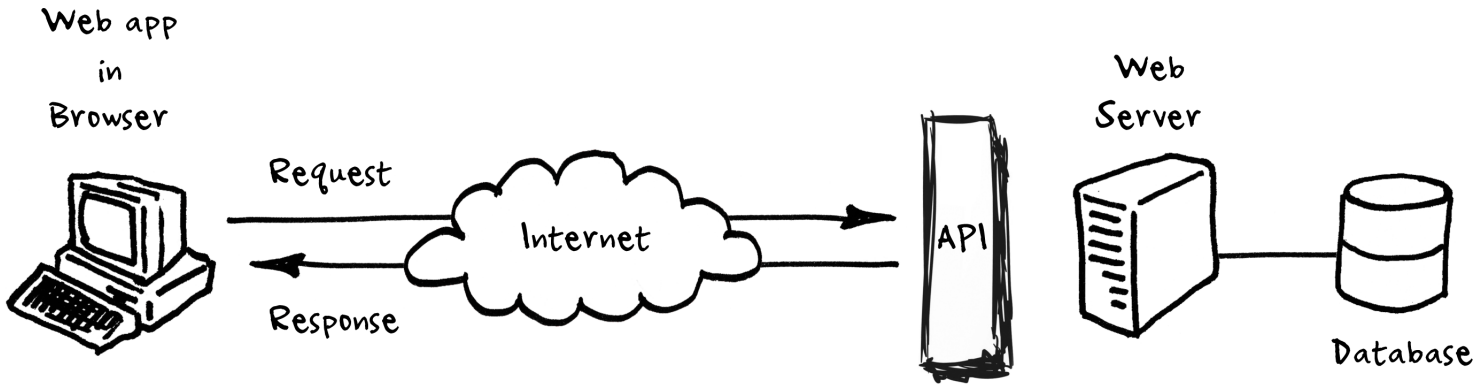
Web API using JavaScript



Outline

1. Web and HTTP
2. Web API
3. Web API using Node.js Express
4. Implementing CRUD Operations

Course Roadmap

Web Client

Request

Response

Web Server

Frontend development

- HTML for page structure
- CSS for styling
- JavaScript for interaction
- UI Components

Backend development

- Web API
- Data Management

express + node

mongoDB

What is a Web API

- Web API: A set of methods exposed over the web via HTTP to allow **programmatic access to applications**
- Web API are designed for **broad reach**:
  - Can be accessed by a broad range of clients including browsers and mobile devices
  - Can be implemented or consumed in any language
- Uses HTTP as an **application protocol**


WWW

Web and HTTP

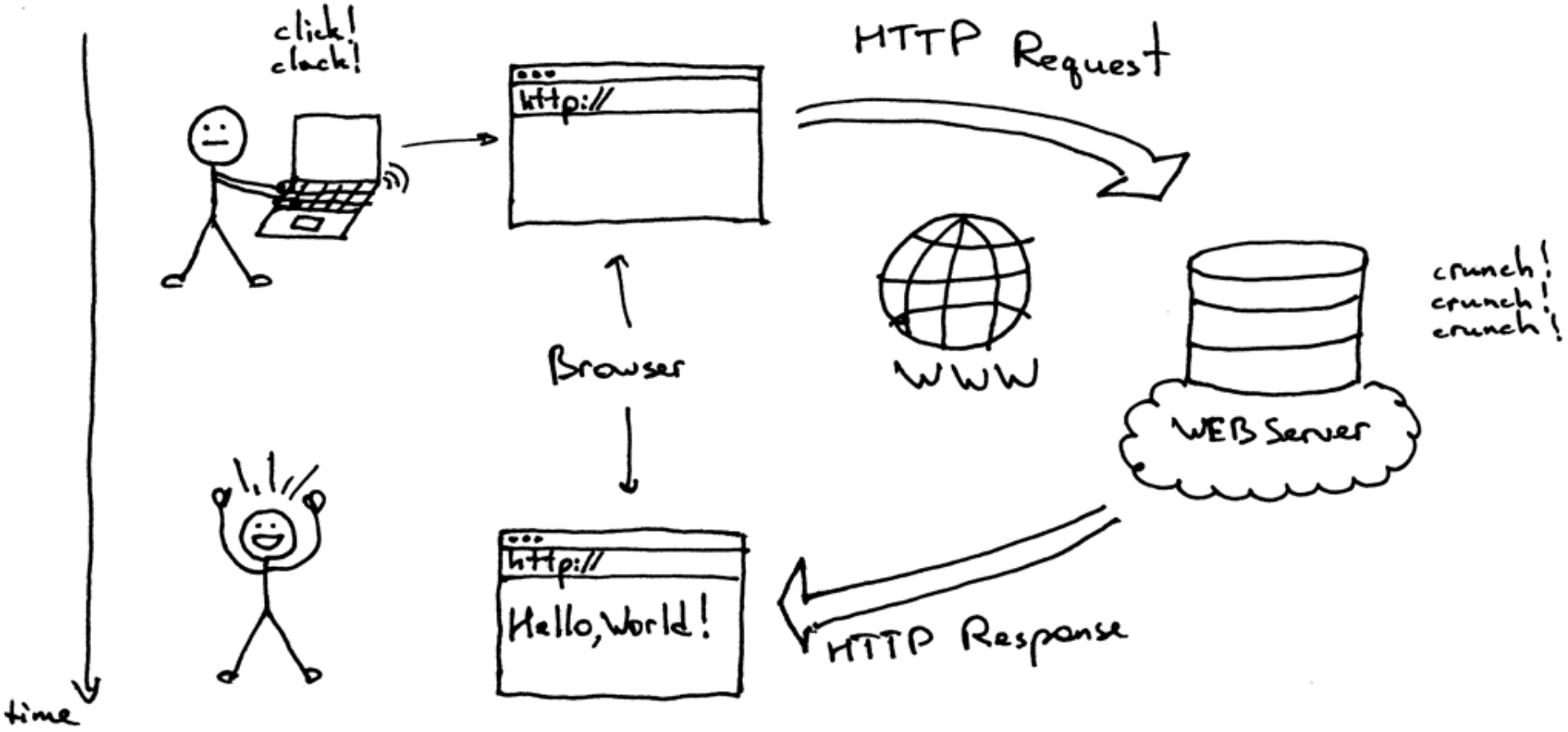

GET /index.html
HTTP/1.1
HTTP/1.1 200 OK
"Welcome to our
Web site!"

What is Web?

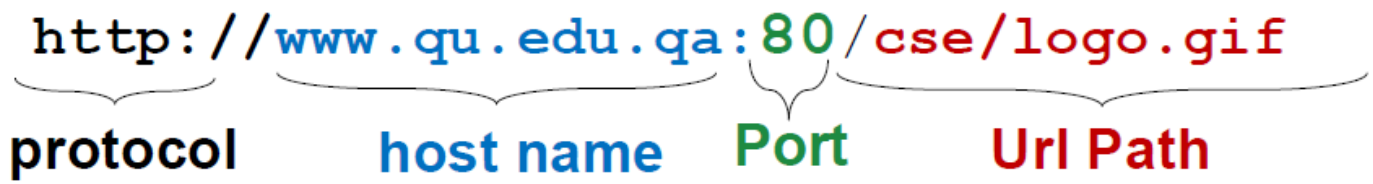
- Web = **global distributed system of interlinked resources accessed over the Internet using the HTTP protocol**
  - Consists of set of **resources** located on different servers:

    HTML pages, images, videos and other resources
  - Resources have unique **URL** (Uniform Resource Locator) address
  - Accessed through standard **HTTP** protocol
- The Web has a Client/Server architecture:
  - **Web browser** (client) requests resources (using HTTP protocol) and displays them
  - **Web server** sends resources in response to requests (using HTTP protocol)

How the Web Works?



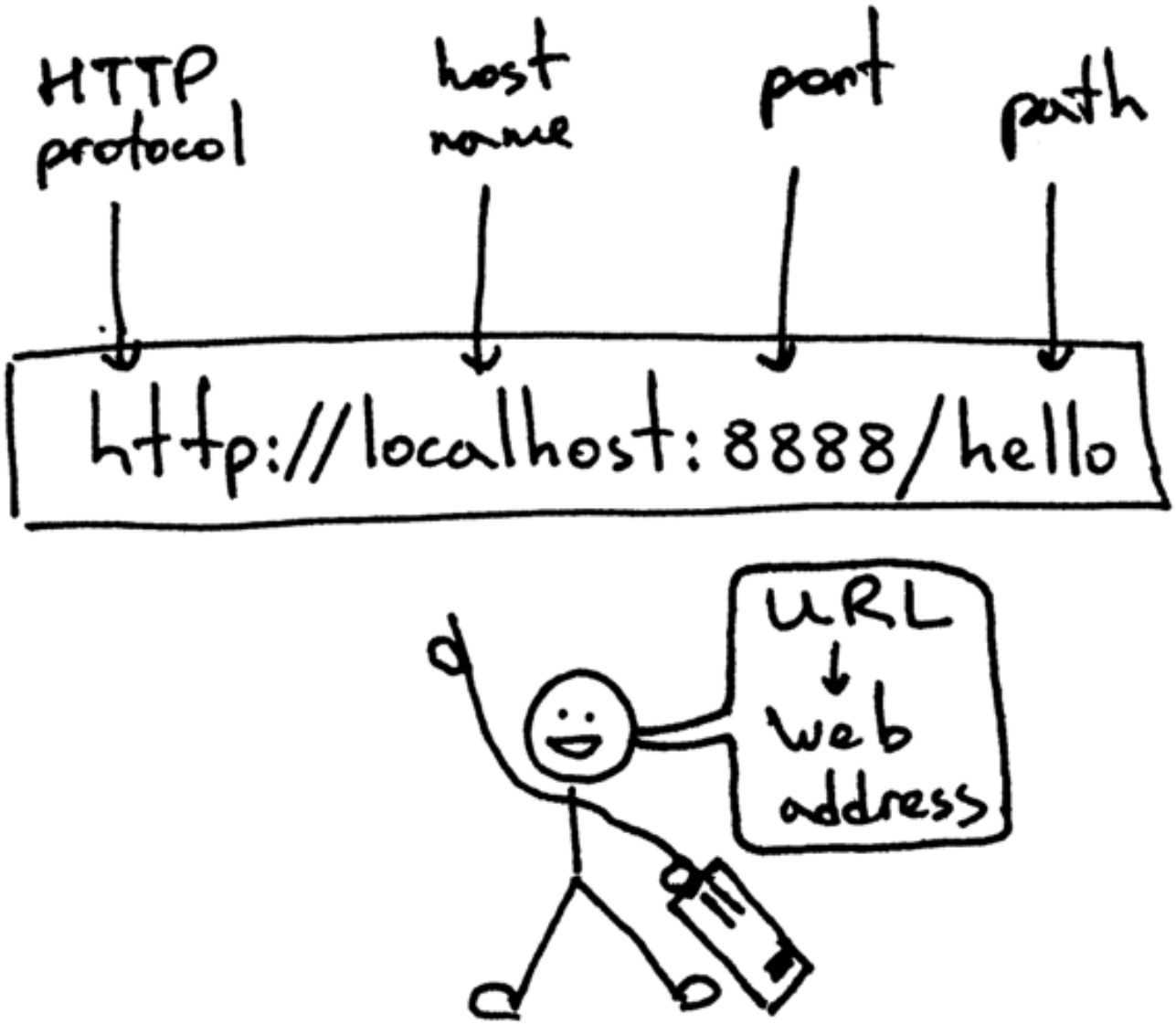
Uniform Resource Locator (URL)



- URL is a formatted string, consisting of:
  - **Protocol** for communicating with the server (e.g., http, https, ...)
  - **Name of the server or IP** address plus port (e.g. qu.edu.qa:80, localhost:8080)
  - **Path of a resource** (e.g. **/ceng/index.html**)
  - **Parameters** aka **Query String** (optional), e.g.

    https://www.google.com/search**?q=qatar%20university**

URL Example


HTTP protocol
host name
port
path
http://localhost:8888/hello
URL
web address

URL Encoding

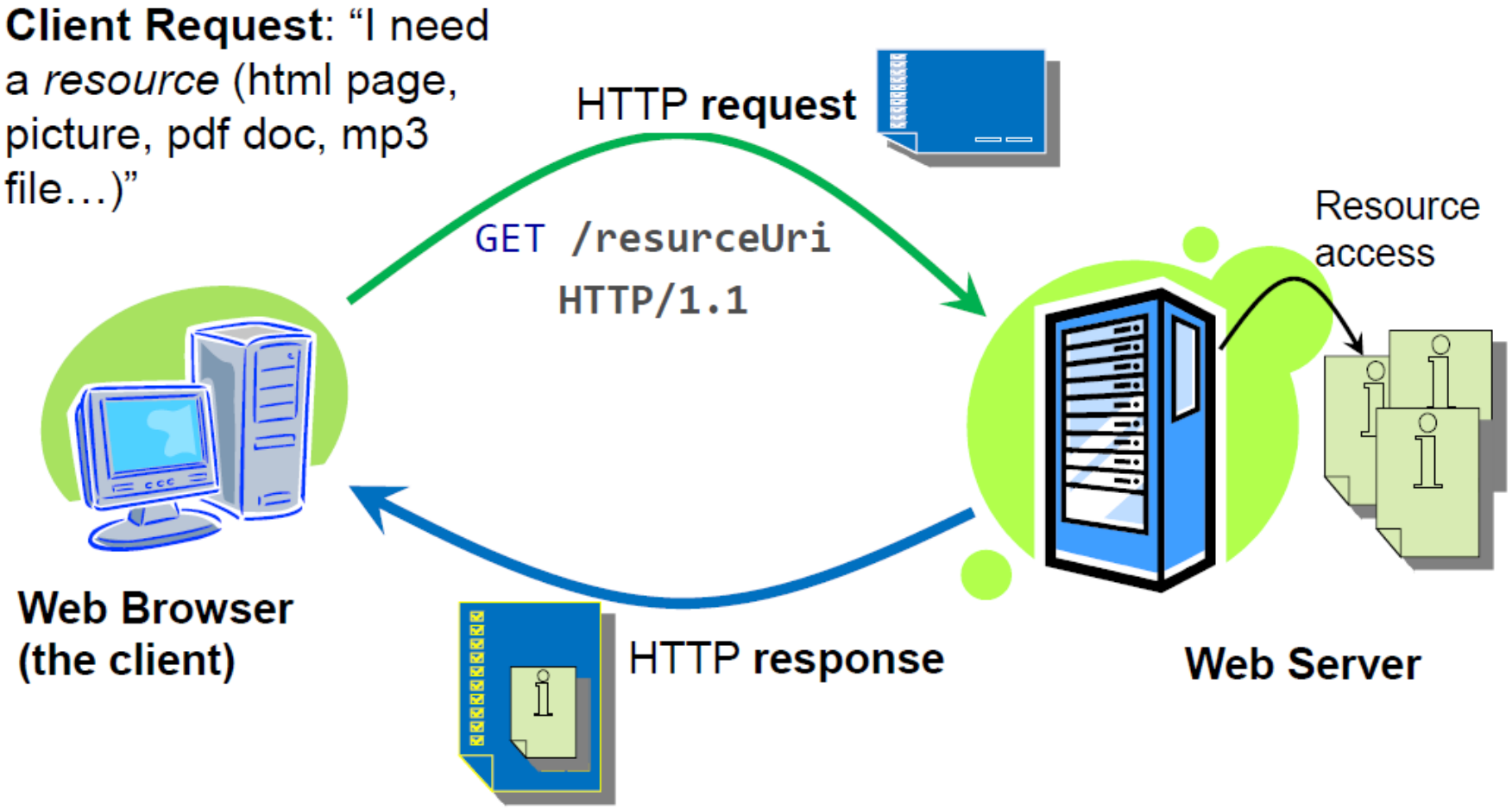
- According RFC 1738, the characters allowed in URL are alphanumeric [0-9a-zA-Z] and the special characters $-_.+!*'()
- Unsafe characters should be encoded, e.g.,

  http://google.com/search?q=qatar%20university

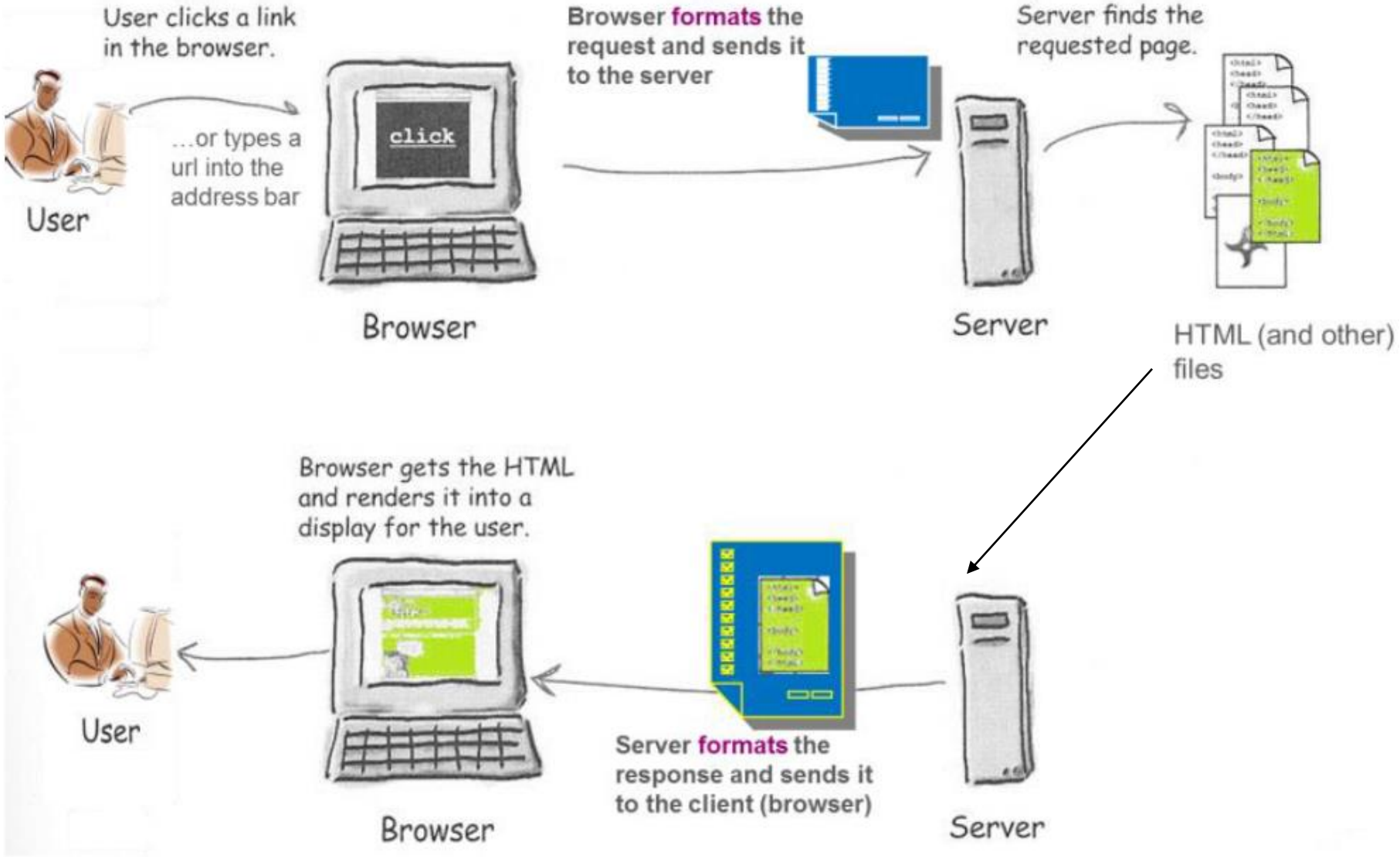
Commonly encoded values:

ASCII Character	URL-encoding
space	%20
!	%21
"	%22
#	%23
$	%24
%	%25
&	%26

Web uses Request/Response interaction model
HTTP is the *message protocol* of the Web

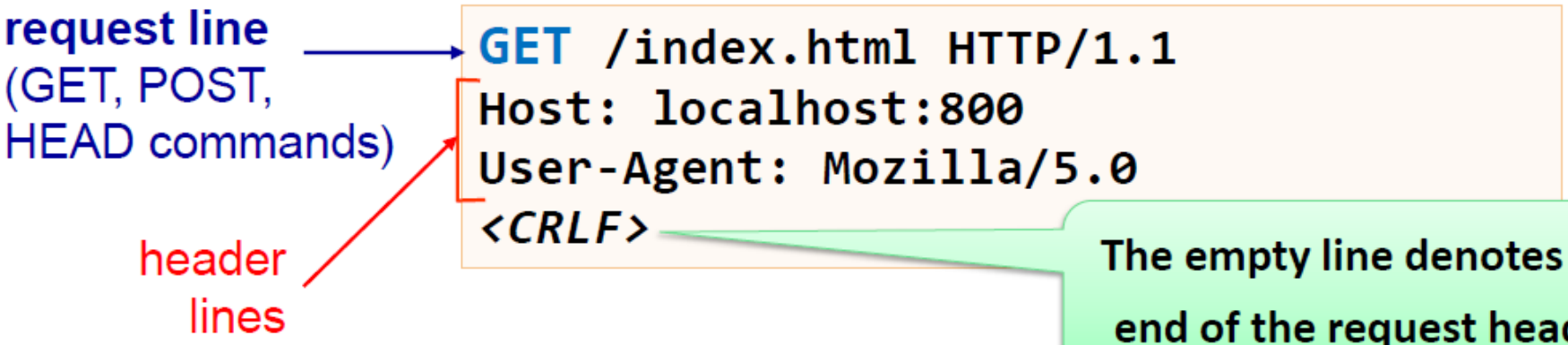


The sequence for retrieving a resource



Request and Response Examples

- **HTTP request:**



- **HTTP response:**

HTTP Request Message

- Request message sent by a client consists of

– **Request line** – request method (GET, POST, HEAD, ...), resource URI, and protocol version

– **Request headers** – additional parameters

– **Body** – optional data

  - e.g. posted form data, files, etc.

```
<request method> <URI> <HTTP version>
<headers>
<empty line>
<body>
```

HTTP Request Methods

- **GET**
  - – **Retrieve a resource** (could be static resource such as an image or a dynamically generated resource)
  - – Input is appended to the request URL E.g.,

  http://google.com/?q=Qatar
- **POST**
  - – **Create or Update a resource**
  - – Web pages often include form input. Input is submitted to server in the **message body.** E.g.,


```
POST /calc HTTP/1.1
Host: localhost
Content-Type: application/x-www-form-urlencoded
Content-Length: 27
num1=20&operation=*&num2=10
```

HTTP Response Message

- Response message sent by the server

– **Status line** – protocol version, status code, status phrase

– **Response headers** – provide metadata such as the Content-Type

– **Body** – the contents of the response (i.e., the requested resource)

```
<HTTP version> <status code> <status text>
<headers>
<empty line>
<response body>
```

HTTP Response – Example

status line (protocol status code status text)

Try it out and see HTTP in action using **HttpFox**

```
HTTP/1.1 200 OK
Content-Type: text/html
Server: QU Web Server
Content-Length: 131
<CRLF>
<html>
  <head><title>Calculator</title></head>
  <body>20 * 10 = 200
      <br><br>
      <a href='/calc'>Calculator</a>
  </body>
</html>
```

HTTP response headers

The empty line denotes the end of the response header

Response body. e.g., requested HTML file

Common Internet Media Types

- The **Content-Type** header describes the media type contained in the body of HTTP message
- **Full list @** http://en.wikipedia.org/wiki/MIME_type
- Commonly used media types (**type/**subtype):

Type/Subtype	Description
application/json	JSON data
image/gif	GIF image
image/png	PNG image
video/mp4	MP4 video
text/xml	XML
text/html	HTML
text/plain	Just text

HTTP Response Staus Codes

- Status code appears in 1st line in the response message
- HTTP response code classes
  - 2xx: success (e.g., “200 OK”)
  - 3xx: redirection (e.g., "302 Found")

  "302 Found" is used for redirecting the Web browser to another URL
  - 4xx: client error (e.g., “404 Not Found”)
  - 5xx: server error (e.g., “503 Service Unavailable”)

Popular Status Codes

Code	Reason	Description
200	OK	Success!
301	Moved Permanently	Resource moved, don't check here again
302	Moved Temporarily	Resource moved, but check here again
304	Not Modified	Resource hasn't changed since last retrieval
400	Bad Request	Bad syntax?
401	Unauthorized	Client might need to authenticate
403	Forbidden	Refused access
404	Not found	Resource doesn't exist
500	Internal Server Error	Something went wrong during processing
503	Service Unavailable	Server will not service the request

Browser Redirection

- HTTP browser redirection example
  - HTTP GET requesting a moved URL:

(Request-Line)	GET /qu HTTP/1.1
Host	localhost:800
User-Agent	Mozilla/5.0 (Windows NT 6.3; WOW64; rv:27.0) Gecko/20100101 Firefox/27.0
Accept	text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8

  - The HTTP response says that the browser should request another URL:

(Status-Line)	HTTP/1.1 301 Moved Permanently
Location	http://qu.edu.qa

2

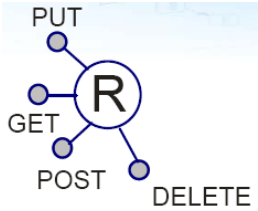
Web API
(aka REST Services)

What is a REST Service?

- Web API = Web accessible Application Programming Interface. Also known as REST Services.
- Web API is a web service that accepts requests and returns **structured data** (JSON in most cases)
  - Programmatically accessible at a particular URL
  - You can think of it as a Web page returning JSON instead of HTML

- Major goal = **interoperability between heterogeneous systems**

REST Principles



- **Resources have unique address** (**nouns**) i.e., a URI

e.g., http://example.com/customers/123

- **Can use a Uniform Interface** (**verbs**) to access them:

  - HTTP verbs: GET, POST, PUT, and DELETE

- **Resource has representation(s)** (**data format**)

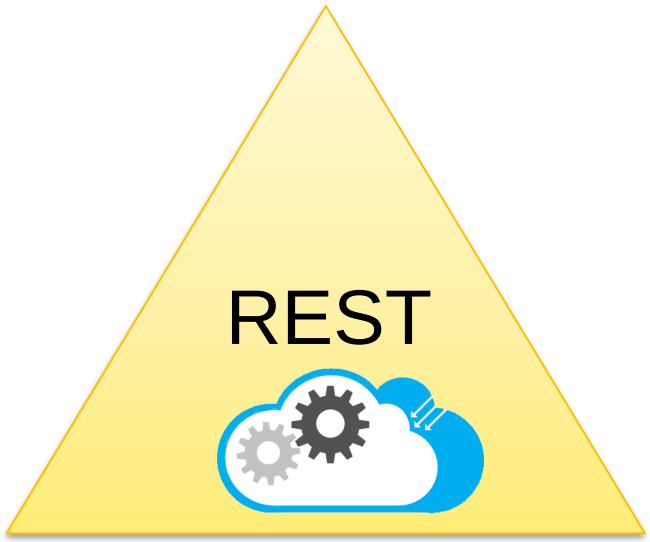
  - A resource can be in a variety of data formats: JSON, XML, RSS..

Resources

- The key abstraction in REST is a **resource**
- A resource is a conceptual mapping to a set of entities
  - Any **information that can be named can be a resource**: a document or image, a temporal service (e.g. "today's weather in Doha"), a collection of books and their authors, and so on

REST Services Main Concepts

Nouns (Resources)
e.g., http://example.com/employees/12345



Verbs
e.g., GET, POST

Representations
e.g., XML, JSON

Naming Resources

- REST uses URL to identify resources

> Dedicated **api** path is recommended for better organization

  - http://localhost/**api**/books/
  - http://localhost/api/books/ISBN-0011
  - http://localhost/api/books/ISBN-0011/authors

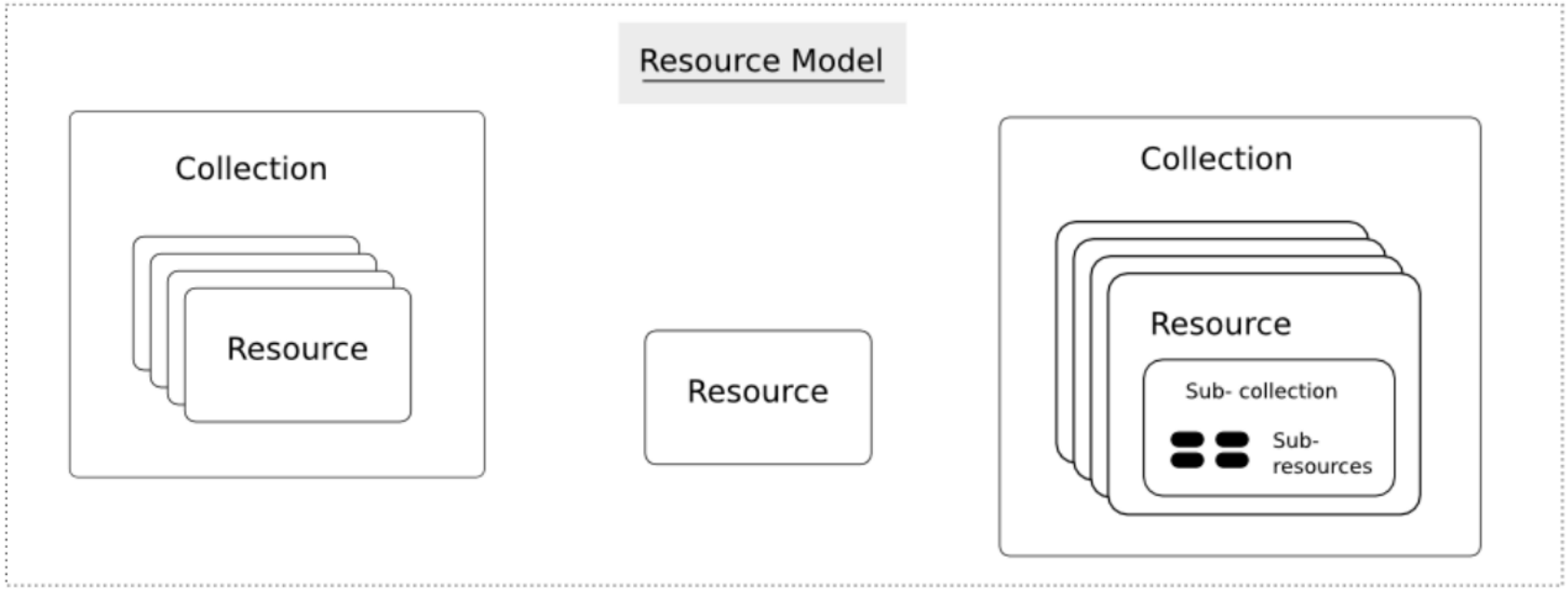
  - http://localhost/api/classes
  - http://localhost/api/classes/cmps356
  - http://localhost/api/classes/cs356/students

- As you traverse the **path** from more generic to more specific, you are navigating the data

Example CRUD (Create, Read, Update and Delete) API that manages books

- Create a new book
  - **POST** /books
- Retrieve all books
  - **GET** /books
- Retrieve a particular book
  - **GET** /books/:id
- Replace a book
  - **PUT** /books/:id
- Update a book
  - **PATCH** /books/:id
- Delete a book
  - **DELETE** /books/:id

A Collection with Resources



A Collection with Resources

A Singleton Resource

Sub-collections and Sub-resources

Representations

Two main formats:

- **JSON**

```
{
    code: 'cmp123',
    name: 'Web Development'
}
```

- **XML**

```
<course>
    <code>cmp123</code>
    <name>Web Development</name>
</course>
```

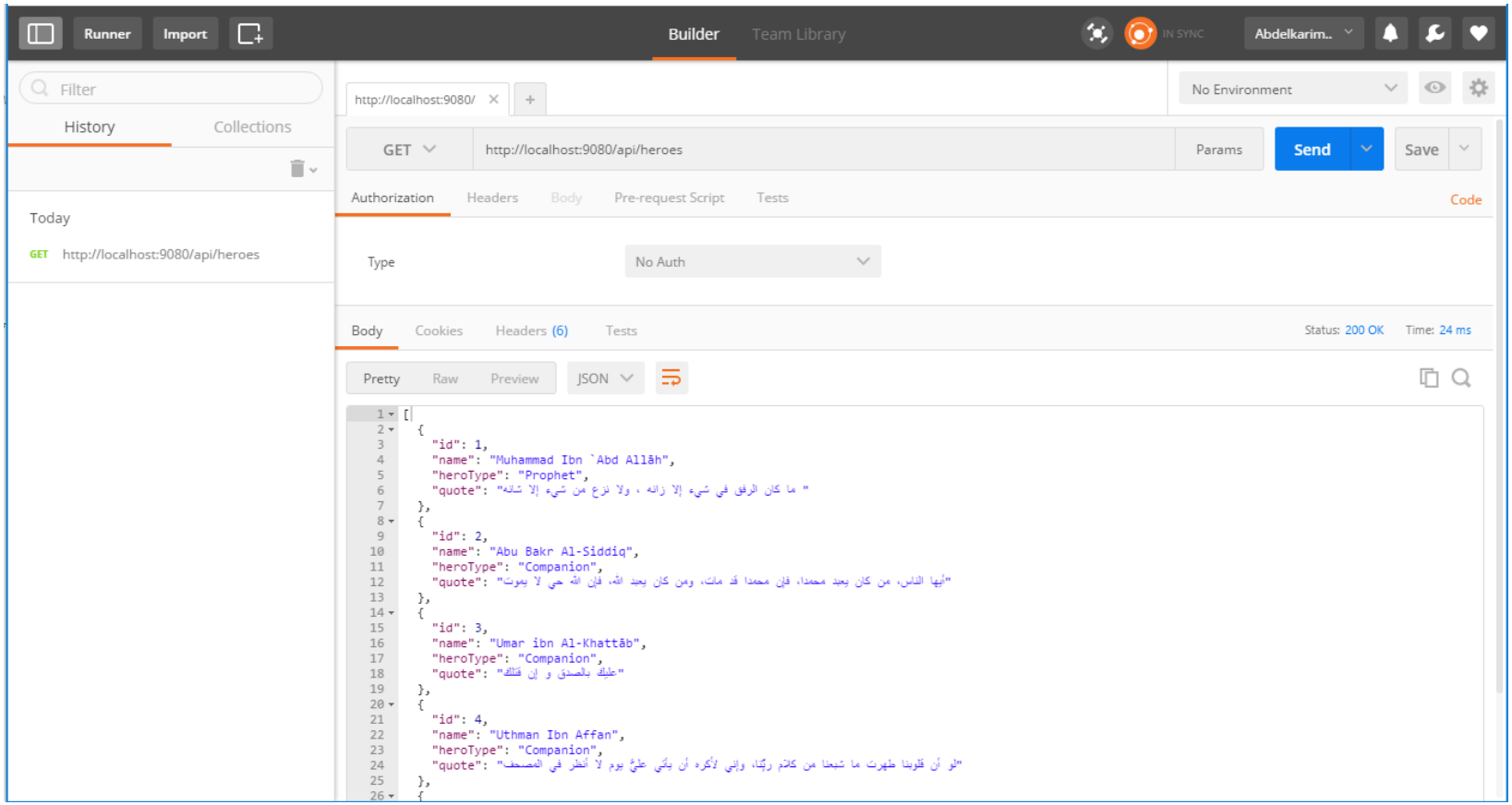
HTTP Verbs

- Represent the actions to be performed on resources
- Retrieve a representation of a resource: **GET**
- Create a new resource:
  - Use **POST** when the server decides the new resource URI
    - Post is not repeatable
  - Use **PUT** when the client decides the new resource URI
    - Put is repeatable
- **PUT** is typically used for update
- Delete an existing resource: **DELETE**
- Get metadata about an existing resource: **HEAD**
- Get which of the verbs the resource understands: **OPTIONS**

Testing REST Services

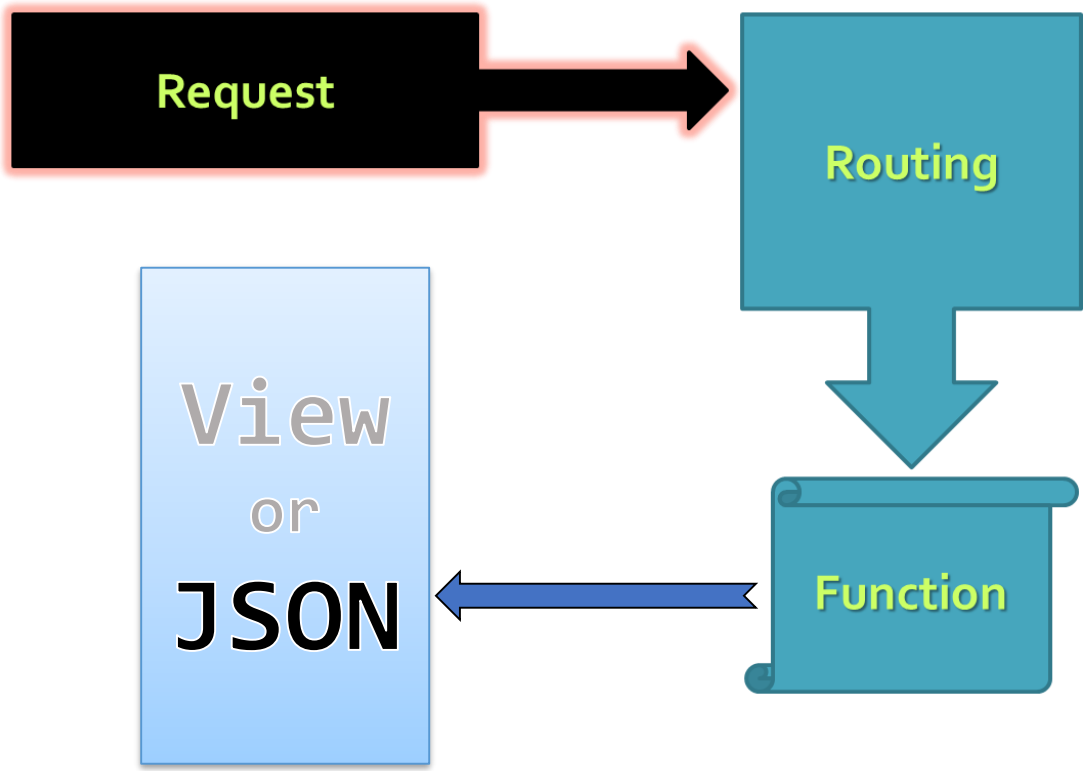
- Using Postman to test Web API

https://www.getpostman.com/postman



3

Web API using Node.js Express



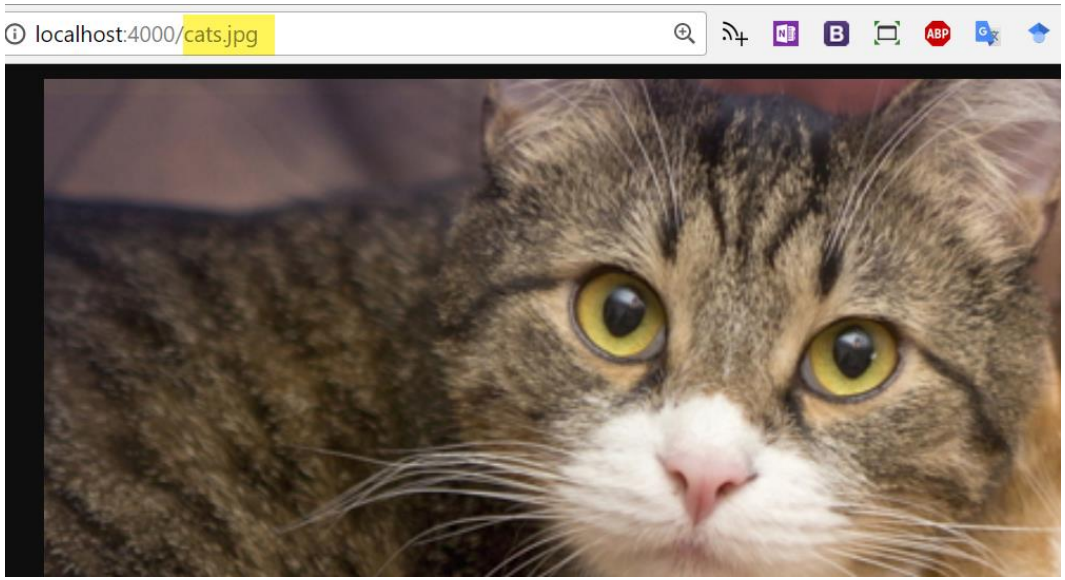
Create and Start an Express App

```
let express = require('express');
let app = express();

app.get('/', (req, res) => {
    res.send('السلام عليكم ورحمة الله وبركاته')
});

let port = 3000;
app.listen(port, () => {
    console.log(`App is available @ http://localhost:${port}`)
});
```

- A function **registered** to listen to the URL http://localhost:3000/
- When someone visits this Url the function associated with **get** **'/'** will run and 'السلام عليكم ورحمة الله وبركاته' will be returned to the requester

Serving Static Resources using Express

- To serve up static files, the **express.static** middleware function is used with the folder path of the files to be served



Routing

- Requests can be routed based on:
  - **HTTP Verb** – GET, POST, PUT, DELETE
  - **URL Path** – e.g., /users
- App Route **maps** an **HTTP Verb** (e.g., GET or POST) + a **URI Path** (like /users/123) to a **route handler function**
  - The handler function is passed a **req** and a **res** objects
  - The **req** object represents the HTTP request and has the query string, parameters, body and HTTP headers
  - The **res** object represents the HTTP response and it is used to send the generated response

36

Path Parameters

- Named **path parameters** can be added to the URL path. E.g., **/students/:id**
- **req.params** is an object containing properties mapped to the named path parameters
  - E.g., if you have the path **/students/:id**, then the "id" property is available as **req.params.id**

```
app.get('/api/students/:id', (req, res) => {
    const studentId = req.params.id
    console.log('req.params.id', studentId)
})
```

```
app.get('/authors/:authorId/books/:bookId', (req, res) => {
    // If the Request URL was http://localhost:3000/authors/34/books/8989
    // Then req.params: { authorId: "34", bookId: "8989" }
    res.send(req.params);
})
```

Query Parameters

- Named **query parameters** can be added to the URL path after a ? E.g., **/posts?sortBy=createdOnDate**
- Query parameters are often used for **optional** parameters (e.g., optionally specifying the property to be used to sort of results)
- **req.query** is an object containing a property for each query parameter in the URL path
  - If you have the path /posts?**sortBy**=createdOnDate, then the "**sortBy**" property is available as req.query.**sortBy**

```
app.get('/api/students?sortBy=studentId', (req, res) => {
    // req.query.sortBy  => "studentId"
    const sortBy = req.query.sortBy
    console.log(req.query.sortBy', sortBy)
})
```


38

Working with a Request Body

- To access the request body a middleware is used to parse the request body
- **express.json()** is a middleware function that extracts the body portion of an incoming request and assigns it to **req.body**

```
const express = require('express');
const app = express();
app.use( express.json() );
app.post('/heroes', async (req, res) => {
    let hero = req.body;
    await heroRespository.addHero(hero);
    res.status(201);
});
```

Express Router

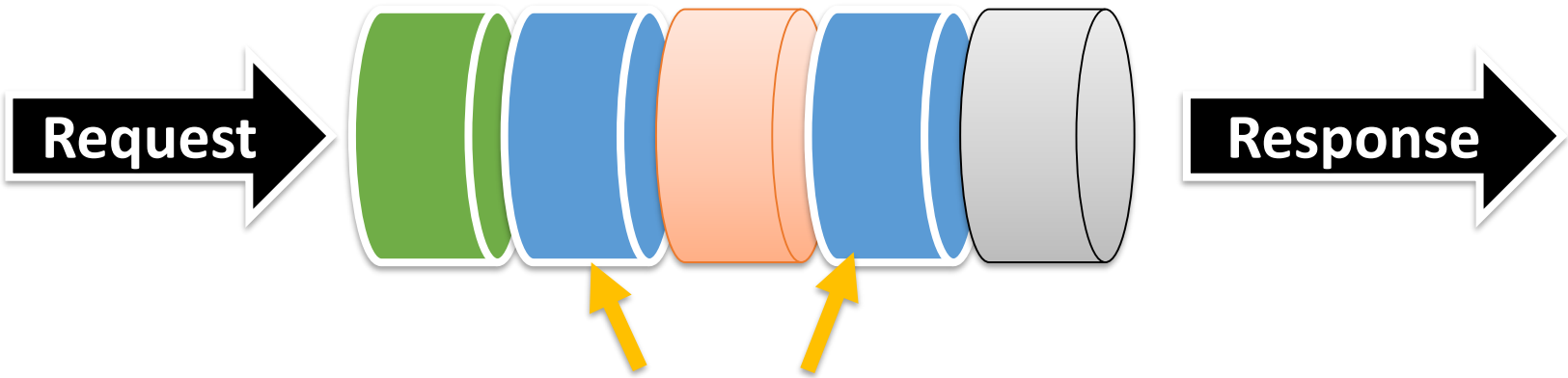
- For simple app routes can defined in **app.js**
- For large application, Express Router allows defining the routes in a separate file(s) then attaching routes to the app to:
  - Keep *app.js* clean, simple and organized
  - Easily find and maintain routes

```
// routes.js file
let router = express.Router()
router.get('/api/students', studentController.getStudents )
module.exports = router

//app.js file - mount the routes to the app
let routes = require('./routes')
app.use('/', routes)
```

Express Middleware

- Express middleware allows **pipelining** a request through a series of functions.
  - Each middleware function may **modify** the request or the response
- Request Processing Pipeline: the request passes through an *array* of functions before it reaches the route handler

```
/* express.json() is a middleware function that extracts the body portion of
an incoming request and assigns it to req.body.
 */
app.use( express.json() );
```

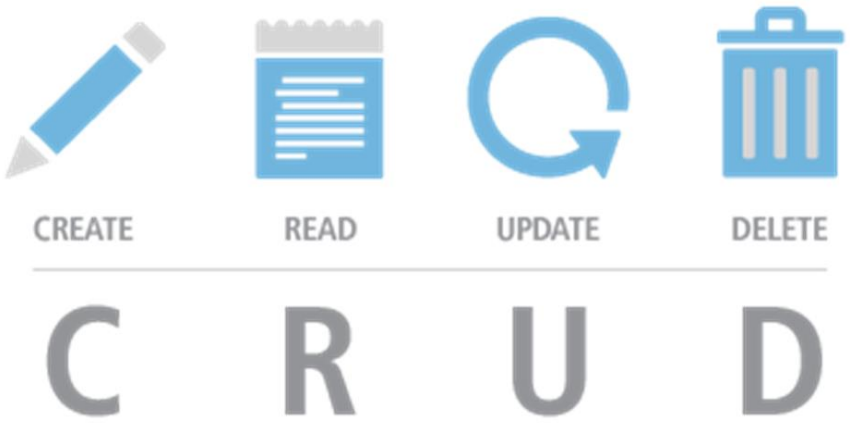


Middleware (body parser, logging, authentication, router etc.)

Custom Middleware Example

```
const express = require('express’);
const app = express();

//Define a middleware function
function logger (req, res, next) {
    req.requestTime = new Date();
    console.log(`Request received at ${req.requestTime}`);
    next();
}

// Attach it to the app
app.use(logger);

app.get('/', function (req, res) {
    const responseText = `Hello World! Requested at: ${req.requestTime}`;
    res.send(responseText);
})
```

4



Implementing CRUD Operations

CRUD Operations

- See the posted Hero and Student Examples

```
const heroService = require('./services/HeroService');

//Heroes Web API
router.route('/heroes')
    .get( heroService.getHeroes )
    .post( heroService.addHero );

router.route('/heroes/:id')
    .get( heroService.getHero )
    .put( heroService.updateHero )
    .delete( heroService.deleteHero );
```

Summary

- Express is a popular and easy to use web framework
- It makes building an Http Server and Web APIT a lot easier
- Provides routing and static content delivery out of the box
- Uses **express.json()** middleware to parse the request body

Resources

- Express Documentation

https://expressjs.com/en/4x/api.html

- Web API Design

- https://docs.microsoft.com/en-us/azure/architecture/best-practices/api-design
- https://pages.apigee.com/rs/apigee/images/api-design-ebook-2012-03.pdf

- Mozilla Developer Network

https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express_Nodejs