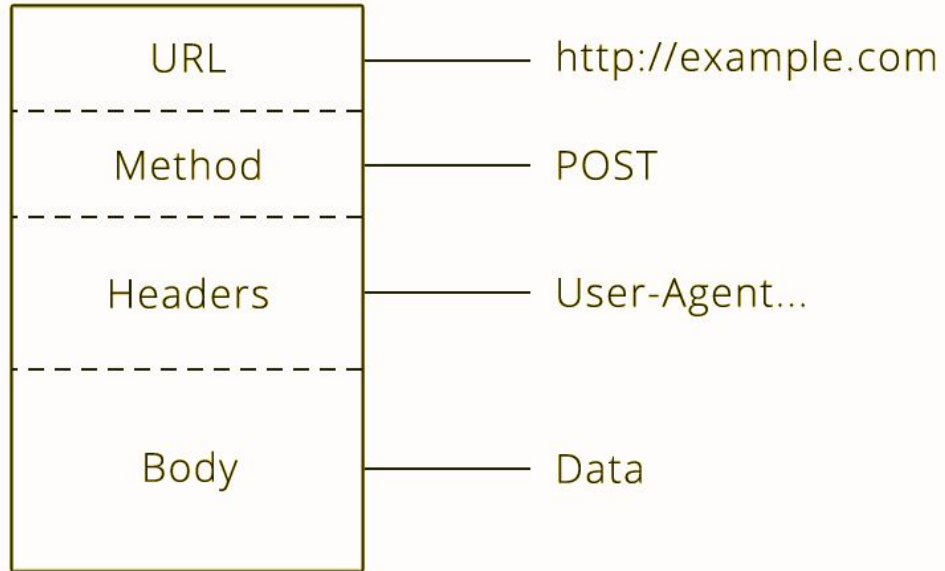
GET <https://mysite.com/api/user/1>

POST <https://mysite.com/api/users>

PUT <https://mysite.com/api/user/1>

DELETE <https://mysite.com/api/user/1>

Request

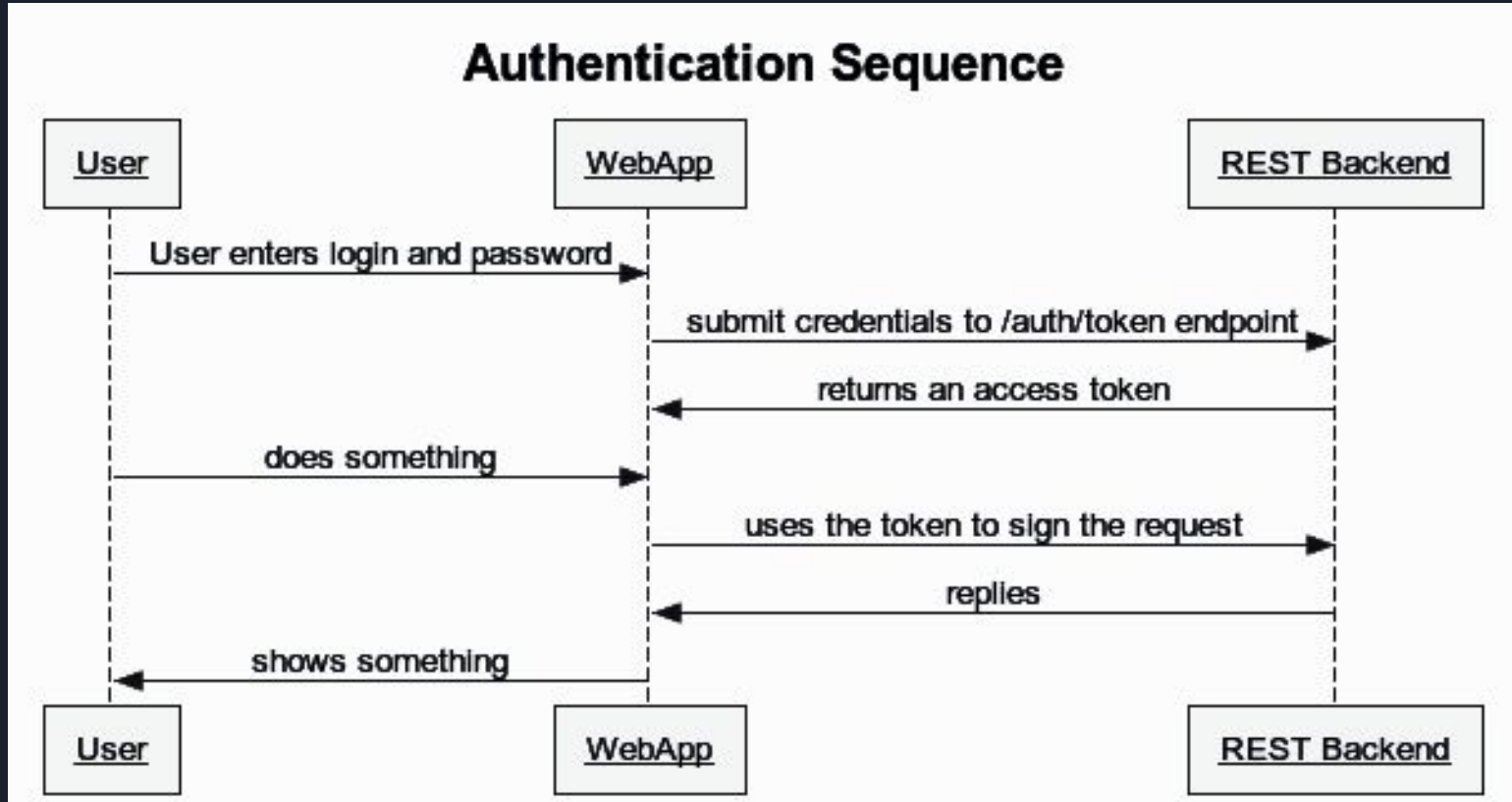


Request

Analogy



Authentication



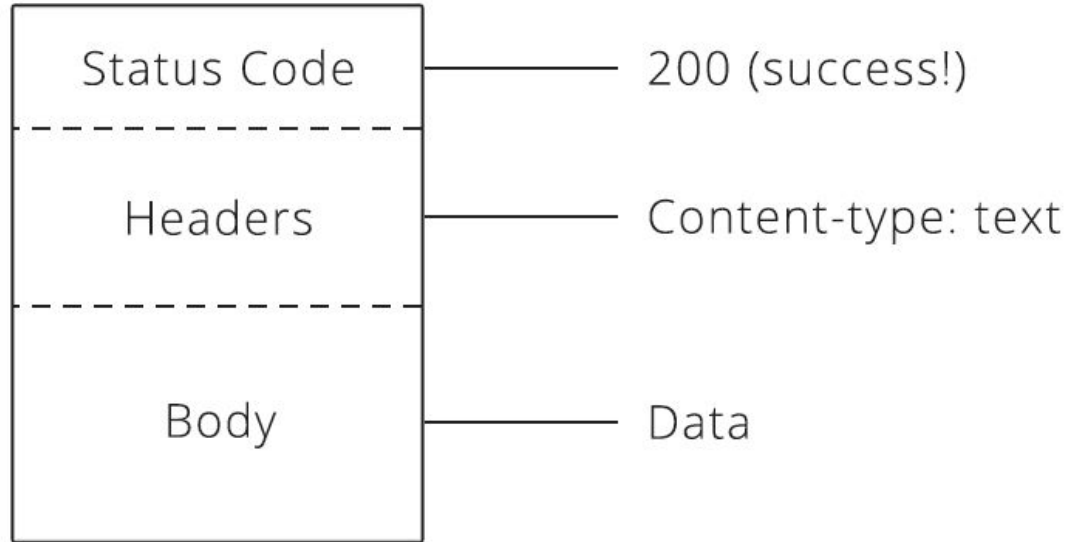


Example

```
curl -X GET https://api.github.com/users/username
```

<https://curl.haxx.se/>

Response



Response



JSON

JavaScript Object Notation

- lightweight data-interchange format
- Human-readable
- Easy to parse



JSON

JavaScript Object Notation

- Object
- Array
- Value



JSON

JavaScript Object Notation

- Object
 - unordered set of name/value pairs
{ name : value }
- Array
 - ordered collection of values
[value]
- Value
 - String
 - Number
 - Object
 - Array



Example

```
{
  "success":true,
  "data":[
    {
      "_id":"5b52d3ebac7afc0cbd1d3344",
      "name":"Coke (can)",
      "price":"1.00",
      "sectionId":"Soft Drinks",
      "available":true
    },
    {
      "_id":"5b52d771ac7afc0cbd1d3350",
      "name":"123",
      "price":"123",
      "sectionId":"0",
      "available":false
    }
  ]
}
```



JSON

JavaScript Object Notation

- Object
 - unordered set of name/value pairs
{ name : value }
- Array
 - ordered collection of values
[value]
- Value
 - Sting
 - Number
 - Object
 - Array
 - True
 - False
 - Null



JSON Syntax Rules

- Name and Value pairs: {"name" : "value"}
- Double quotes around key and value
- Use proper data type
- File type is ".json"
- Content type is "Application/json"

Analogy





Formatting and Validating

<https://jsonformatter.curiousconcept.com>

Create a JSON:

1. First and Last name
2. Country
3. City
4. Skills
5. Is Working? (true or false)
6. Company name

JSON vs. XML

```
<?xml version="1.0" encoding="UTF-8" ?>
<ProductInformation ExportTime="2010-11-23 23:56:40" ExportContext="german" ContextID="german" WorkspaceID="Test" id="1" parent="0">
- <AnalyzerResult>
  - <AnalyzerLists>
    - <AnalyzerList name="items.added">
      <AnalyzerElement ItemID="product?id=123456" ProductID="123456" />
      <AnalyzerElement ItemID="product?id=789" ProductID="789" />
    </AnalyzerList>
    - <AnalyzerList name="items.deleted">
      <AnalyzerElement ItemID="product?id=111111" ProductID="111111" />
      <AnalyzerElement ItemID="product?id=222222" ProductID="222222" />
    </AnalyzerList>
    - <AnalyzerList name="items.dummy_test">
      <AnalyzerElement ItemID="product?id=test1" ProductID="test1" />
      <AnalyzerElement ItemID="product?id=test2" ProductID="test2" />
    </AnalyzerList>
  </AnalyzerLists>
  - <AnalyzerDummyTest>
    <AnalyzerDummyTest name="Dummy not processed" />
  </AnalyzerDummyTest>
</AnalyzerResult>
- <Products>
  - <Product id="123456" name="Product A">
    - <MetaData>
      <Value AttributeID="AttrA">false</Value>
      <Value AttributeID="AttrB">>true</Value>
      <Value AttributeID="AttrShortName">Product A Short Name</Value>
      <Value AttributeID="AttrLongName">Product A Long Name</Value>
    </MetaData>
  </Product>
  - <Product id="789" name="Product B">
    - <MetaData>
      <Value AttributeID="AttrA">>true</Value>
      <Value AttributeID="AttrB">>false</Value>
      <Value AttributeID="AttrShortName">Product B Short Name</Value>
      <Value AttributeID="AttrLongName">Product B Long Name</Value>
    </MetaData>
  </Product>
</Products>
</ProductInformation>
```

JSON vs. XML

XML:

- Less human-readable
 - Redundant information
 - Element's attributes
-
- SQL Server:
 - Less server disk space
 - Slower performance

```
<?xml version="1.0" encoding="UTF-8" ?>
<ProductInformation ExportTime="2010-11-23 23:56:40" ExportContext="german" ContextID="german" WorkspaceID="Test" id="1" parent="0">
- <AnalyzerResult>
  - <AnalyzerLists>
    - <AnalyzerList name="items.added">
      <AnalyzerElement ItemID="product?id=123456" ProductID="123456" />
      <AnalyzerElement ItemID="product?id=789" ProductID="789" />
    </AnalyzerList>
    - <AnalyzerList name="items.deleted">
      <AnalyzerElement ItemID="product?id=111111" ProductID="111111" />
      <AnalyzerElement ItemID="product?id=222222" ProductID="222222" />
    </AnalyzerList>
    - <AnalyzerList name="items.dummy_test">
      <AnalyzerElement ItemID="product?id=test1" ProductID="test1" />
      <AnalyzerElement ItemID="product?id=test2" ProductID="test2" />
    </AnalyzerList>
  </AnalyzerLists>
  - <AnalyzerDummyTest>
    <AnalyzerDummyTest name="Dummy not processed" />
  </AnalyzerDummyTest>
</AnalyzerResult>
- <Products>
  - <Product id="123456" name="Product A">
    - <MetaData>
      <Value AttributeID="AttrA">false</Value>
      <Value AttributeID="AttrB">true</Value>
      <Value AttributeID="AttrShortName">Product A Short Name</Value>
      <Value AttributeID="AttrLongName">Product A Long Name</Value>
    </MetaData>
  </Product>
  - <Product id="789" name="Product B">
    - <MetaData>
      <Value AttributeID="AttrA">true</Value>
      <Value AttributeID="AttrB">false</Value>
      <Value AttributeID="AttrShortName">Product B Short Name</Value>
      <Value AttributeID="AttrLongName">Product B Long Name</Value>
    </MetaData>
  </Product>
</Products>
</ProductInformation>
```



XML formatter

<https://www.freeformatter.com/xml-formatter.html>

Create an XML:

1. First and Last name
2. Hobby
3. Favorite movie
4. Favorite musician/band



Explain What Is A “Resource” In REST?

REST architecture treats every content as a resource. These resources can be either text files, HTML pages, images, videos or dynamic business data.

REST Server provides access to resources and REST client accesses and modifies these resources. Here each resource is identified by URIs/ global IDs.



What Is The Most Popular Way To Represent A Resource In REST?

REST uses different representations to define a resource like text, JSON, and XML.

XML and JSON are the most popular representations of resources.



Which Protocol Is Used By RESTful Web Services?

RESTful web services make use of HTTP protocol as a medium of communication between client and server.



What Is Messaging In RESTful Web Services?

RESTful web services make use of HTTP protocol as a medium of communication between client and server. The client sends a message in the form of an HTTP Request.

In response, the server transmits the HTTP Response. This technique is called Messaging. These messages contain message data and metadata i.e. information about the message itself.



State The Core Components Of An HTTP Request?

1. The Verb which indicates HTTP methods such as GET, PUT, POST, DELETE.
2. URI stands for Uniform Resource Identifier (URI). It is the identifier for the resource on the server.
3. HTTP Version which indicates HTTP version, for example-HTTP v1.1.
4. Request Header carries metadata (as key-value pairs) for the HTTP Request message. Metadata could be a client (or browser) type, the format that client supports, message body format, and cache settings.
5. Request Body indicates the message content or resource representation.



State The Core Components Of An HTTP Response?

1. Status/Response Code – Indicates Server status for the resource present in the HTTP request. For example, 404 means resource not found and 200 means response is ok.
2. HTTP Version – Indicates HTTP version, for example-HTTP v1.1.
3. Response Header – Contains metadata for the HTTP response message stored in the form of key-value pairs. For example, content length, content type, response date, and server type.
4. Response Body – Indicates response message content or resource representation.



Name The Most Commonly Used HTTP Methods Supported By REST?

1. GET - It requests a resource at the request-URL. It should not contain a request body as it will get discarded. Maybe it can be cached locally or on the server.
2. POST – It submits information to the service for processing; it should typically return the modified or new resource.
3. PUT – At the request URL it updates the resource.
4. DELETE – It removes the resource at the request-URL.
5. OPTIONS -It indicates the supported techniques.
6. HEAD – It returns meta information about the request URL.



Mention some key characteristics of REST?

Some key characteristics of REST includes

REST is stateless, therefore the SERVER has no state (or session data)

With a well-applied REST API, the server could be restarted between two calls as every data is passed to the server

Web service mostly uses POST method to make operations, whereas REST uses GET to access resources