Web Application Development

Produced by

David Drohan (ddrohan@wit.ie)

Department of Computing & Mathematics
Waterford Institute of Technology

http://www.wit.ie

Waterford Institute *of* Technology
INSTITIÚID TEICNEOLAÍOCHTA PHORT LÁIRGE

Introduction to

RESTful Web Services & APIs

ACCESSIBLE WEB APPLICATION COMPONENTS

Agenda

- Background (How, What & Why)
- REST & APIs
- HTTP Methods
- RESTful Architecture Design

Background

A LITTLE BIT OF HOW, WHAT & WHY

A bit of History

According to one story, the term **Web Service** was first uttered by Bill Gates at a software conference in July of 2000. But the concept of two computers communicating with each other to share business information, and speaking in a shared vocabulary with a strongly defined set of protocols wasn't new. The concept had been around since before the internet was even invented. One of the earliest implementations of web service style architecture is known as EDI, or Electronic Data Interchange.

EDI was documented in 1996 by the National Institute Of Standards And Technology, as the computer to computer interchange of strictly formatted messages that represent documents other than monetary instruments.

A bit of History

In other words, EDI was designed to go beyond financial transactions and enable the movement of data back and forth over the internet. It was originally designed for large-scale e-commerce. But it was also used for all sorts of business transactions including invoicing and payments etc.

So, basically, a **Web Service** is any piece of software available on the web that uses **HTTP** and **XML** or **JSON** as a messaging system. This clearly implies that it is ***not bound*** to any specific operating system or programming language.

Hence, **Web Services** provide a standard means of inter-operation between software applications, irrespective of its running on different platforms and frameworks.

A bit of History

- The official definition of a Web Service is "a software system designed to support interoperable machine-to-machine interaction over a network." Breaking this down, the communication between servers requires an agreed-to (i.e., interoperable) format and structure*: I need to understand what you want, and you need to understand what I send you.*
- Web services have really come a long way since their inception. By 2002, the Web consortium had released the definition of WSDL and SOAP web services. This formed the standard of how web services were implemented.
- In 2004, the web consortium also released the definition of an additional standard called **RESTful**. Over the past number of years, this standard has become **very** popular. And is being used by many of the popular IT companies around the world which include Google, Facebook and Twitter, to name but a few.

Why Web Services ?

Exposing Existing Functionality over the network

- A web service is a unit of managed code that can be remotely invoked using HTTP, that is, it can be activated using HTTP requests. Web services allows you to expose the functionality of your existing code over the network. Once it is exposed on the network, other applications can use the functionality of your program.

Interoperability

- Web services allow various applications to talk to each other and share data and services among themselves. Other applications can also use the web services. For example, a VB or .NET application can talk to Java web services and vice versa. Web services are used to make the application platform and technology independent.

Why Web Services ?

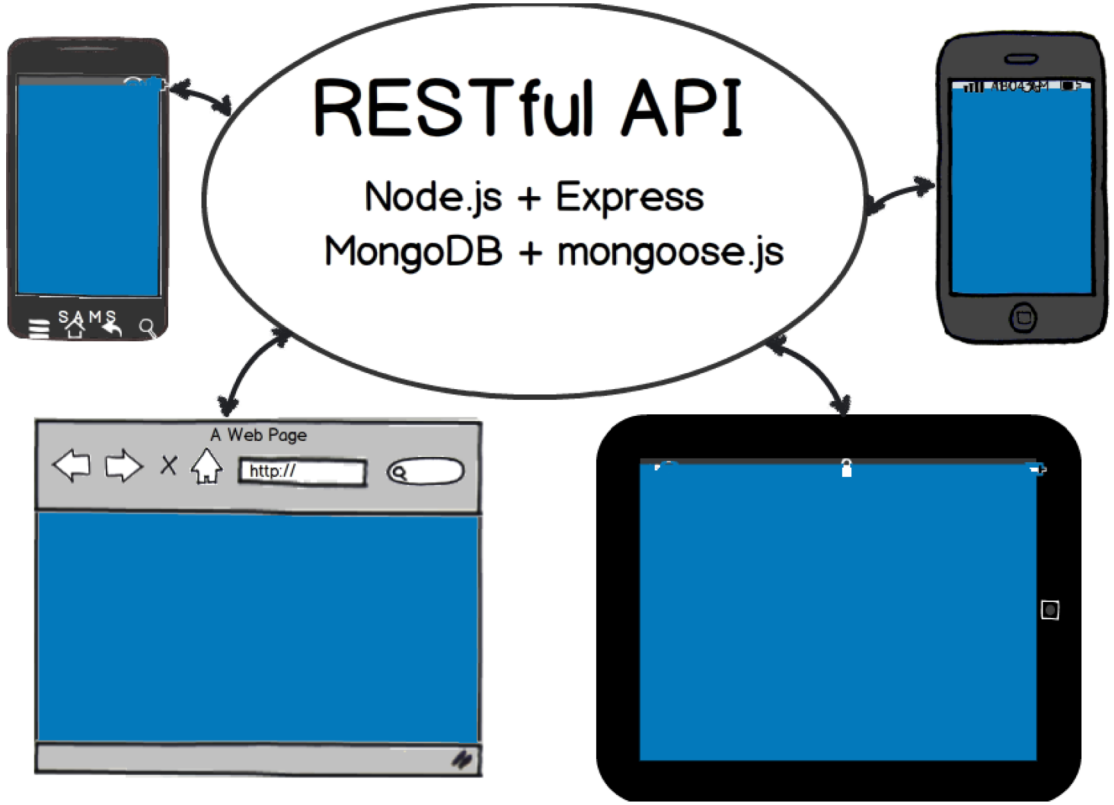
Standardized Protocol

- Web services use standardized industry standard protocol for the communication. All the four layers (Service Transport, XML Messaging, Service Description, and Service Discovery layers) use well-defined protocols in the web services protocol stack. This standardization of the protocol stack gives the business many advantages such as a wide range of choices, reduction in the cost due to competition, and increase in the quality.

Low Cost Communication

- Web services use SOAP or REST over HTTP protocol, so you can use your existing low-cost internet for implementing web services. This solution is much less costly compared to proprietary solutions like EDI/B2B. Web services can also be implemented on other reliable transport mechanisms like FTP.

REST

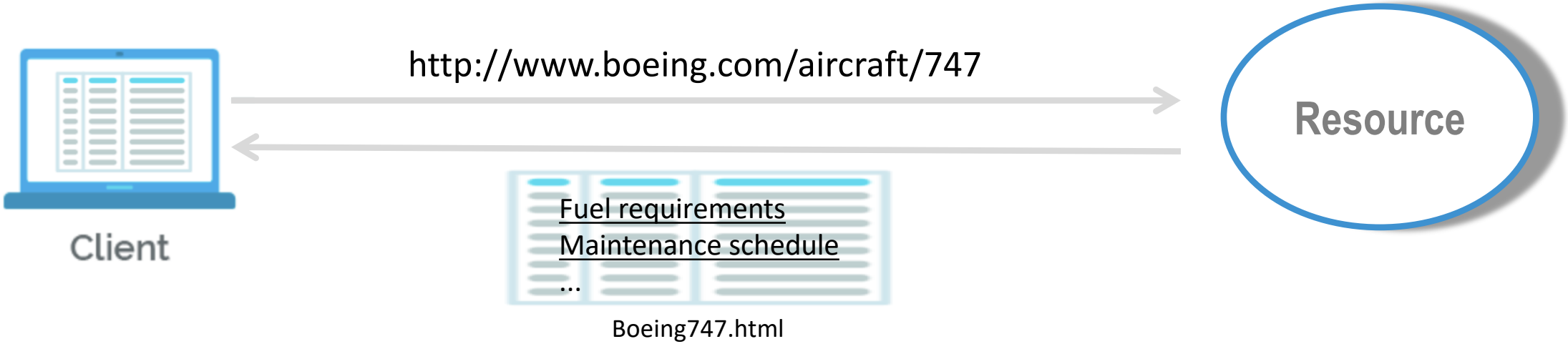


THE ARCHITECTURAL STYLE OF THE WEB

REST

- REST is a way to access resources which lie in a particular environment.
  - For example, you could have a server that could be hosting important documents or pictures or videos. All of these are an example of **resources**. If a client, say a web browser needs any of these resources, it has to send a request to the server to access these resources. Now REST defines a way on how these resources can be accessed.
- REST is used to build Web Services that are lightweight, maintainable, and scalable in nature. A service which is built on the REST architecture is called a **RESTful** service.
- The underlying protocol for REST is HTTP, which is the basic web protocol.
- REST stands for **REpresentational State Transfer**

Why is it called "**Re**presentational **S**tate **T**ransfer"?



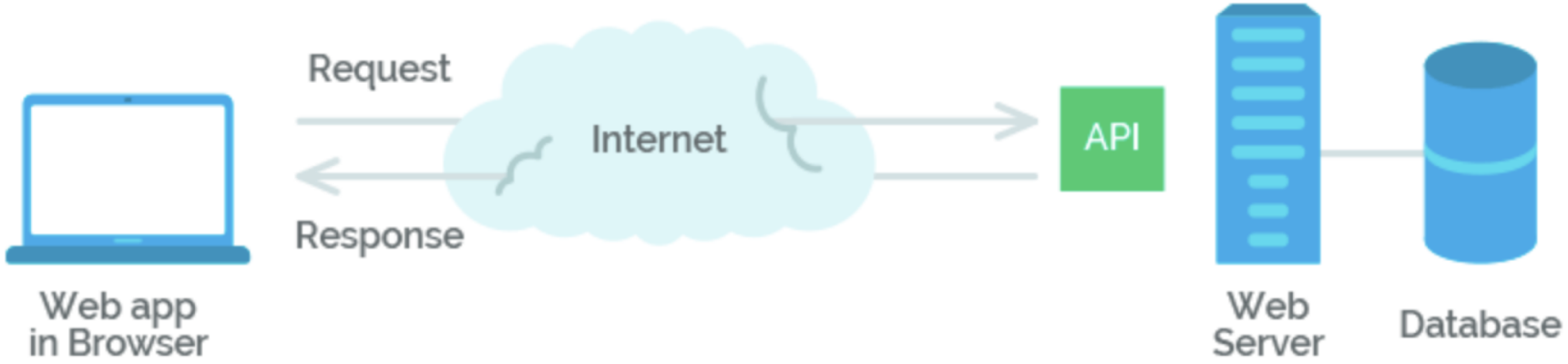
- The Client references a Web resource using a URL.
- A **representation** of the resource is returned (in this case as an HTML document).
- The representation (*e.g.,* Boeing747.html) places the client in a new **state**.
- When the client selects a hyperlink in Boeing747.html, it accesses another resource.
- The new representation places the client application into yet another state.
- Thus, the client application **transfers** state with each resource representation.

REST

- The REST model is an extension of how the World Wide Web itself works.
  - First, a request is made for a site via a URL call and the site is returned to the browser. Using the same HTML GET, POST and PUT requests, a RESTful Web Service returns the requested information using HTML or XML or JSON. This allows developers to put Web Service calls directly into web sites with no interface development to interpret the returning data. This is the theory behind mashup site development.
- Web Services have become the silent but critical backbone to our modern device-driven world. The common protocols, interfaces and communication standards that have evolved over the last 15 years or so allow us to develop mobile, tablet and PC-based applications that can order lunch, return stock prices or find the perfect gift rapidly, reliably and securely.

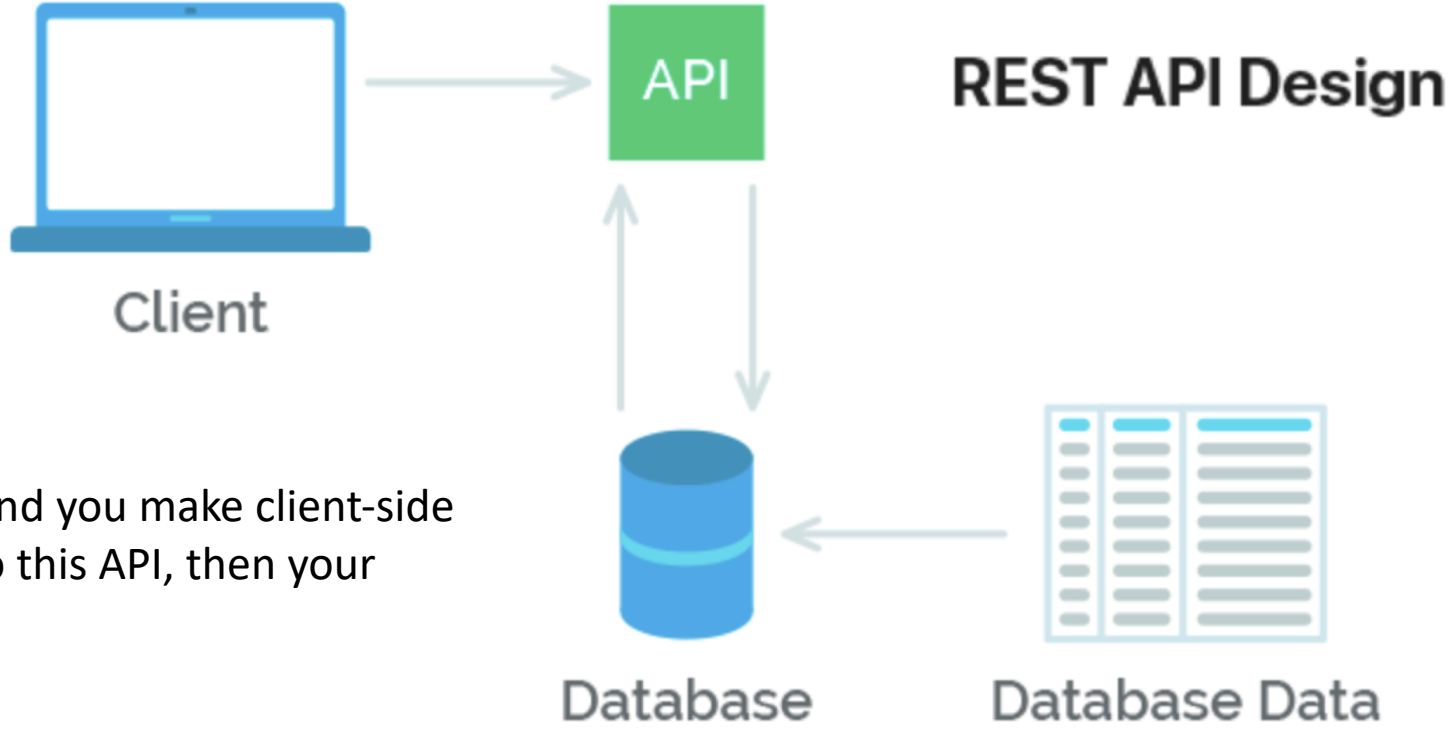
A bit about APIs

- An **API** (application programming interface) is a set of rules and mechanisms by which one application or component interacts with the others. It seems that the name speaks for itself, but let's get deeper into details. APIs can return data that you need for your application in a convenient format (e.g. JSON or XML). Here we will focus on JSON only.



REST & APIs

- REST is <u>not</u> a standard or protocol, it is an approach to, or architectural style for, writing APIs.



If your back-end server has a REST API and you make client-side requests (from a website/application) to this API, then your client is RESTful.

How it Works

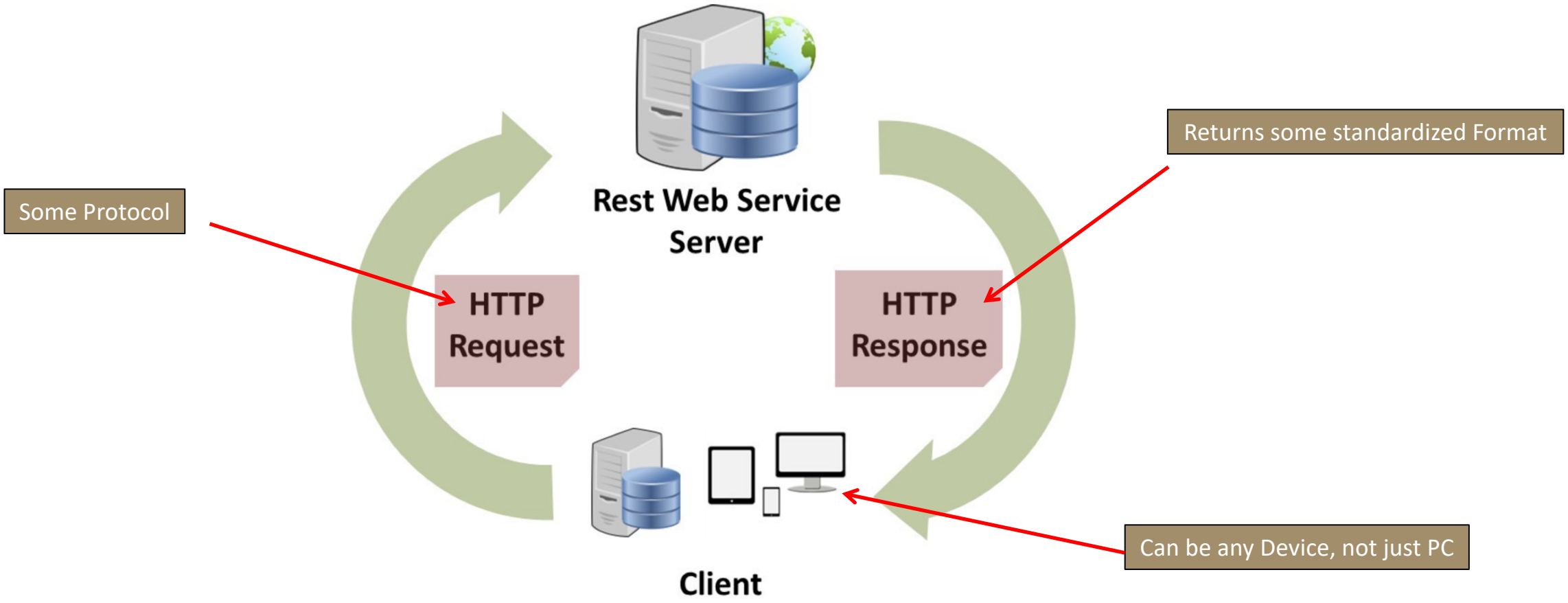
- RESTful API best practices come down to four essential operations:
  - receiving data in a convenient format;
  - creating new data;
  - updating data;
  - deleting data.
- REST relies heavily on HTTP and each operation uses its own HTTP method:
  - GET = "give me some info" (Retrieve)
  - POST = "here's some new info" (Create)
  - PUT = "here's some update info" (Update)
  - DELETE = "delete some info" (Delete)
- All these methods (operations) are generally called CRUD. They manage data or as Wikipedia says, "create, read, update and delete" it.

How it Works

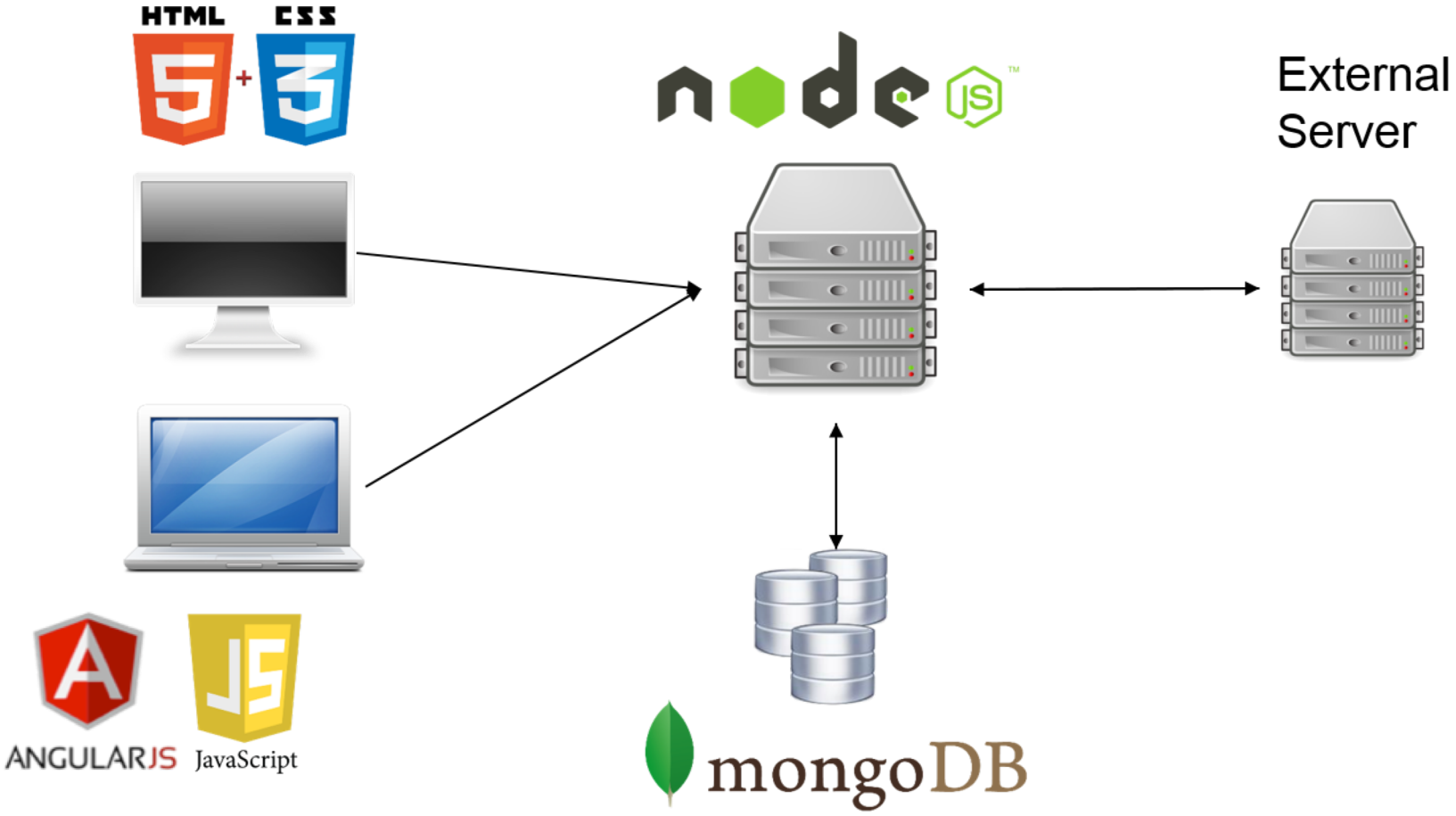
- The fact that REST contains a single common interface for requests and databases is its great advantage. This can be viewed in the table below.
- All requests you make have their own HTTP status codes.
  - 1xx - informational;
  - 2xx - success;
  - 3xx - redirection;
  - 4xx - client error;
  - 5xx - server error.

HTTP	POST	GET	PUT	DELETE
SQL	INSERT	SELECT	UPDATE	DELETE
CRUD	CREATE	READ	UPDATE	DELETE

How it Works


Some Protocol
Rest Web Service Server
HTTP Request
HTTP Response
Returns some standardized Format
Client
Can be any Device, not just PC

How ours will Work…



WEB BASICS

THE BASIC OPERATIONS OF THE WEB

RESTful architecture design

- Having a basic understanding of the different HTTP methods, or *verbs*, an API supports is very useful when **exploring and testing APIs**.
- As already mentioned, there are typically 4 operations used in RESTful APIs
  - GET
  - POST
  - PUT
  - DELETE
- All Resources in REST are separate entities, they can be both independent and dependent.

Web Basics: Retrieving Info using *HTTP GET*

`GET` requests are the most common and widely used methods in APIs and websites. Simply put, the GET method is used to **retreive data from a server at the specified resource**. For example, say you have an API with a `/users` endpoint. Making a GET request to that endpoint should return a list of all available users.

Since a GET request is only requesting data and not modifying any resources, it's considered a safe and idempotent method.

Testing an API with GET requests

When you're creating tests for an API, the `GET` method will likely be the most frequent type of request made by consumers of the service, so it's important to **check every known endpoint with a GET request**.

At a basic level, these things should be validated:

- Check that a valid GET request returns a `200` status code.
- Ensure that a GET request to a specific resource returns the correct data. For example, `GET /users` returns a list of users.

GET is often the **default method in HTTP clients**, so creating tests for these resources should be simple with any tool you choose.

Web Basics: Adding Info using *HTTP POST*

In web services, `POST` requests are used to **send data to the API sever** to create or udpate a resource. The data sent to the server is stored in the request body of the HTTP request.

The simplest example is a contact form on a website. When you fill out the inputs in a form and hit *Send*, that data is put in the **response body** of the request and sent to the server. This may be JSON, XML, or query parameters (there's plenty of other formats, but these are the most common).

It's worth noting that a `POST` request is **non-idempotent**. It mutates data on the backend server (by creating or updating a resource), as opposed to a `GET` request which does not change any data. Here is a great explanation of idempotentcy.

Testing an API with POST requests

The second most common HTTP method you'll encounter in your API tests is `POST`. As mentioned above, `POST` requests are used to **send data to the API server** and create or update a resource. Since POST requests modify data, it's important to **have API tests for all of your POST methods**.

Here are some tips for testing POST requests:

- Create a resource with a `POST` request and ensure a `200` status code is returned.
- Next, make a `GET` request for that resource, and ensure the data was saved correctly.
- Add tests that ensure `POST` requests **fail** with incorrect or ill-formatted data.

Web Basics: Updating Info using *HTTP PUT*

Simlar to POST, `PUT` requests are used to send data to the API to **create or update a resource**. The difference is that **PUT requests are idempotent**. That is, calling the same PUT request multiple times **will always produce the same result**. In contrast, calling a POST request repeatedly make have side effects of creating the same resource multiple times.

Testing an API with PUT requests

Testing an APIs `PUT` methods is very similar to testing POST requests. But now that we know the difference between the two (idempotency), we can **create API tests to confirm this behavior**.

Check for these things when testing PUT requests:

- Repeatedly cally a `PUT` request always returns the same result (idempotent).
- After updating a resource with a `PUT` request, a `GET` request for that resource should return the new data.
- `PUT` requests should fail if invalid data is supplied in the request – **nothing should be updated**.

Web Basics: Deleting Info using *HTTP DELETE*

The `DELETE` method is exactly as it sounds: **delete the resource at the specified URL**. This method is one of the more common in RESTful APIs so it's good to know how it works.

If a new user is created with a POST request to `/users`, and it can be retrieved with a `GET` request to `/users/{{userid}}`, then making a `DELETE` request to `/users/{{userid}}` will completely remove that user.

Testing an API with DELETE requests

`DELETE` requests should be heavily tested since they generally remove data from a database. Be careful when testing DELETE methods, make sure you're using the correct credentials and not testing with real user data.

A **typical test case for a DELETE request** would look like this:

1. Create a new user with a `POST` request to `/users`
2. With the user id returned from the `POST`, make a `DELETE` request to `/users/{{userid}}`
3. A subsequent `GET` request to `/users/{{userid}}` should return a 404 not found status code.

In addition, sending a DELETE request to an unknown resource should return a non-200 status code.

Web Apps vs. Web Services

Web Apps

- Return HTML
- Take GET or POST data as input (generally)
- Result intended for a human (via a browser)
- Informal (at best) description of data that resource accepts and result that resource returns

Web Services

- Return XML/JSON
- Result generally intended for a program
- Formal definition of data that resource accepts and result that resource returns

REST Principles

- Everything is a resource
- Every resource is identified by a unique identifier
- Use simple and uniform interfaces
- Communication is done by representation
- Be Stateless

- *We'll look at these, and more, in the later sections.*

References

- https://www.seguetech.com/web-services/
- https://mlsdev.com/blog/81-a-beginner-s-tutorial-for-understanding-restful-api
- https://assertible.com/blog/7-http-methods-every-web-developer-should-know-and-how-to-test-them
- https://restful.io/an-introduction-to-api-s-cee90581ca1b
- https://www.ntu.edu.sg/home/ehchua/programming/webprogramming/HTTP_Basics.html
- https://www.service-architecture.com/articles/web-services/web_services_explained.html
- https://www.guru99.com/restful-web-services.html
- https://medium.com/@sagar.mane006/understanding-rest-representational-state-transfer-85256b9424aa

References

"*REST* " was coined by Roy Fielding
in his Ph.D. dissertation [1] to describe a *design pattern* for implementing networked systems.

[1] http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm

Questions?