7. APIs

If you’re waiting for lecture to start, try a `pip install requests`

Logistics ⚙️

- **Virtual Career Fair** next Wednesday, the 12th!
-

Recap: Frameworks

Which of the following is the best example of a web framework being opinionated?

1) Only working in a certain programming language
2) Only supporting a certain kind of user authentication
3) Only working when your code is free of bugs
4) Only running on Windows operating systems

Recap: Frameworks

Which of the following is the best example of a web framework being opinionated?

1) Only working in a certain programming language
2) Only supporting a certain kind of user authentication
3) Only working when your code is free of bugs
4) Only running on Windows operating systems

Recap: Flask

Which of these are (by default) legitimate places to store your HTML templates for a flask app?

1) A folder called “templates”, in the same directory as your Python app file.
2) A folder called “Templates”, in the same directory as your Python app file.
3) A folder called “template”, in the same directory as your Python app file.
4) All of the above

Recap: Flask

Which of these are (by default) legitimate places to store your HTML templates for a flask app?

1) A folder called “templates”, in the same directory as your Python app file.
2) A folder called “Templates”, in the same directory as your Python app file.
3) A folder called “template”, in the same directory as your Python app file.
4) All of the above

Recap: Static vs. dynamic web pages

CSS not updating? 🤦

- Your browser may be caching CSS and not downloading your new style sheets. You can confirm this is happening by going to “127.0.0.1:5000/<path_to_your_css_file>” or accessing browser developer tools.
- Ctrl-R or Cmd-R is a hard refresh in most browsers.
- You can also set a Flask environment variable to disable caching: `app.config['SEND_FILE_MAX_AGE_DEFAULT'] = 0`

Agenda 📅

1. Web APIs
    a. Practice with New York Times API
2. Git Review + .gitignore
    a. How to hide those pesky API keys!

Why do we care (about APIs)? 🤔

- Why do third-party APIs matter?
- Big companies like Facebook and Google don't use many external APIs, so why cover it?

Why do we care (about APIs)? 🤔

- As a software engineer…
  - Your team will need to determine if an API will be needed to solve a problem, or if another means such as building something from scratch is better.
- In industry…
  - Even though FB doesn't use too many third-party APIs, most of my job is spent using or building *internal* APIs!

API

Application Programming Interface

… is the set of rules that allows programs to talk to each other.

… gives users the powerful functionality of an existing application without having to rewrite it!

Web APIs 

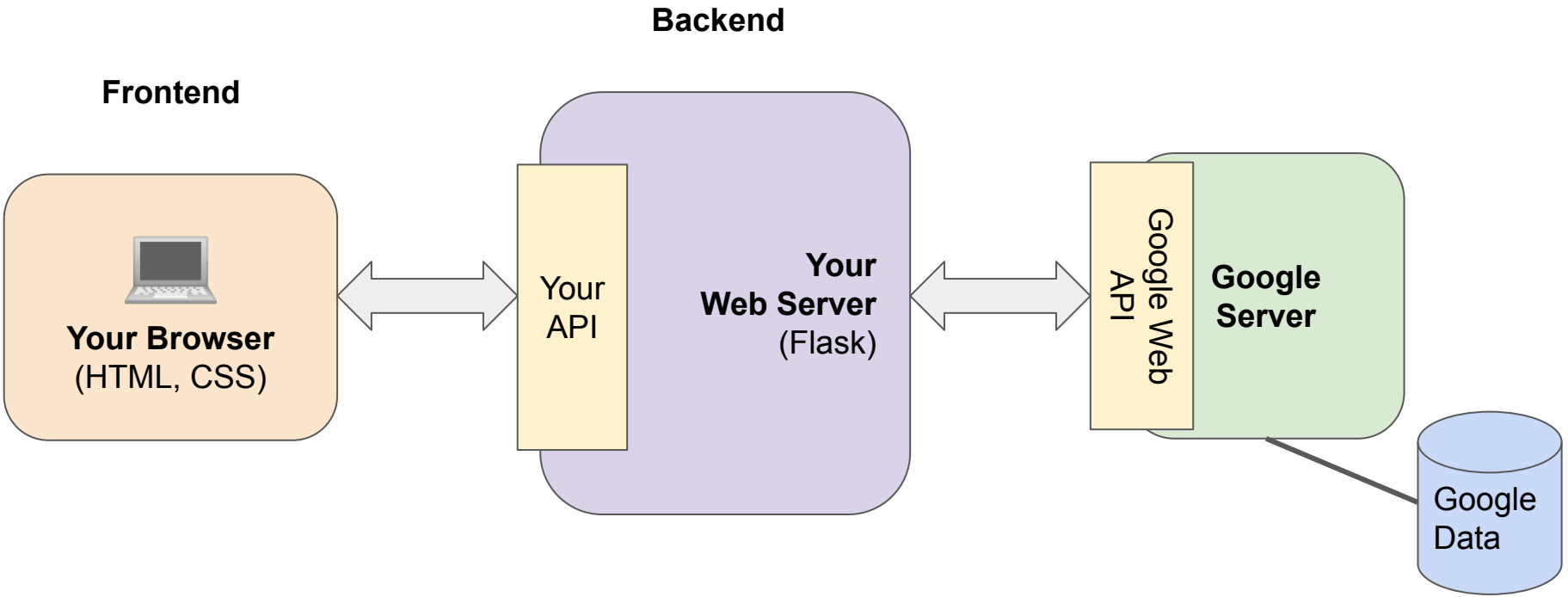
- For the most part, APIs in this class will reference web APIs
- A common goal for using many (web) APIs is to get rich data over the Internet! (Spotify's songs, YouTube's videos)

Web APIs and Software Engineering

- Modern software development is all about reusing others' work
  - We've already used frameworks (Flask) and services (Git, GitHub and soon, Heroku)!
- It is nearly impossible to build something completely from scratch
  - Whatever "from scratch" might mean to you, you can always go one level deeper
- You should strive to use others' work as much as possible in your careers
  - Avoid reinventing the wheel!

REST API 😴

- **REST** (**Re**presentational **S**tate **T**ransfer) is the rules that the API follows
- Client provides a URL (request)
- Server returns data (response)
- Often uses **HTTP** (**H**yper**t**ext **T**ransfer **P**rotocol) standards



Project Architecture

Anatomy of a Request 🦴

1. **Endpoint** - URL you request resource from
    a. Root endpoint - https://api.spotify.com/v1/
    b. Path - https://api.spotify.com/v1/artists/{id}/top-tracks
    c. Query parameter - https://api.spotify.com/v1/artists/{id}/top-tracks?market=US

Anatomy of a Request 🦴

1. Endpoint - URL you request resource from
    a. Root endpoint - https://api.spotify.com/v1/
    b. Path - https://api.spotify.com/v1/artists/{id}/top-tracks
    c. Query parameter - https://api.spotify.com/v1/artists/{id}/top-tracks?market=US
2. **Method** - HTTP type of request
    a. GET - read operation (get a resource/data)
    b. POST - write operation (create a new resource); in the case of Spotify, you may need this to create your access token for fetching data
    c. PUT
    d. PATCH
    e. DELETE

Anatomy of a Request 🦴

1. Endpoint - URL you request resource from
    a. Root endpoint - https://api.spotify.com/v1/
    b. Path - https://api.spotify.com/v1/artists/{id}/top-tracks
    c. Query parameter - https://api.spotify.com/v1/artists/{id}/top-tracks?market=US
2. Method - HTTP type of request
    a. GET - read operation (get a resource/data)
    b. POST - write operation (create a new resource); in the case of Spotify, you may need this to create your access token for fetching data
    c. PUT
    d. PATCH
    e. DELETE
3. **Headers** - information for things like authentication
4. **Data** - information you want to send to server

Calling a Web API endpoint 📞

- Terminal: use curl
  - Documentation will often show a curl command
  - curl -X "GET" "https://api.spotify.com/v1/albums/0GaMO6MLhbmtAFnKmKrdXB*?market=US*" -H "Accept: application/json" -H "Content-Type: application/json" -H "Authorization: Bearer {insert token here}"
- Python: use requests
  - requests.get("https://api.spotify.com/v1/albums/0GaMO6MLhbmtAFnKmKrdXB", headers={'Authorization': 'Bearer {token}'}, params={'market': 'US'})
- Use the API documentation console!

```
import requests

requests.get("https://api.nytimes.com", params={"q": "keyword"})
```

JSON (**J**ava**S**cript **O**bject **N**otation)

- Common string format to send and request data through REST API
- Convert HTTP response in Python with response.json()
    - Further parse with json.loads() (using the ‘json’ library)

```
{
  "album_type": "album",
  "artists": [
    {
      "href": "https://api.spotify.com/v1/artists/2BTZIqw0ntH9MvilQ3ewNY",
      "id": "2BTZIqw0ntH9MvilQ3ewNY",
      "name": "Cyndi Lauper",
            ...
    }
  ],
...
```

```
data["artists"][0]["name"]
```

Let’s make JSON pretty… 😍

```
>>> import json
>>>
>>> your_json = '["foo", {"bar":["baz", null, 1.0, 2]}]'
>>> parsed = json.loads(your_json)
>>> print(json.dumps(parsed, indent=4, sort_keys=True))
[
    "foo",
    {
        "bar": [
            "baz",
            null,
            1.0,
            2
        ]
    }
]
```

Authorization 🔐

- API **keys** and Auth **tokens** are like a user's passwords to call an API - each user has a different set of keys/tokens that authorize them to use the API.
- **Bonus: OAuth2** is the standard for authorization flow, although you may not use it in this class
- Client credentials - in our basic fetches, we don’t need to authorize the **user,** we just want to authorize the **client**
    - In this case, you use your special API key(s) to fetch a token for your client. That token then will authorize your client to make further requests.

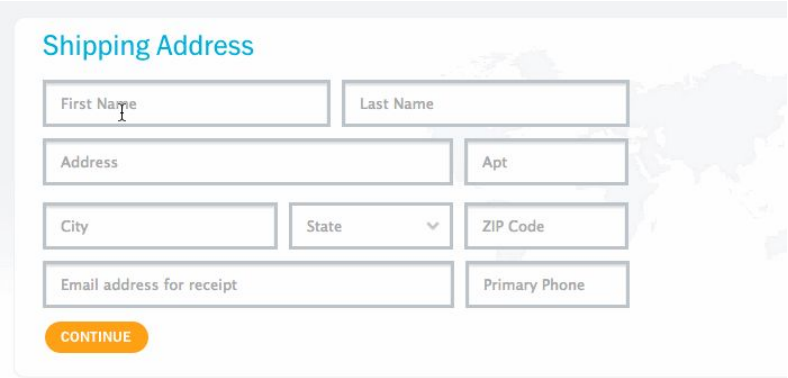
It’s demo time! 

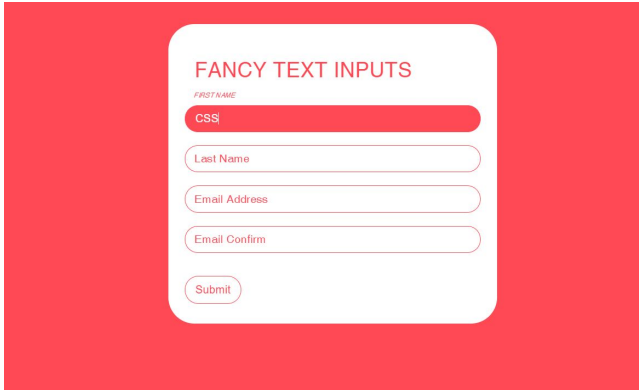
New York Times API

What about this other API I have to use??

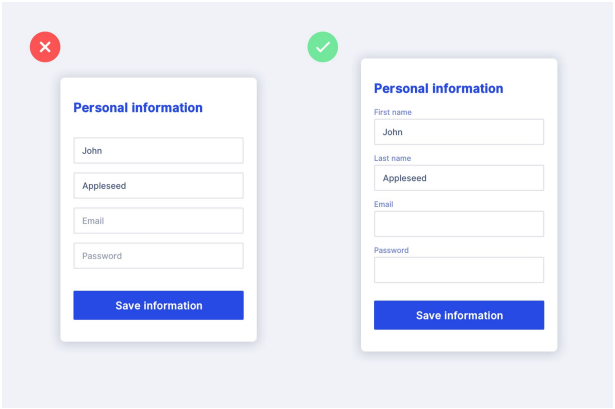
- Each API is different than any other API. So you're going to end up having to do a lot of independent research and debugging to figure out how each API works. **Documentation is key!**
- But they all generally have one thing in common: the use of API keys!
- I'm not going to be going over project APIs in class, but Google is your best friend to find examples!

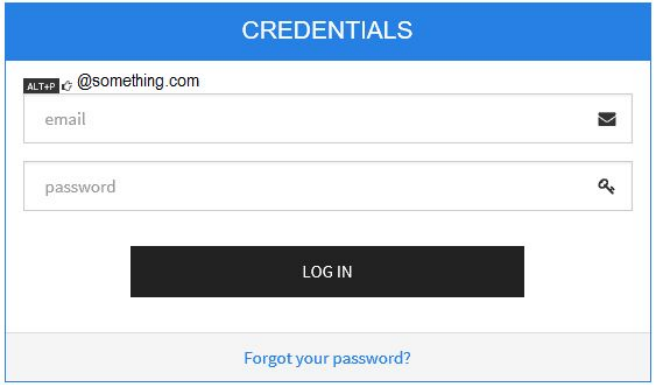
Break


Shipping Address
First Name
Last Name
Address
Apt
City
State
ZIP Code
Email address for receipt
Primary Phone
CONTINUE


FANCY TEXT INPUTS
FIRST NAME
CSS
Last Name
Email Address
Email Confirm
Submit

HTML Forms


Personal information
John
Appleseed
Email
Password
Save information
Personal information
First name
John
Last name
Appleseed
Email
Password
Save information


CREDENTIALS
ALT+P @something.com
email
password
LOG IN
Forgot your password?

HTML Forms

- Two new element types: <form> and <input>
- <form> should usually have two attributes: “method” and “action”
  - “Method” specifies which kind of HTTP request will be sent out when the form is submitted, e.g. “GET” or “POST”
  - “Action” specifies which route the input data will be sent to (e.g. “/index”, “/login”)
- <input> can have a lot of different types -- see the HTML docs!
  - We’re mostly going to focus on <input type=”text”> and input <type=”submit”>

```
<form method="POST" action="/login">
    <input type="text" name="username" placeholder="Your Username">
    <input type="submit" value="Submit">
</form>
```

HTML Forms - Receiving Data

- There are a lot of new Flask concepts here. We'll cover them in detail in a week.
- For now, you just should be aware that we'll need some server-side logic to process form data.

```
from flask import request
@app.route("/login", methods=["POST"])
def login():
    data = request.form
    username = data.get("username")
```

Until next time… ✌🏽

- Do HW7 before next class
- Next class, we will dive deep on different roles in tech!