KINGSLAND UNIVERSITY

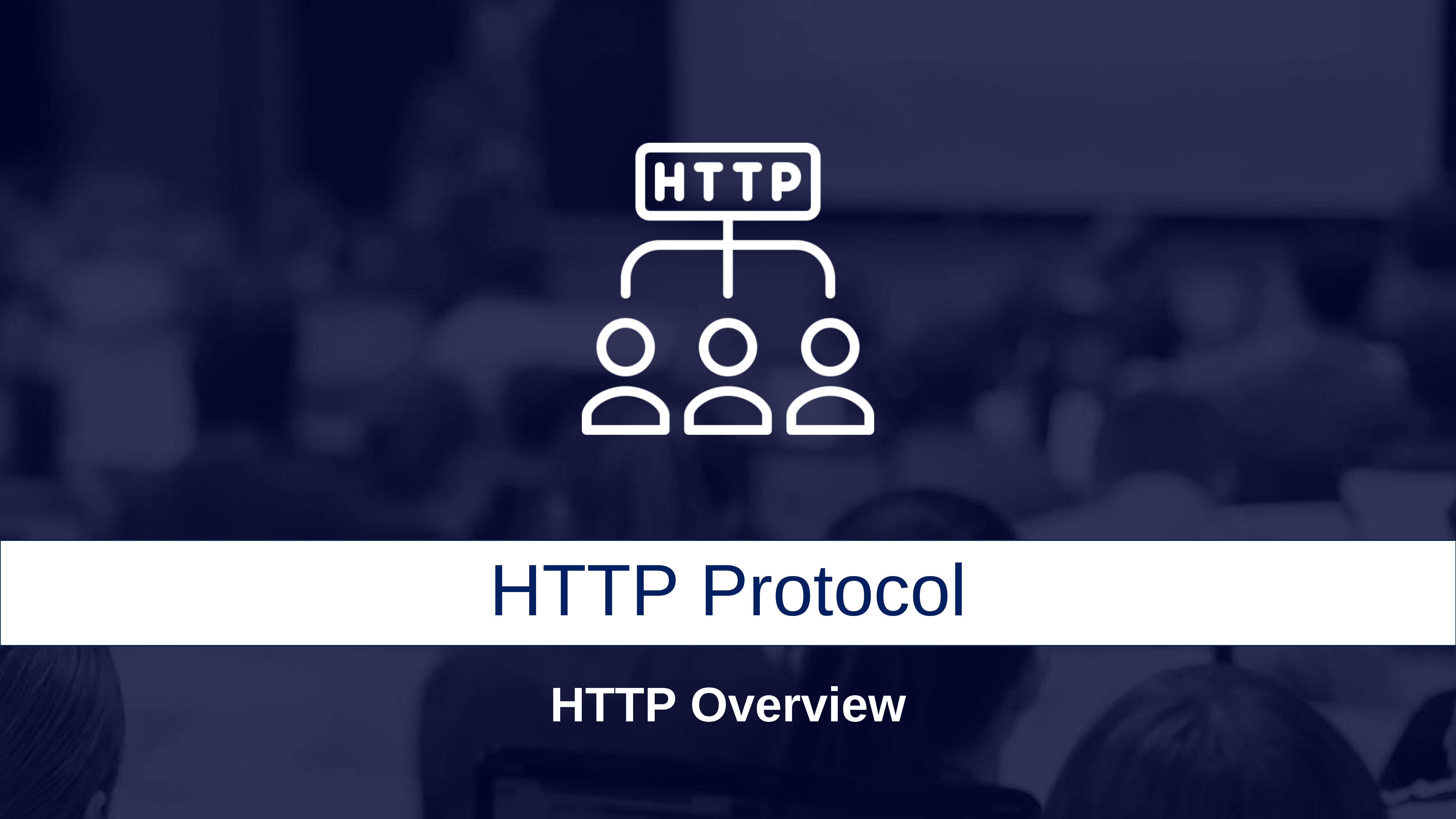
REST Services and AJAX

HTTP, RESTful Web Services, AJAX Concepts, XMLHttpRequest

Table of Contents

Have a Question?

#js-advanced



HTTP Protocol

HTTP Overview

HTTP Basics

- HTTP (**H**yper **T**ext **T**ransfer **P**rotocol)
  - Text-based client-server protocol for the Internet
  - For transferring Web resources (HTML files, images, styles, etc.)
  - Request-response based

Web Client

HTTP request

HTTP response

Web Server

HTTP Request Methods

- **HTTP** defines **methods** to indicate the desired action to be performed on the identified resource

Method	Description
GET	Retrieve / load a resource
POST	Create / store a resource
PUT	Update a resource
DELETE	Delete (remove) a resource
PATCH	Update resource partially
HEAD	Retrieve the resource's headers
OPTIONS	Returns the HTTP methods that the server supports for the specified URL

HTTP GET Request - Example

```
GET /users/testnakov/repos HTTP/1.1
Host: api.github.com
Accept: */*
Accept-Language: en
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/54.0.2840.71
Safari/537.36
Connection: Keep-Alive
Cache-Control: no-cache
<CRLF>
```

HTTP request line

HTTP headers

The request body is empty

HTTP POST Request – Example

```
POST /repos/testnakov/test-nakov-repo/issues HTTP/1.1
Host: api.github.com
Accept: */*
Accept-Language: en
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible;MSIE 6.0; Windows NT 5.0)
Connection: Keep-Alive
Cache-Control: no-cache
<CRLF>
{"title":"Found a bug",
 "body":"I'm having a problem with this.",
 "labels":["bug","minor"]}
<CRLF>
```

HTTP request line

HTTP headers

The request body holds the submitted data

HTTP Response – Example

```
HTTP/1.1 200 OK
Date: Fri, 11 Nov 2016 16:09:18 GMT+2
Server: Apache/2.2.14 (Linux)
Accept-Ranges: bytes
Content-Length: 84
Content-Type: text/html
<CRLF>
<html>
  <head><title>Test</title></head>
  <body>Test HTML page.</body>
</html>
```

HTTP response status line

HTTP response headers

HTTP response body

HTTP Response Status Codes

Status Code	Action	Description
200	OK	Successfully retrieved resource
201	Created	A new resource was created
204	No Content	Request has nothing to return
301 / 302	Moved	Moved to another location (redirect)
400	Bad Request	Invalid request / syntax error
401 / 403	Unauthorized	Authentication failed / Access denied
404	Not Found	Invalid resource
409	Conflict	Conflict was detected, e.g. duplicated email
500 / 503	Server Error	Internal server error / Service unavailable

Content-Type and Disposition

- The **Content-Type** / **Content-Disposition** headers specify how the HTTP request / response body should be processed

JSON-encoded data

```
Content-Type: application/json
```

UTF-8 encoded HTML page. Will be shown in the browser

```
Content-Type: text/html; charset=utf-8
```

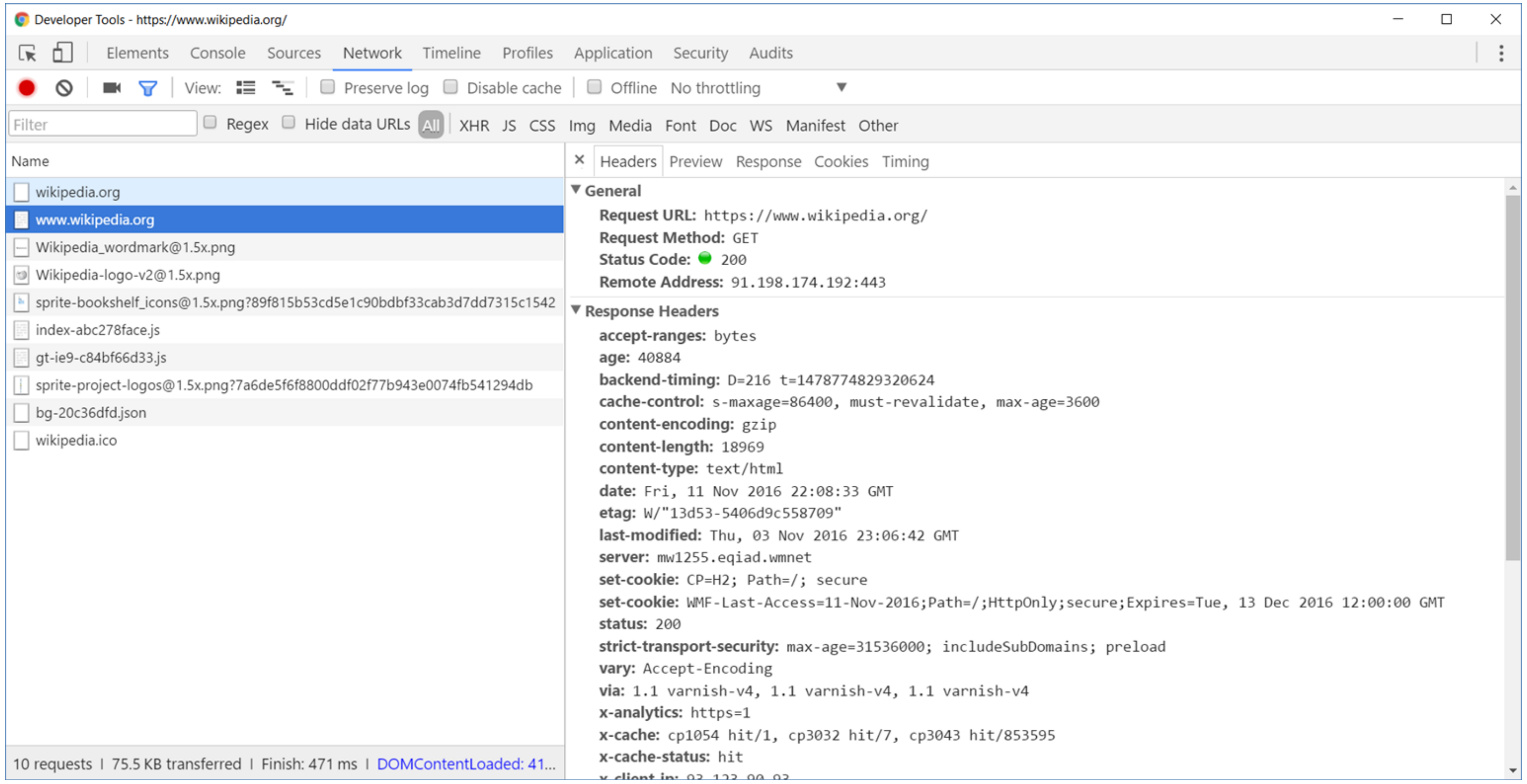
```
Content-Type: application/pdf
Content-Disposition: attachment;
filename="Financial-Report-April-2016.pdf"
```

This will download a PDF file named Financial-Report-April-2016.pdf

Chrome Dev Tools, Fiddler, Postman

HTTP Developer Tools

Chrome Developer Tools



Developer Tools - https://www.wikipedia.org/
Elements Console Sources Network Timeline Profiles Application Security Audits
View:
Preserve log
Disable cache
Offline
No throttling
Filter
Regex
Hide data URLs
All
XHR JS CSS Img Media Font Doc WS Manifest Other
Headers Preview Response Cookies Timing

Name

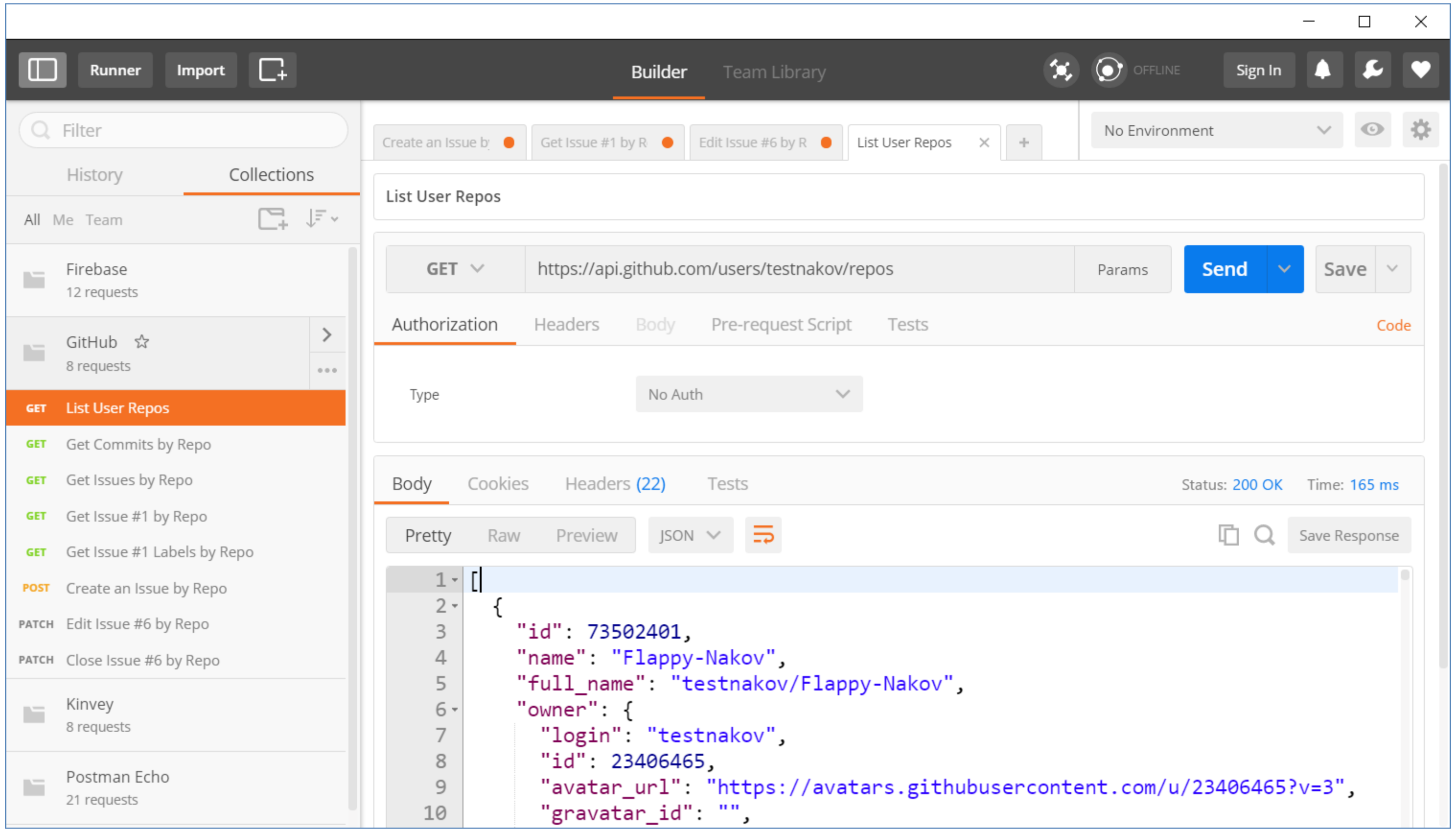
- wikipedia.org
- www.wikipedia.org
- Wikipedia_wordmark@1.5x.png
- Wikipedia-logo-v2@1.5x.png
- sprite-bookshelf_icons@1.5x.png?89f815b53cd5e1c90bdbf33cab3d7dd7315c1542
- index-abc278face.js
- gt-ie9-c84bf66d33.js
- sprite-project-logos@1.5x.png?7a6de5f6f8800ddf02f77b943e0074fb541294db
- bg-20c36dfd.json
- wikipedia.ico

10 requests | 75.5 KB transferred | Finish: 471 ms | DOMContentLoaded: 41...

```
▼ General
   Request URL: https://www.wikipedia.org/
   Request Method: GET
   Status Code: ● 200
   Remote Address: 91.198.174.192:443
▼ Response Headers
   accept-ranges: bytes
   age: 40884
   backend-timing: D=216 t=1478774829320624
   cache-control: s-maxage=86400, must-revalidate, max-age=3600
   content-encoding: gzip
   content-length: 18969
   content-type: text/html
   date: Fri, 11 Nov 2016 22:08:33 GMT
   etag: W/"13d53-5406d9c558709"
   last-modified: Thu, 03 Nov 2016 23:06:42 GMT
   server: mw1255.eqiad.wmnet
   set-cookie: CP=H2; Path=/; secure
   set-cookie: WMF-Last-Access=11-Nov-2016;Path=/;HttpOnly;secure;Expires=Tue, 13 Dec 2016 12:00:00 GMT
   status: 200
   strict-transport-security: max-age=31536000; includeSubDomains; preload
   vary: Accept-Encoding
   via: 1.1 varnish-v4, 1.1 varnish-v4, 1.1 varnish-v4
   x-analytics: https=1
   x-cache: cp1054 hit/1, cp3032 hit/7, cp3043 hit/853595
   x-cache-status: hit
   x-client-ip: 93.123.90.93
```

Read more about Chrome Developer Tools

Postman



List User Repos

GET https://api.github.com/users/testnakov/repos

Type: No Auth

Status: 200 OK Time: 165 ms

```
[
  {
    "id": 73502401,
    "name": "Flappy-Nakov",
    "full_name": "testnakov/Flappy-Nakov",
    "owner": {
      "login": "testnakov",
      "id": 23406465,
      "avatar_url": "https://avatars.githubusercontent.com/u/23406465?v=3",
      "gravatar_id": "",
```

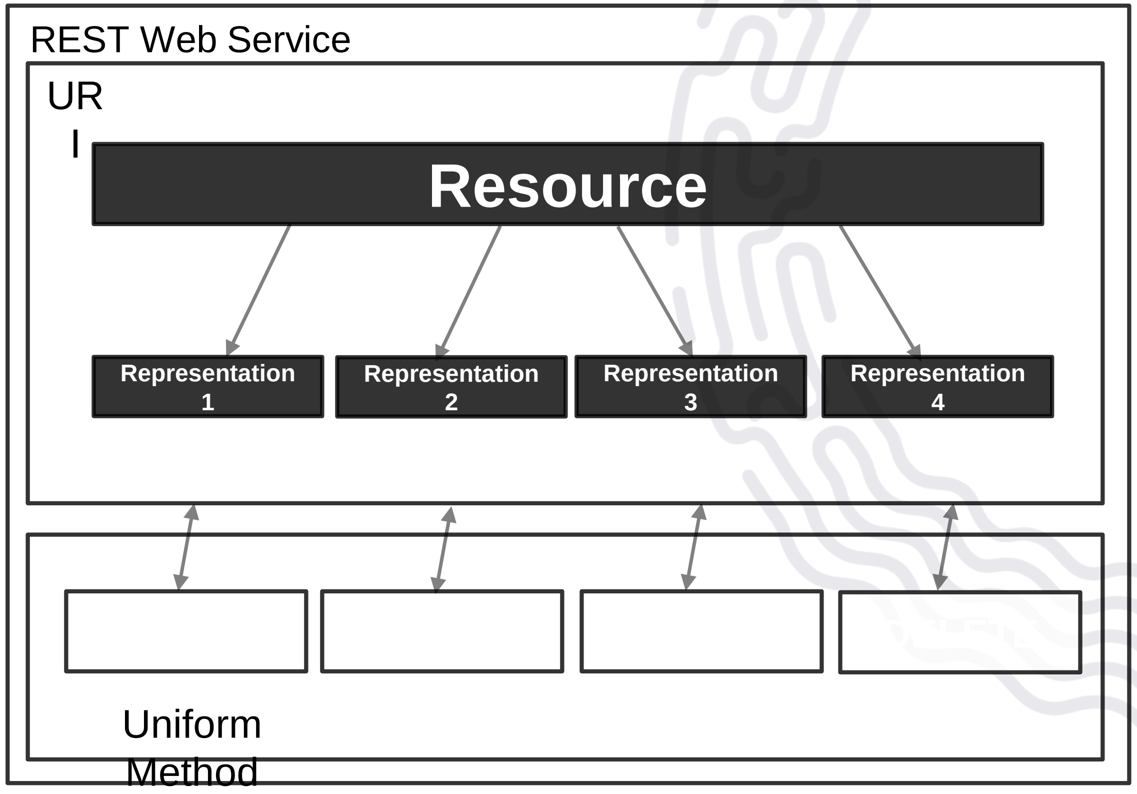
Read more about Postman REST Client

{REST}

REST and RESTful Services

REST and RESTful Services

- **Re**presentational **S**tate **T**ransfer (**REST**)
  - Architecture for **client-server communication** over HTTP
  - Resources have **URI** (address)
  - Can be **created/retrieved/ modified/deleted**/etc.
- RESTful API/RESTful Service
  - Provides access to **server-side resources** via **HTTP** and **REST**



REST Architectural Constraints

- REST defines **6 architectural constraints** which make any web service - a true RESTful API
  - Client-server architecture
  - Statelessness
  - Cacheable
  - Layered system
  - Code on demand (optional)
  - Uniform interface

Read more about REST Architectural Constraints

REST and RESTful Services – Example

- Create a new post

POST	http://some-service.org/api/posts

- Get all posts / specific post

GET	http://some-service.org/api/posts

GET	http://some-service.org/api/posts/17

- Delete existing post

DELETE	http://some-service.org/api/posts/17

- Replace / modify existing post

PUT/PATCH	http://some-service.org/api/posts/17

Accessing GitHub Through HTTP

GitHub REST API

GitHub API

- List user's all public repositories:

GET	https://api.github.com/users/testnakov/repos

- Get all commits from a public repository:

GET	https://api.github.com/repos/testnakov/softuniada-2016/commits

- Get all issues/issue #1 from a public repository

GET	/repos/testnakov/test-nakov-repo/issues

GET	/repos/testnakov/test-nakov-repo/issues/1

Github: Labels Issue

- Get the first issue from the "**test-nakov-repo**" repository
- Send a **GET** request to:
  - **https://api.github.com/repos/testnakov/test-nakov-repo/issues/:id**
  - Where **:id** is the current issue

GitHub API (2)

- Get all labels for certain issue from a public repository:

GET	https://api.github.com/repos/testnakov/test-nakov-repo/issues/1/labels

- Create a new issue to certain repository (with authentication)

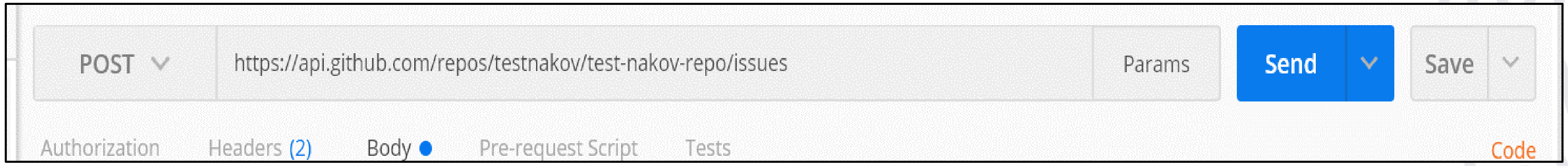
POST	https://api.github.com/repos/testnakov/test-nakov-repo/issues

Headers	Authorization: Basic base64(user:pass)

Body	{"title":"Found a bug", "body": "I'm having a problem with this."}

Github: Create Issue

- Create an issue when you send a "**POST**" request
- Use your Github account **credentials** to submit the issue



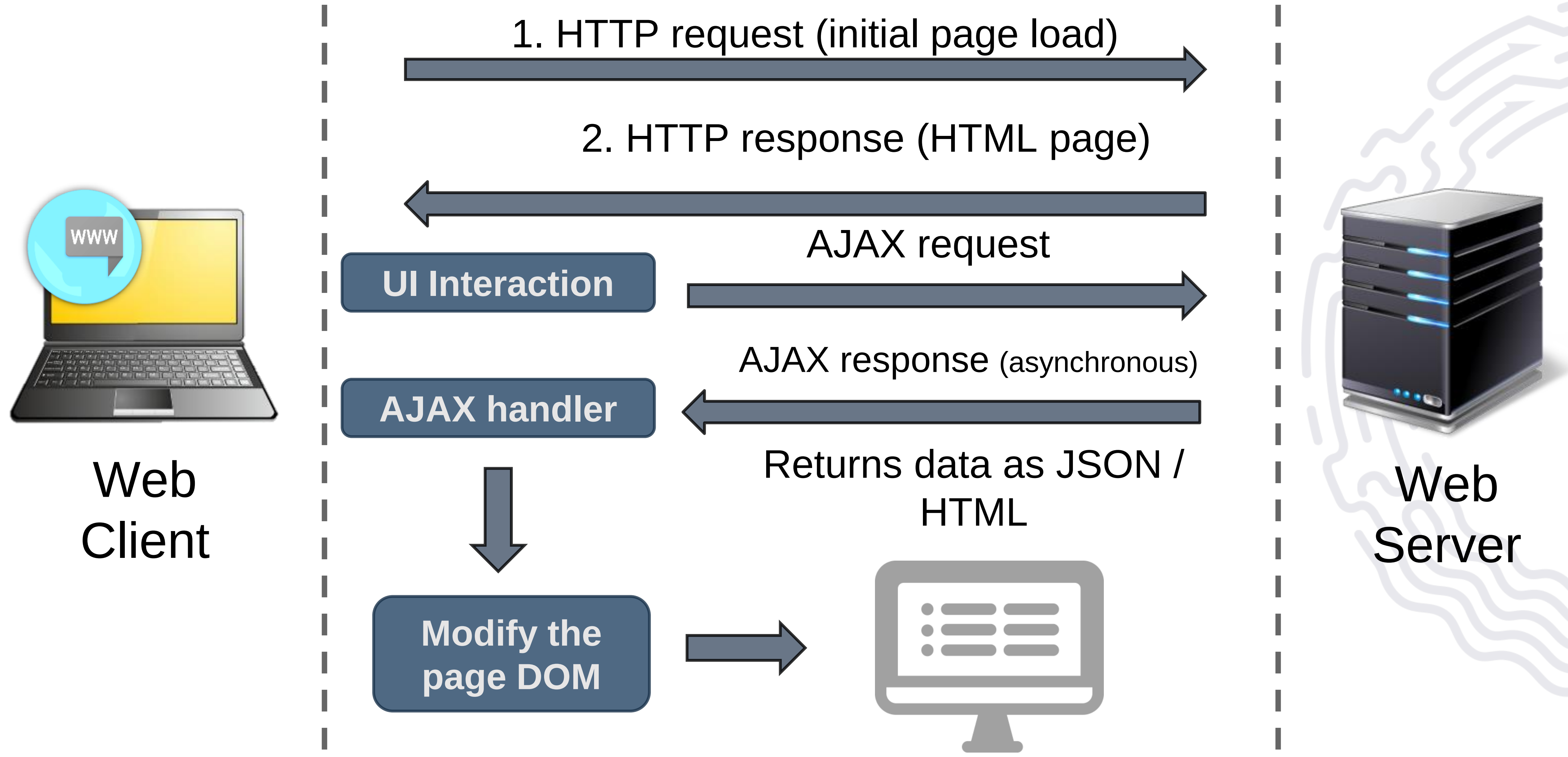
Asynchronous JavaScript and XML

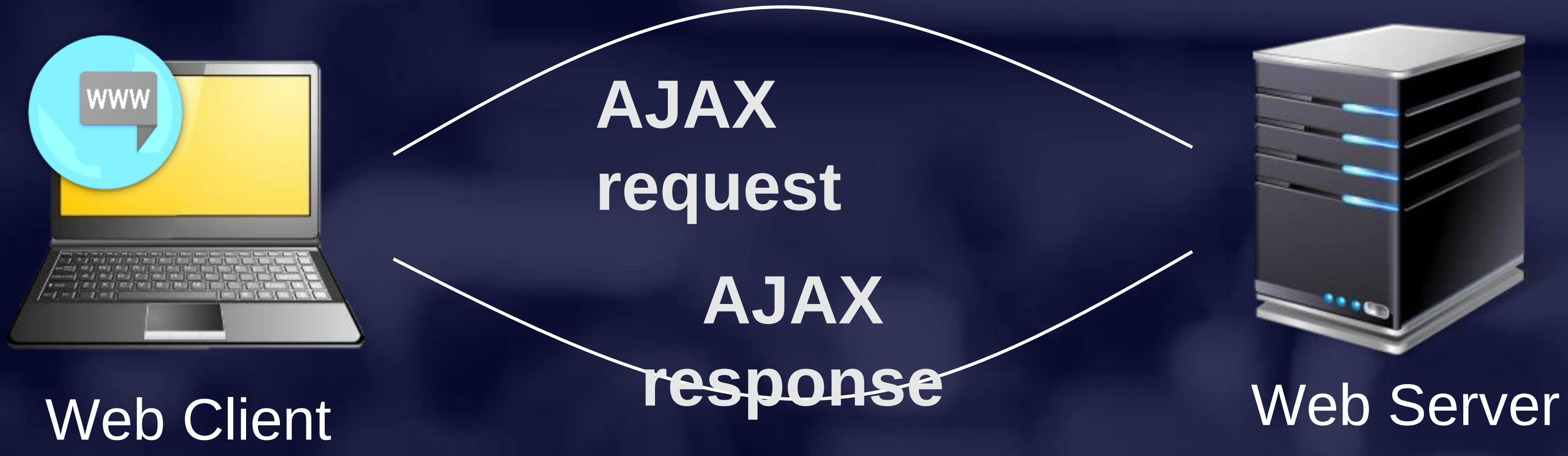
AJAX

What is AJAX?

- **Asynchronous JavaScript And XML**
  - Background loading of **dynamic content/data**
  - Load data from the Web server and **render** it
- Two types of AJAX
  - **Partial page rendering**
    - Load HTML fragment + show it in a **<div>**
  - **JSON service**
    - Loads JSON object and displays it

AJAX: Workflow


1. HTTP request (initial page load)
2. HTTP response (HTML page)
UI Interaction
AJAX request
AJAX response (asynchronous)
AJAX handler
Returns data as JSON / HTML
Modify the page DOM
WWW
Web Client
Web Server



Using the XMLHttpRequest Object

XMLHttpRequest – Standard API for AJAX

```html
<button id = "load">Load Repos</button>
<div id="res"></div>
```

```javascript
let button = document.querySelector("#load");
button.addEventListener('click', function loadRepos() {
   let url = 'https://api.github.com/users/testnakov/repos';
   const httpRequest = new XMLHttpRequest();
   httpRequest.addEventListener('readystatechange', function () {
      if (httpRequest.readyState == 4 && httpRequest.status == 200) {
         document.getElementById("res").textContent = httpRequest.responseText;
      }
   });
   httpRequest.open("GET", url);
   httpRequest.send();
});
```

What is Fetch?

- The **`fetch()`** method allows making network requests
- It is similar to **XMLHttpRequest** (XHR). The main **difference** is that the **Fetch API**:
  - Uses **Promises**
  - Enables a **simpler** and **cleaner** API
  - Makes code more readable and maintainable

```
fetch('./api/some.json')
  .then(function(response) {…})
  .catch(function(err) {…})
```

Basic Fetch Request

- The response of a **`fetch()`** request is a **Stream** object
- The **reading** of the stream happens **asynchronously**
- When the **`json()`** method is called, a **Promise** is **returned**
  - The **response status** is checked (should be **200**) **before parsing** the response as **JSON**

```
if (response.status !== 200) {
  // handle error
}
response.json()
  .then(function(data) { console.log(data)})
```

Chaining Promises

- When working with a JSON API, you can:
  - Define the **status** and **JSON parsing** in **separate functions**
  - The functions **return promises** which can be **chained**

```
fetch('users.json')
  .then(status)
  .then(json)
  .then(function(data) {…})
  .catch(function(error) {…});
```

GET Request

- **Fetch API** uses the **GET** method so that a direct call would be like this

```
fetch('https://swapi.co/api/people/4')
  .then((response) => response.json())
  .then((data) => console.log(JSON.stringify(data)))
  .catch((error) => console.error(error))
```

POST Request

- To make a **POST** request, we can set the **method** and **body** parameters in the **`fetch()`** options

```
fetch('/url', {
    method: 'post',
    headers: { 'Content-type': 'application/json' },
    body: JSON.stringify(data),
})
```

Body Methods

- **`clone()`** - create a clone of the response
- **`json()`** - resolves the promise with JSON
- **`redirect()`** - create new promise but with different URL
- **`text()`** - resolves the promise with string

Response Types

- `basic` - normal, same origin response
- `cors` - response was received from a valid cross-origin request
- `error` - error network
- `opaque` - Response for "no-cors" request to cross-origin resource

Body Methods (2)

- **`opaqueredirect`** - the fetch request was made with **redirect: "manual"**
- **`arrayBuffer()`** - return a promise that resolve with an ArrayBuffer
- **`blob()`** - determinates with a Blob
- **`formData()`** - return promise that determinate with FormData object

Live Exercises

Summary

- **HTTP** is text-based request-response protocol
- **REST** uses **GET**, **POST**, **PUT**, **PATCH**, **DELETE**
- **RESTful** services address resources by URL
  - Provide **CRUD** operations over HTTP
- **AJAX** background loading of dynamic content
  - **XMLHttpRequest**
  - **Fetch**

Questions?

License

KINGSLAND UNIVERSITY

THANK YOU