Module 7: API’s; Using an API; Building an API; Ajax; Web Application Architectures;

Edgardo Molina, PhD | Lead Instructor

CUNY Tech Prep 2016-2017

Outline

- API’s
    - What are they?
    - Using a Web API
    - Building a Web API
- What is Ajax?
- Web Application Architectures

API's

API’s

What is an API?

API's

What is an API?

An Application Programming Interface is a set of *1) subroutine definitions*, *2) protocols*, and *3) tools* for building software and applications.

API’s

What is an API?

An Application Programming Interface is a set of *1) subroutine definitions*, *2) protocols*, and *3) tools* for building software and applications.

Where do we find API’s?

API’s

What is an API?

An Application Programming Interface is a set of *1) subroutine definitions*, *2) protocols*, and *3) tools* for building software and applications.

Where do we find API’s?

- Operating Systems
- Software Libraries
- Databases
- **Web Applications** (*Web API’s*)
- etc...

What are the benefits of Web API's?

Our applications can interact with remote applications available over the web.

We can distribute our applications over the web to provide availability.

We can consume 3rd party services/applications we won't have to build.

Example API uses

To use 3rd Party Web Services

- Credit Cards Processing
    - `https://stripe.com/docs`
- Sending Email Newsletters
    - `http://developer.mailchimp.com/documentation/mailchimp/`
- Using Mapping Services
    - `https://developers.google.com/maps/`

Example API uses

Desktop and Mobile App Data Persistence

- Email clients
- Gaming (MMO)
- Dating Apps

Example API uses

Scripting Task Execution

- CI/CD and Git
- Provisioning Cloud services
- Backup and Logging services

Web Services

Web Services are implementation of API's that communicate over HTTP.

Most popular implementations follow RESTful design. Use of HTTP methods (GET, POST, PUT, PATCH, DELETE) when designing API routes.

Applications exchange data with Web Services using a common data format. The JSON format is popular with RESTful web services.

Data Exchange Formats: JSON and XML

Both JSON and XML are:

- Human and Machine readable data formats
- Allow for the exchange of structured data between applications and/or services

JSON

JavaScript Object Notation (JSON) is a simple data format based on JavaScript objects.

XML

eXtensible Markup Language (XML) is a data format that encloses data within tags similar to those of HTML.

JSON Example

```
{
  ”firstName”: ”John”,
  ”lastName”: ”Smith”,
  ”age”: 25,
  ”address”: {
    ”streetAddress”: ”21 2nd Street”,
    ”postalCode”: ”10021-3100”
  },
  ”phoneNumbers”: [
    {
      ”type”: ”home”,
      ”number”: ”212 555-1234”
    },
    {
      ”type”: ”office”,
      ”number”: ”646 555-4567”
    }
  ]
}
```

XML Example

```xml
<Person>
    <firstName>John</firstName>
    <lastName>Smith</lastName>
    <age>25</age>
    <address>
        <streetAddress>21 2nd Street</streetAddress>
        <postalCode>10021-3100</postalCode>
    </address>
    <phoneNumbers>
        <phoneNumber>
            <type>home</type>
            <number>212 555-1234</number>
        </phoneNumber>
        <phoneNumber>
            <type>office</type>
            <number>646 555-4567</number>
        </phoneNumber>
    </phoneNumbers>
</Person>
```

Other Data Exchange formats:

- CSV/TSV (Comma or Tab Delimeted files)
- BSON (`http://bsonspec.org/`)
- YAML (`http://www.yaml.org/`)
- Protocol Buffers (`https://developers.google.com/protocol-buffers/`)

These are also known as data interchange formats or as data serializers.

Other Protocols for remote API's

A Remote Procedure Call (RPC) is another protocol for calling subroutines on remote applications.

Various languages, platforms, and standard bodies (Java, Microsoft, CORBA, SOAP etc) have defined RPC implementations over the years.

Popular implementations in use today:

- Apache Thrift
- gRPC

Let’s use a RESTful API

Weather Service

Open Weather Map provides a web service for requesting the current weather for different cities.

`http://openweathermap.org/`

Look for the API Documentation.

`http://openweathermap.org/current`

You will need to sign up and apply for a Token to use the API.

Weather Service

Sample API responses:

`http://api.openweathermap.org/data/2.5/weather?zip=94040,us&appid=e810e54448d44835f1af45248aca8535`

`http://api.openweathermap.org/data/2.5/weather?zip=10016,us&appid=e810e54448d44835f1af45248aca8535&mode=xml`

`http://api.openweathermap.org/data/2.5/weather?zip=10016,us&appid=e810e54448d44835f1af45248aca8535&mode=html`

Using this Weather Service API

`http://jsfiddle.net/9hmx47oq/3/`

Let’s build an API

Developing a Web Service

You can develop Web Services in Express

Respond with JSON

```
res.json({
    firstName: ”John”,
    lastName: ”Doe”,
    age: 21,
    gender: ”male”
});
```

See example: `https://github.com/medgardo/ctp-lecture-code/tree/master/module-07/`

What is Ajax?

AJAX

Ajax stands for *Asynchronous JavaScript and XML*

- This is the mechanism web browsers provide client-side JavaScript for sending and receiving data to and from web servers
- The HTTP requests and responses happen in the background
- The web page in the browser does not reload or refresh
- Client-side JavaScript can then modify the page with data from the response

By using Ajax, web pages feel faster and interactive.

AJAX

Ajax Data Responses

Although XML is in the name, Ajax was designed to allow multiple datatypes: *Plain Text, HTML, XML, JS, and JSON*.

Vanilla JS vs jQuery Ajax comparison

`https://www.sitepoint.com/guide-vanilla-ajax-without-jquery/`

AJAX

The Web Broswer API (or vanilla JS Ajax)

The Web Browsers provides the XMLHttpRequest (XHR) object for making Ajax requests.

`https://en.wikipedia.org/wiki/XMLHttpRequest`

The jQuery API

jQuery like other frontend JavaScript frameworks provide simpler wrappers around the XMLHttpRequest object.

`http://api.jquery.com/jQuery.ajax/`

Vanilla JS Ajax Example

```
var xhr = new XMLHttpRequest();
xhr.open('GET', 'send-ajax-data.php');
xhr.send(null);


xhr.onreadystatechange = function () {
  var DONE = 4; // readyState 4 means the request is done.
  var OK = 200; // status 200 is a successful return.
  if (xhr.readyState === DONE) {
    if (xhr.status === OK)
      console.log(xhr.responseText); // 'This is the returned text.'
    } else {
      console.log('Error: ' + xhr.status); // An error occurred duri
    }
  }
};
```

jQuery Ajax Example

```
$.ajax({
  url: 'send-ajax-data.php',
  method: 'GET'
})
.done(function(res) {
  console.log(res);
})
.fail(function(err) {
  console.log('Error: ' + err.status);
});
```

This is the Promise based API

Web Application Architectures

Architecture Types

Monolithic Apps

SPA’s (Single Page Apps)

Microservices

Monolithic Apps

In a monolithic application:

- There is one codebase
- All features are added to the same codebase
- All developers work on the same codebase
- All backend, database, and frontend rendering is done in one app.

Monolithic Apps

Benefits

Monolithic Apps

Benefits

- Code is based around one core language and framework
    - Team builds **strong competency**
- Only one app to deploy and maintain
- Every developer can work across the front and back-end due to tighter coupling and single language required
- A small team can build a lot of features quickly
- Hiring and communication are easier

Monolithic Apps

Complications

Monolithic Apps

Complications

- Building and Testing takes a *long* time
- Simple updates and new features require a full app redeploy
- Harder for a large team to work on a single codebase
- Harder for any one developer to know the entire app
- Technical Debt
    - Fastest method to get up and running is not the best solution in the long term

Single Page Apps

In a SPA:

- The entire frontend code is sent to the user in a single HTML page load
- The users browser is responsible for all frontend code execution
- The SPA only contacts the server to *request* or *send* data
- The server only provides and consumes data in a lightweight format (JSON/XML)
- The SPA communicates “exclusively” through Ajax calls to the server

Why did SPA’s come about?

Why did SPA’s come about?

Users

Want web apps to feel *fast* and *responsive* like desktop apps

Long page loads or entire page refresh can cause customers to spend less or leave the web site

Developers

If we have to support mobile apps, then we have to build an API

The same API can be used by a SPA web frontend

Web frontend can be delegated to a separate team

Single Page Apps

Complications

Single Page Apps

Complications

- The SPA runs on the users computer
    - performance can vary widely
    - limited to users version/capabilities of browser/plugins
- Only programming language available is JavaScript
    - Other languages like TypeScript possible with transpilers
    - Transpilers means even more work for the users browser

Microservices

As the application *complexity* and development *team* grows, it is harder to work in a monolith.

SPA's show us we can split off the frontend. This is valuable if we are already supporting multiple teams, such as:

- Android Team: works in Java
- iOS Team: works in Objective-C/Swift
- SPA Team: works in JavaScript

Microservices

In the microservice architecture:

- The backend is split into smaller interoperating apps and/or API's
  - These are call microservices or webservices (ws)
- Each webservice can be built in completely different languages if desired
- Each webservice is tested, deployed, and maintained independently
- Each webservices complexity can be kept small and developed by a small team
- Webservices speak via their interface and data exchange format

Microservices

Examples

Microservices

Examples

- Companies with many products will turn Authentication into a Microservice
    - OAuth/OpenID (Who does it: Google, Facebook, etc)
    - *Why*: For single sign on convenience
- Companies also extract reusable components into webservices
    - Logging
    - Storage
    - Notifications
    - Discussion/comment boards

Microservices

Complications

Microservices

Complications

- Good Interfaces are *difficult* to design!!!
  - Requires a lot of planning
  - Major API changes will likely cascade to other microservices
- Choosing many diverse languages/frameworks will:
  - Make it harder to hire
  - Require more deployment and maintanence expertise by devops
- It's an *optimization* you may not need yet, or ever!
  - Optimizing ahead of time will delay time to market
  - Incur costs
  - You may never reach the scale required to make use of it

Monolith vs Microservices

Further reading

Articles on the subject:

- What are Microservices?
  - `http://martinfowler.com/articles/microservices.html`
- Costs of using Microservices?
  - `http://martinfowler.com/bliki/MicroservicePremium.html`
- Monoliths vs Microservices *(and in-between)*
  - `https://blog.codeship.com/monolithic-core-vs-fully-microservice-architecture/`