RESTful APIs

Evaluate Your Progress so far

- Can you add basic JS code to HTML page? **…YES**
- Can you add basic Jquery function to HTML page? **…YES**
- Can you use JS to change an element border color? **…YES**
- Can you write and run basic NodeJS programs? **…YES**
- Have you already decided on your idea? **…YES**
- Have you made initial planning for the idea? **…YES**

If answer to all is **YES** then you are on track

Representational **S**tate **T**ransfer (REST)

- REST relies on HTTP protocol
- HTTP protocol The underlying architectural principle of the web
- simple HTTP is used to make calls between machines
- RESTful applications use HTTP requests to post data (create and/or update), read data (e.g., make queries), and delete data. Thus, REST uses HTTP for all four CRUD (Create/Read/Update/Delete) operations.

What is REST

- Client contacts the server, the server responds
- Client asks for a resource (image, html page, video, data,…etc)
- Each resource has a unique URI
- Client performs the requests using HTTP
- Client may send data if necessary using JSON
- Server responds with JSON and a response code

URI + HTTP + JSON

What is an URI?

- We already understand that a URL (Uniform Resource Locator) locates a unique resource on the Internet:

  http://www.example.com/mypage.html

- A web URL uniquely identifies a web page anywhere in the world
- So we can use the same idea to uniquely identify the thing we want to interact with using the same principle
- We are not dealing with web pages so we can call it a **URI** (Universal resource Identifier) rather than a URL

- a URI to uniquely locate anything on the Internet:

  http://www.example.com/bookshop/book/id/42

- This uniquely locates record 42 in the books table in a bookshop database

URI Example

`http://api.example.com:8080/books?q=nodejs&order=asc`	
Protocol	how your web browser should communicate with a web server when sending or fetching a web page or document
Subdomain	a subdivision of the main domain name
Domain Name	a unique reference that identifies a web site on the internet
Port	a way to identify a specific process to which an Internet or other network message is to be forwarded when it arrives at a server
Path	Identifies a resource or collection on the server
Query	Separates the path from the parameters
Parameter	Additional information supplied as key-value pairs. Each pair is separated using the & character

HTTP Protocol

- The Hypertext Transfer Protocol (HTTP) is an application-level protocol for distributed, collaborative, hypermedia information systems.
- It is the foundation protocol for the World Wide Web (WWW)
- HTTP is a TCP/IP based communication protocol (default port is 80)

HTTP Protocol

- **HTTP is connectionless:** after a request is made, the client disconnects from the server and waits for a response. The server processes the request and re-establishes the connection with the client to send a response back.
- **HTTP is media independent:** It means, any type of data can be sent by HTTP as long as both the client and the server know how to handle the data content. It is required for the client as well as the server to specify the content type using appropriate MIME-type.
- **HTTP is stateless:**The server and client are aware of each other only during a current request. Afterwards, both of them forget about each other.

HTTP Request

Verb	Indicates the HTTP methods such as GET, POST, DELETE, PUT, etc.
URI	Uniform Resource Identifier (URI) to identify the resource on the server.
HTTP Version	Indicates the HTTP version. For example, HTTP v1.1.
Request Header	Contains metadata for the HTTP Request message as key-value pairs. For example, client (or browser) type, format supported by the client, format of the message body, cache settings, etc.
Request Body	Message content or Resource representation.

Example of HTTP Request

```
POST /accounts HTTP/1.1
Host: example.com
Content-Type: application/json
Authorization: Basic
am9obmRvZTpwYXNzd29yZA==
Content-Length: 20

{"Name": "John Doe"}
```

The HTTP Response

Response Code	Indicates the HTTP methods such as GET, POST, DELETE, PUT, etc.
HTTP Version	Indicates the HTTP version. For example, HTTP v1.1.
Response Header	Contains metadata for the HTTP Response message as key-value pairs. For example, content length, content type, response date, server type, etc.
Response Body	Response message content or Resource representation.

Example of HTTP Response

```
HTTP/1.1 201 Created
Date: Mon, 27 Jul 2016 12:28:53 GMT
Server: Apache/2.2.14 (Ubuntu)
eTag: 4cvfkjg85jh4lklnd7
Content-Length: 43
Content-Type: application/json
Connection: Closed

{"name": "John Doe", "username": "johndoe"}
```

HTTP Methods

Method	CRUD	Description
POST	Create	Upload data and create a new record defined by the URI
GET	Retrieve	Retrieve data from the specified URI
PUT	Update	Update a record specified by the URI
DELETE	Delete	Delete the record defined by the URI

- GET cannot change a resource
- This makes it SAFE

- DELETE, PUT, POST can change a resource
- These are UNSAFE

GET Method

- GET: Requests data from a specified resource

/test/demo_form.asp**?name1=value1&name2=value2**

- GET requests can be cached
- GET requests remain in the browser history
- GET requests can be bookmarked
- GET requests should never be used when dealing with sensitive data
- GET requests have length restrictions
- GET requests should be used only to retrieve data

POST Method

- POST: Submits data to be processed to a specified resource

POST /test/demo_form.asp HTTP/1.1
Host: myServer.com
name1=value1&name2=value2

- POST requests are never cached
- POST requests do not remain in the browser history
- POST requests cannot be bookmarked
- POST requests have no restrictions on data length

PUT Method

- PUT is for creating when you know the URL of the thing you will create.
- **PUT** is for creating or replacing a resource at a **URL known by the client**.
- **PUT** replaces the resource at the known url if it already exists.
- PUT has mostly been used to update resources

Other HTTP Methods

- HEAD: Same as GET but returns only HTTP headers and no document body
- DELETE: Deletes the specified resource
- OPTIONS: Returns the HTTP methods that the server supports
- CONNECT: Converts the request connection to a transparent TCP/IP tunnel

(HTTP) Response Status Codes

- If something goes wrong we need to get an error code from the server so we understand the problem

- 1xx: Information
- 2xx: Successful
- 3xx: Redirection
- 4xx: Client Error
- 5xx: Server Error

Full list can be found here:
http://www.w3schools.com/tags/ref_httpmessages.asp

Passing Data from Client to Server

- HTTP Header
- GET Parameters

  http://example.com/page?parameter=value&also=another
- POST Parameters

  In a form of JSON or XML, hidden inside the HTTP body
- Path Parameter

  http://example.com/page/value/another

Passing parameters from Server to Client

- HTTP Header
- Response Body
  - In any form, JSON, XML, HTML, image, file, text…etc
- Status code

Data Format

- We have already defined the structure of our request
- This takes the form of a URI
- The data we get back also needs to be in a standard format
- This will allow it to be understood by the client computer
- There are two possible formats:
- XML or JSON
- Most web services use the JSON format
- **J**ava**S**cript **O**bject **N**otation

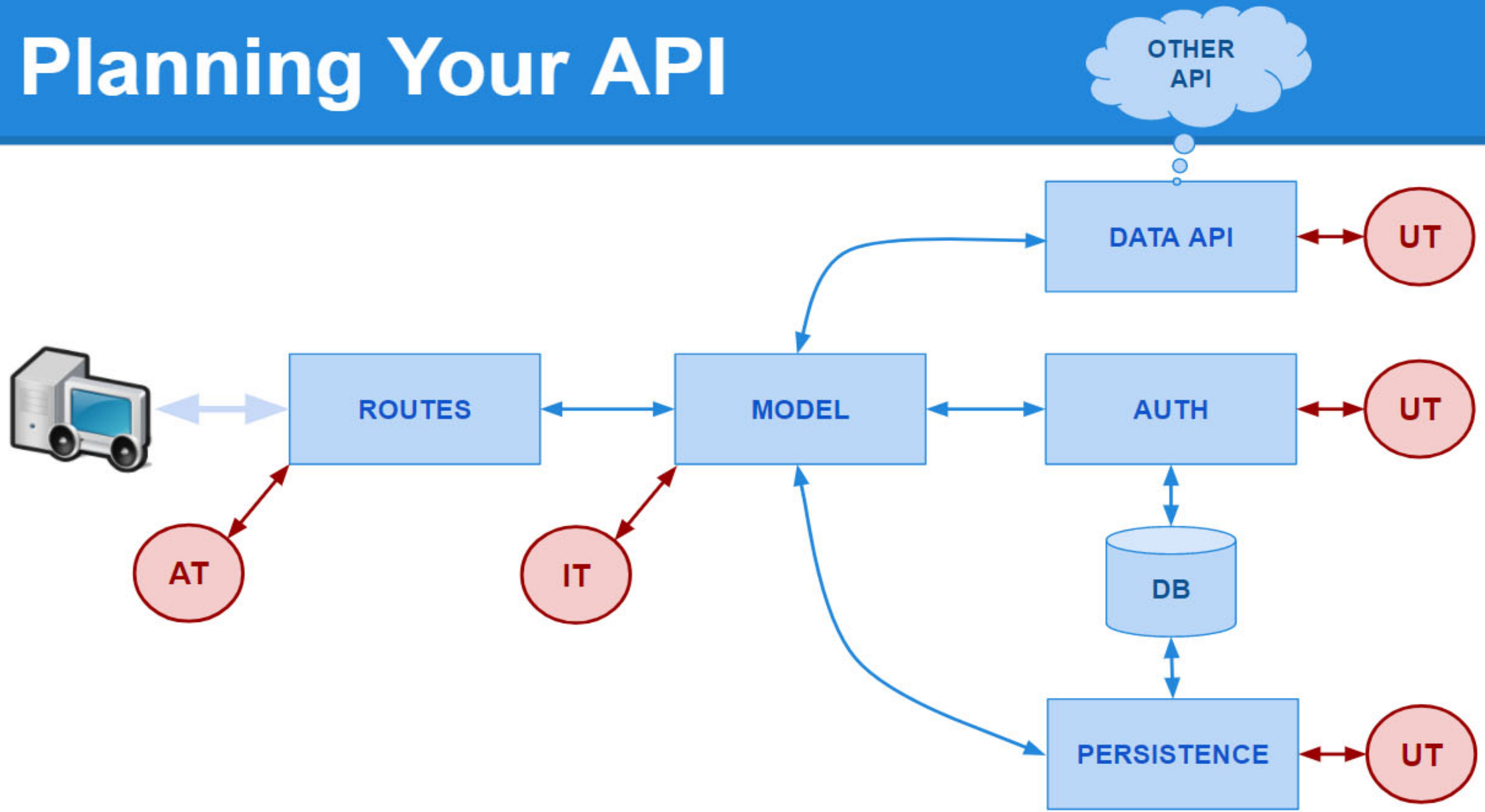
Example of JSON


```
{
 "records": 2,
 "books": [
   {
    "isbn": "9780340881538",
    "title": "The Recruit",
    "author": "Robert Muchamore"
   },
   {
    "isbn": "9780340881545",
    "title": "Cherub",
    "author": "Robert Muchamore"
   }
 ]
}
```

REST Service

- Platform-independent (you don't care if the server is Unix, the client is a Mac, or anything else),
- Language-independent (C# can talk to Java, etc.),
- Standards-based (runs on top of HTTP)
- Can easily be used in the presence of firewalls.

Planning Your API



Example of API planning

- Add a new user

http://example.com/api/v1.0/users/add

- Retrieve, update or delete user with id 10

http://example.com/api/v1.0/users/10

- Login user

http://example.com/api/v1.0/logins/login

- Retrieve password for user with Id 10

http://example.com/api/v1.0/logins/retrievepassword/10

What is Next

- First Lab
  - Starting with Bootstrap: Create Home Page for your Web application
- Second Lab
  - Understanding JavaScript Objects
  - Creating Data Persistence (SQL or noSQL)
- Lecture
  - Introduction to NodeJS