Development Frameworks & APIs

In addition to web servers that can host web applications in various languages, there are many common web development frameworks that help in developing core web application files and functionality. With the increased complexity of web applications, it may be challenging to create a modern and sophisticated web application from scratch. Hence, most of the popular web applications are developed using web frameworks.

As most web applications share common functionality -such as user registration-, web development frameworks make it easy to quickly implement this functionality and link them to the front end components, making a fully functional web application. Some of the most common web development frameworks include:

- `Laravel` (`PHP`): usually used by startups and smaller companies, as it is powerful yet easy to develop for.
- `Express` (`Node.JS`): used by `PayPal`, `Yahoo`, `Uber`, `IBM`, and `MySpace`.
- `Django` (`Python`): used by `Google`, `YouTube`, `Instagram`, `Mozilla`, and `Pinterest`.
- `Rails` (`Ruby`): used by `GitHub`, `Hulu`, `Twitch`, `Airbnb`, and even `Twitter` in the past.

It must be noted that popular websites usually utilize a variety of frameworks and web servers, rather than just one.

APIs

An important aspect of back end web application development is the use of Web APIs and HTTP Request parameters to connect the front end and the back end to be able to send data back and forth between front end and back end components and carry out various functions within the web application.

For the front end component to interact with the back end and ask for certain tasks to be carried out, they utilize them to ask the back end component for a specific task with specific input. The back end components process these requests, perform the necessary functions, and return a certain response to the front end components, which finally renderers the end user's output on the client-side.

Query Parameters

The default method of sending specific arguments to a web page is through `GET` and `POST` request parameters. This allows the front end components to specify values for certain parameters used within the page for the back end components to process them and respond accordingly.

For example, a `/search.php` page would take an `item` parameter, which may be used to specify the search item. Passing a parameter through a `GET` request is done through the URL '`/search.php?item=apples`', while `POST` parameters are passed through `POST` data at the bottom of the `POST HTTP` request:

Code: http

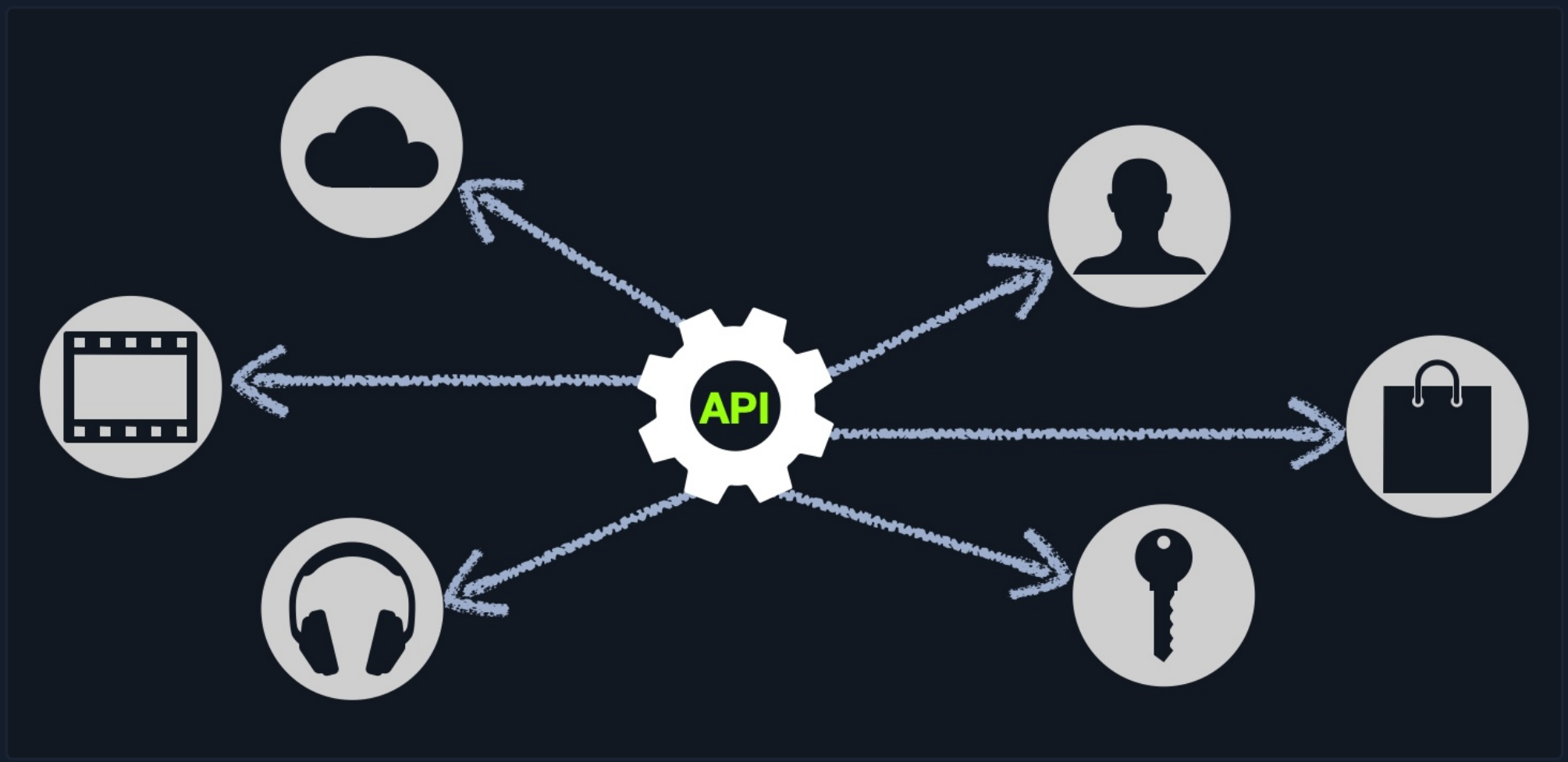
```http
POST /search.php HTTP/1.1
...SNIP...

item=apples
```

Query parameters allow a single page to receive various types of input, each of which can be processed differently. For certain other scenarios, Web APIs may be much quicker and more efficient to use. The Web Requests module takes a deeper dive into `HTTP` requests.

Web APIs

An API (Application Programming Interface) is an interface within an application that specifies how the application can interact with other applications. For Web Applications, it is what allows remote access to functionality on back end components. APIs are not exclusive to web applications and are used for software applications in general. Web APIs are usually accessed over the `HTTP` protocol and are usually handled and translated through web servers.



A weather web application, for example, may have a certain API to retrieve the current weather for a certain city. We can request the API URL and pass the city name or city id, and it would return the current weather in a `JSON` object. Another example is Twitter's API, which allows us to retrieve the latest Tweets from a certain account in `XML` or `JSON` formats, and even allows us to send a Tweet 'if authenticated', and so on.

To enable the use of APIs within a web application, the developers have to develop this functionality on the back end of the web application by using the API standards like `SOAP` or `REST`.

SOAP

The `SOAP` (Simple Objects Access) standard shares data through `XML`, where the request is made in `XML` through an HTTP request, and the response is also returned in `XML`. Front end components are designed to parse this `XML` output properly. The following is an example `SOAP` message:

Code: xml

```xml
<?xml version="1.0"?>

<soap:Envelope
xmlns:soap="http://www.example.com/soap/soap/"
soap:encodingStyle="http://www.w3.org/soap/soap-encoding">

<soap:Header>
</soap:Header>

<soap:Body>
  <soap:Fault>
  </soap:Fault>
</soap:Body>

</soap:Envelope>
```

`SOAP` is very useful for transferring structured data (i.e., an entire class object), or even binary data, and is often used with serialized objects, all of which enables sharing complex data between front end and back end components and parsing it properly. It is also very useful for sharing *stateful* objects -i.e., sharing/changing the current state of a web page-, which is becoming more common with modern web applications and mobile applications.

However, `SOAP` may be difficult to use for beginners or require long and complicated requests even for smaller queries, like basic `search` or `filter` queries. This is where the `REST` API standard is more useful.

REST

The `REST` (Representational State Transfer) standard shares data through the URL path 'i.e. `search/users/1`', and usually returns the output in `JSON` format 'i.e. userid `1`'.

Unlike Query Parameters, `REST` APIs usually focus on pages that expect one type of input passed directly through the URL path, without specifying its name or type. This is usually useful for queries like `search`, `sort`, or `filter`. This is why `REST` APIs usually break web application functionality into smaller APIs and utilize these smaller API requests to allow the web application to perform more advanced actions, making the web application more modular and scalable.

Responses to `REST` API requests are usually made in `JSON` format, and the front end components are then developed to handle this response and render it properly. Other output formats for `REST` include `XML`, `x-www-form-urlencoded`, or even raw data. As seen previously in the `database` section, the following is an example of a `JSON` response to the `GET /category/posts/` API request:

Code: json

```json
{
  "100001": {
    "date": "01-01-2021",
    "content": "Welcome to this web application."
  },
  "100002": {
    "date": "02-01-2021",
    "content": "This is the first post on this web app."
  },
  "100003": {
    "date": "02-01-2021",
    "content": "Reminder: Tomorrow is the ..."
  }
}
```

REST uses various HTTP methods to perform different actions on the web application:

- GET request to retrieve data
- POST request to create data
- PUT request to change existing data
- DELETE request to remove data

Start Instance

1 / 1 spawns left

Waiting to start...

Questions

Answer the question(s) below to complete this Section and earn cubes!

Target: Click here to spawn the target system!

+ 1 Use GET request '/index.php?id=0' to search for the name of the user with id number 1?

Submit your answer here...

Submit

Hint

Previous Next

Go to Questions

Table of Contents

Introduction to Web Applications

Introduction

Web Application Layout

Front End vs. Back End

Front End Components

HTML

Cascading Style Sheets (CSS)

JavaScript

Front End Vulnerabilities

Sensitive Data Exposure

HTML Injection

Cross-Site Scripting (XSS)

Cross-Site Request Forgery (CSRF)

Back End Components

Back End Servers

Web Servers

Databases

Development Frameworks & APIs

Back End Vulnerabilities

Common Web Vulnerabilities

Public Vulnerabilities

Next Steps

Next Steps

My Workstation

OFFLINE

Start Instance

1 / 1 spawns left