Evolve Full Stack Developer

APIs for Fun and Profit

Knock knock

Agenda

Web API Fundamentals

- History
- Definition
- HTTP Usage

REST

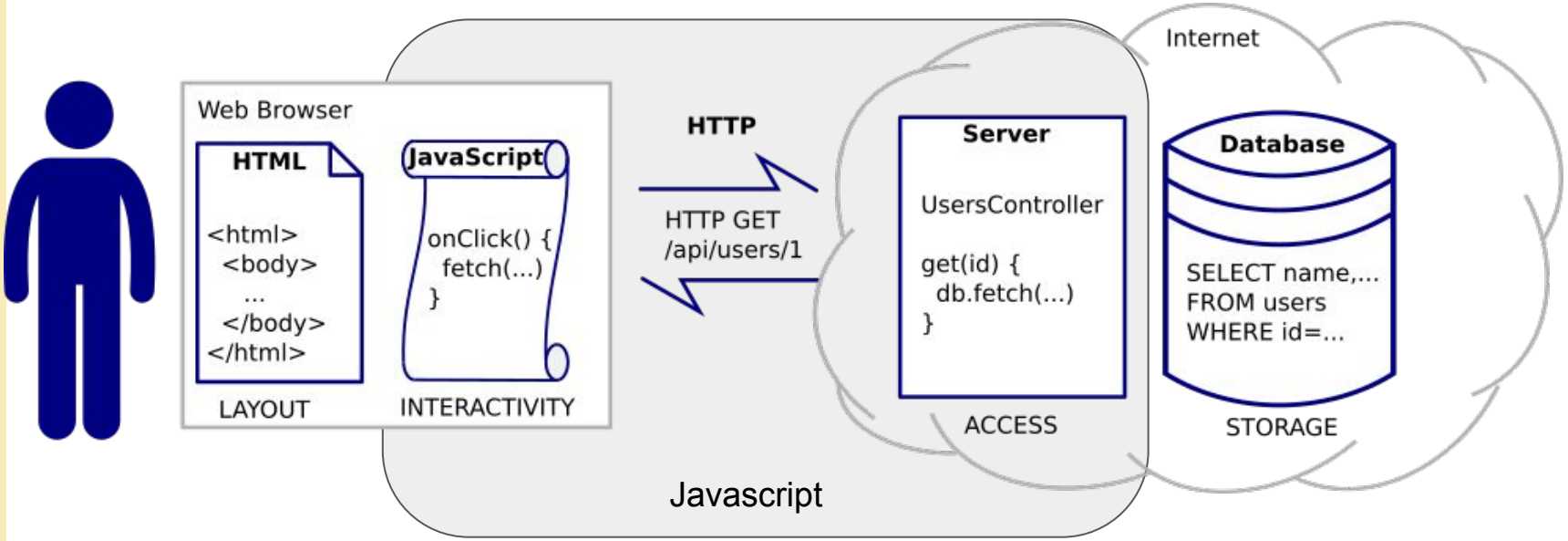
- Path Design

Authentication

- Basic
- Api Key
- Tokens

By the end of today you should be more comfortable designing and using external APIs.

Focus Area



INCEPTIONU

History of Web API

Back ... way back... web servers only served html pages and assets like images that supported them.

Slowly as more and more Javascript code ran on the front end the industry moved towards running code in the browser to render the pages. The code running in the browser would issue http requests to ask for json formatted data to populate the front end components.

It turns out that a pure “data” interface like this is also very useful to allow servers to call other servers looking for information.

Servers that offer data over http are said to support an **Application Programming Interface**, or API. It is an application programming interface because it is only really accessible to programs that are interested in data exchange without needing a user interface.

What is a Web API

An API is a collection of **http urls** that return **json formatted data**. Each url is called an **endpoint.**

The people hosting the API normally give you some documentation to work with that describe which paths to get from (or post to) and a description of the data you will receive (or send).

Any online service probably has an API, even if it is only to support their own front end.

Example:

https://www.twilio.com/docs/usage/api - an API to send SMS messages

HTTP URLs for APIs

Web APIs use normal HTTP requests the same as the browser does, so it is worth reviewing the parts of an URL as they relate to APIs.

https://api.chucknorris.io/jokes/RRE4ib6tR1W0uLZRuRvdDg

- Every URL has a "scheme/host" which is called the **base URL** that we use to identify the server on the internet and which protocol we are using to connect
- Every URL also has the "path" that identify which resource to fetch (or update)

When using an API you need to know how the paths are laid out to fetch data. Most APIs use id's in their paths to identify a single object.

HTTP Requests for APIs

Each HTTP request to an API is typically one of the following types of requests:

- GET - used to get data (like a web browser)
- POST - used to send data to the server (contains a body)
- DELETE

Often the same path can be used in multiple ways, when you GET “/user/1” you get a copy of their record, when you POST “/user/1” you try to update it, and if you DELETE “/user/1”...

Some Web APIs also make use of query parameters:

http://somehost.com/superhero?name=ironman

HTTP Headers / Response Codes for APIs

Common headers used in *API requests*:

Accept: used to tell the server what to send, use “application/json”

Authorization: used to prove who is requesting the service

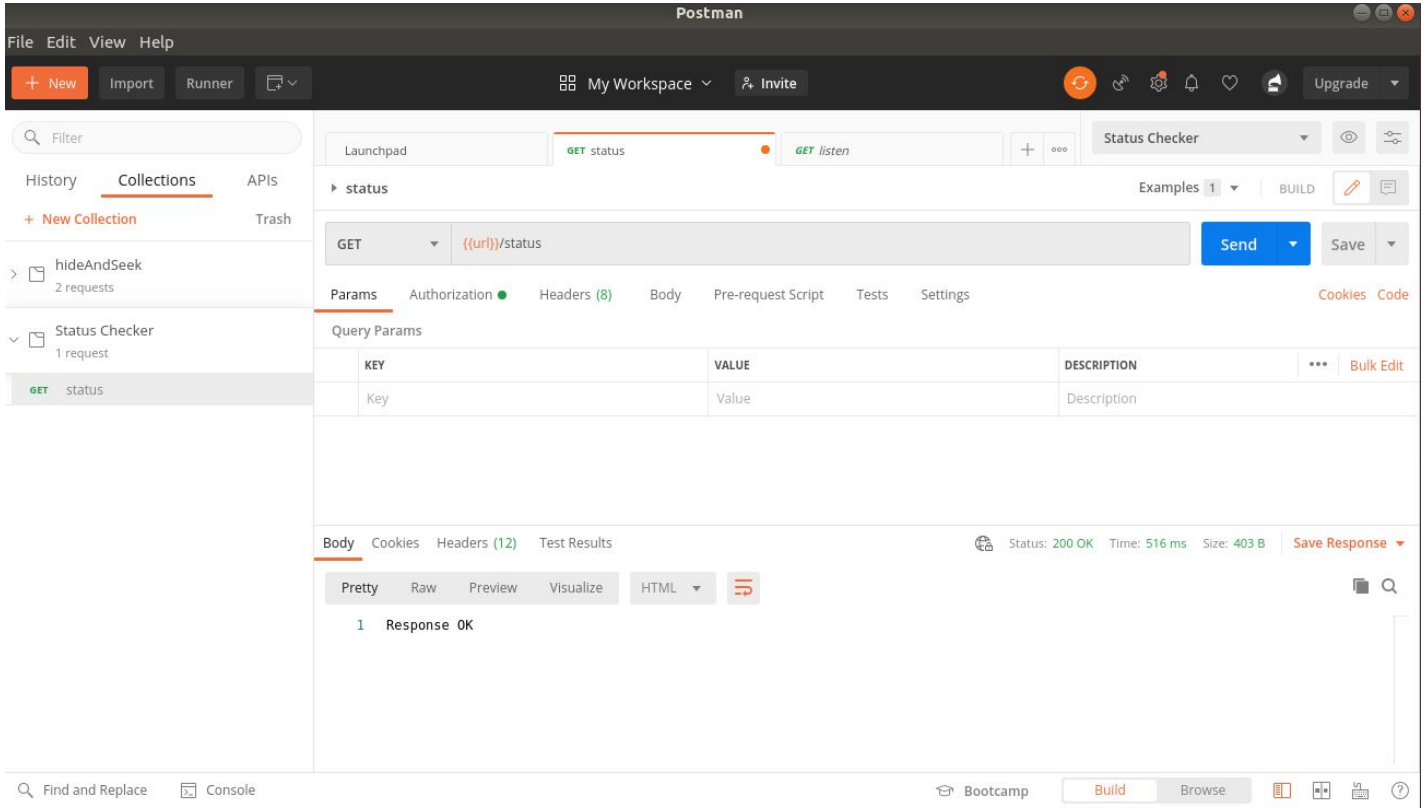
Content-Type: used with POST, use “application/json”

Common headers used in *API responses*:

WWW-Authenticate: used to ask for credentials

Status: 200 OK, 400 Bad Request, 401 Unauthorized, 403 Forbidden, 404 Not Found, 429 Too Many Requests

Testing APIs with Postman



INCEPTIONU

Exercise - find an API and call it

Start here

https://github.com/public-apis/public-apis

Look for APIs without Auth requirements

INCEPTIONU

Web API Design Pattern - REST

Many WebAPI adhere to the “REST” conventions for API design (and yours can too!)

Rest APIs typically look like this:

```
GET /users          returns all users
GET /users/1        returns user with id 1
POST /users         tries to create a new user
PUT /users/1        tries to update a user record by id
DELETE /users/1     tries to delete a user record by id
```

Web API Design Pattern - REST

REST APIs model the system as though it were a **database** of entities that describe the service. Implementation patterns that favor data driven design methods work really well with rest.

REST API calls are stateless, meaning each call contains everything it needs to complete. So each REST call should normally get the same response if you repeat the request (except delete).

Authentication

Most external services will require you to have an account and identify yourself when you try to use the service. Three kinds of authentication are common:

- Basic - using a username/password combination
- Api Key - a secret value you have been given by the service provider
- Bearer Token - used with Open Authentication standard

Authentication - Basic

Basic Authentication includes an encoded version of the username and password in the **Authorization** header of each request.

Always use with https (not that http exists much anymore).

Used when web applications are trying to identify a particular user who is logged in.

The browser will help you out by holding onto the username password combo or requesting it in an ugly dialog box.

Authentication - API Key

API Key authentication involves including a secret value that the service provider has given you in a “custom” header or as a query parameter of each request.

Always use with https.

Used when a server is trying to talk to another server and the identifying the original user isn’t important.

Authentication - Bearer Token

When your service is using a third party for authenticating users. For example when you “sign in with Google” on a non-google website, there is probably a bearer token involved.

The authenticating service creates a token that your web app must hold onto. It can be decrypted by code on your server (but is tamperproof on the client side).

Uses the **Authorization** header (and cookies by default)