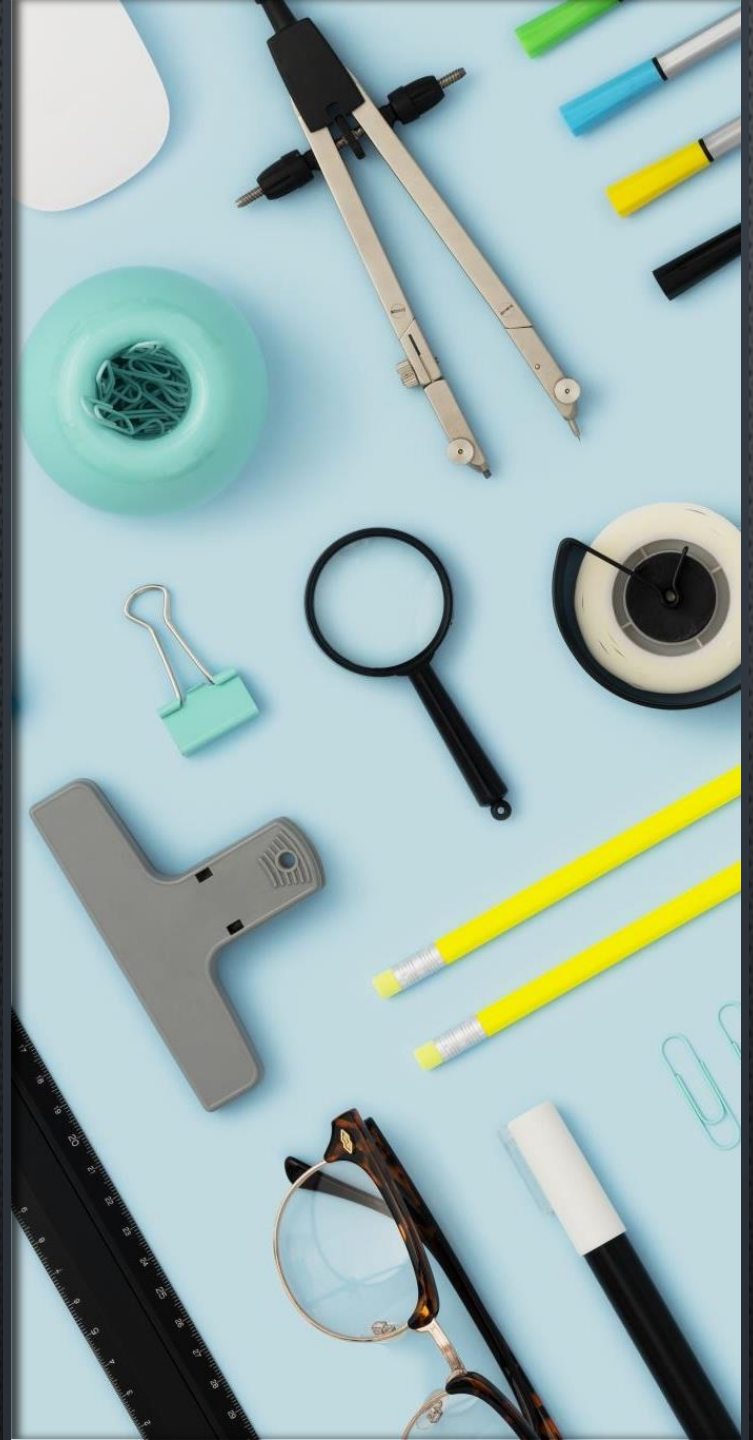
WELCOME TO CSCI E-33A WEB PROGRAMMING W/ PYTHON AND JAVASCRIPT SECTION 7

TF FOR THIS SECTION: GLENN LANGDON

LANGDON@CS50.NET



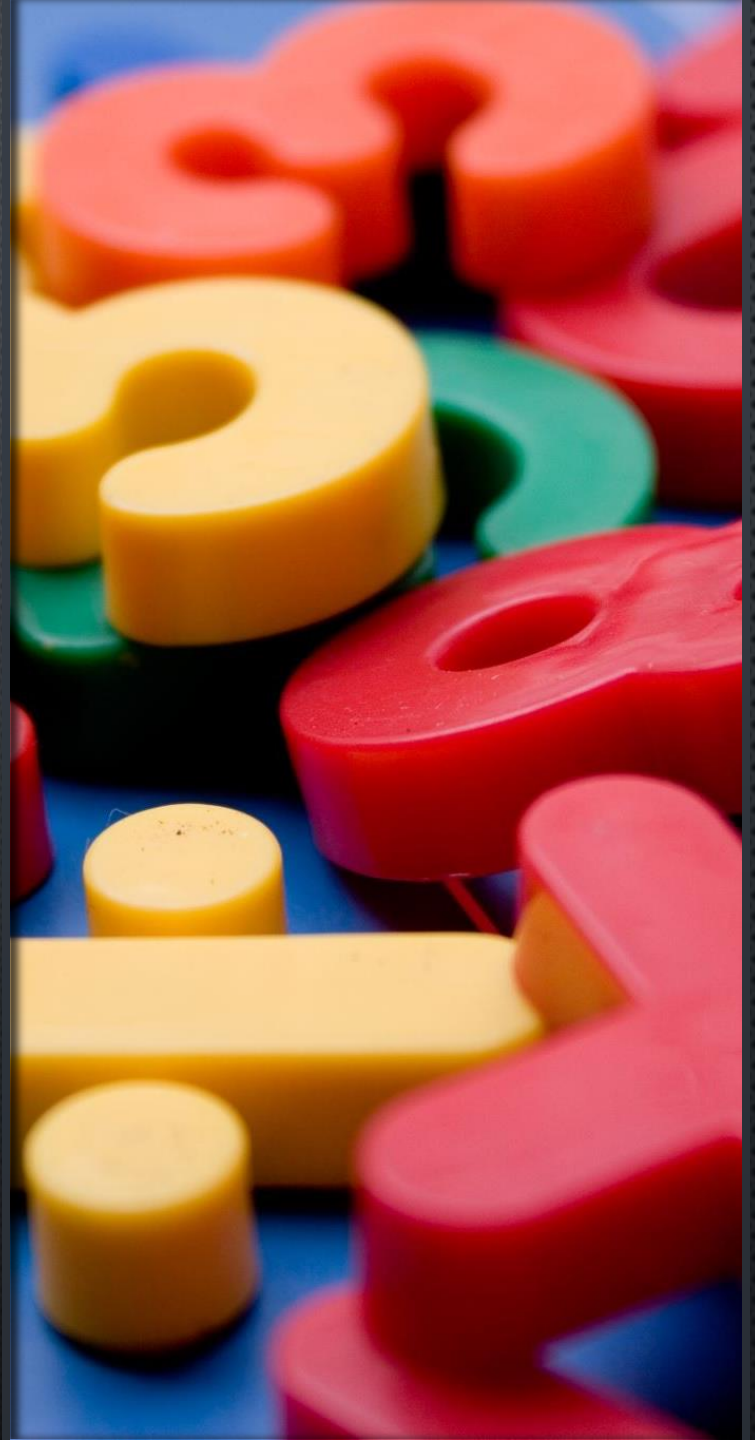
Program for today

- What's left for the semester
  - Proposal due: Sunday, Nov 23, 11:59 EST
  - Status report due: Sunday, Dec. 7, 11:59 EST
  - Final Project due: Sunday, Dec. 14, 11:59 EST
  - No late submissions
- Custom Django commands
- Django/React integration

QUESTIONS?

FREEBIES FOR HARVARD STUDENTS

- GITHUB STUDENT DEVELOPER PACK
  - HTTPS://EDUCATION.GITHUB.COM/PACK

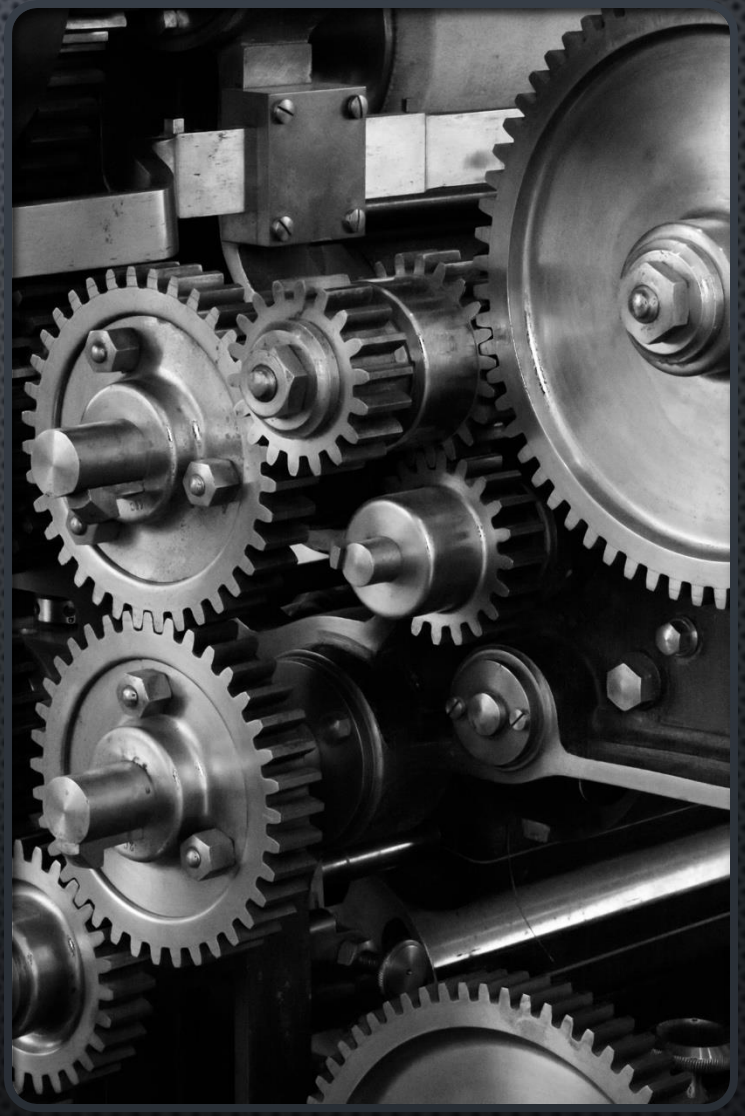


CUSTOM DJANGO MANAGEMENT COMMANDS

- DID YOU KNOW YOU CAN WRITE YOUR OWN MANAGE.PY COMMANDS?
- YOU CAN EXTEND FUNCTIONALITY AND AUTOMATE TASKS SPECIFIC TO YOUR PROJECT
  - AUTOMATION
  - SCHEDULED TASKS
  - DATA MANAGEMENT
  - … AND MORE

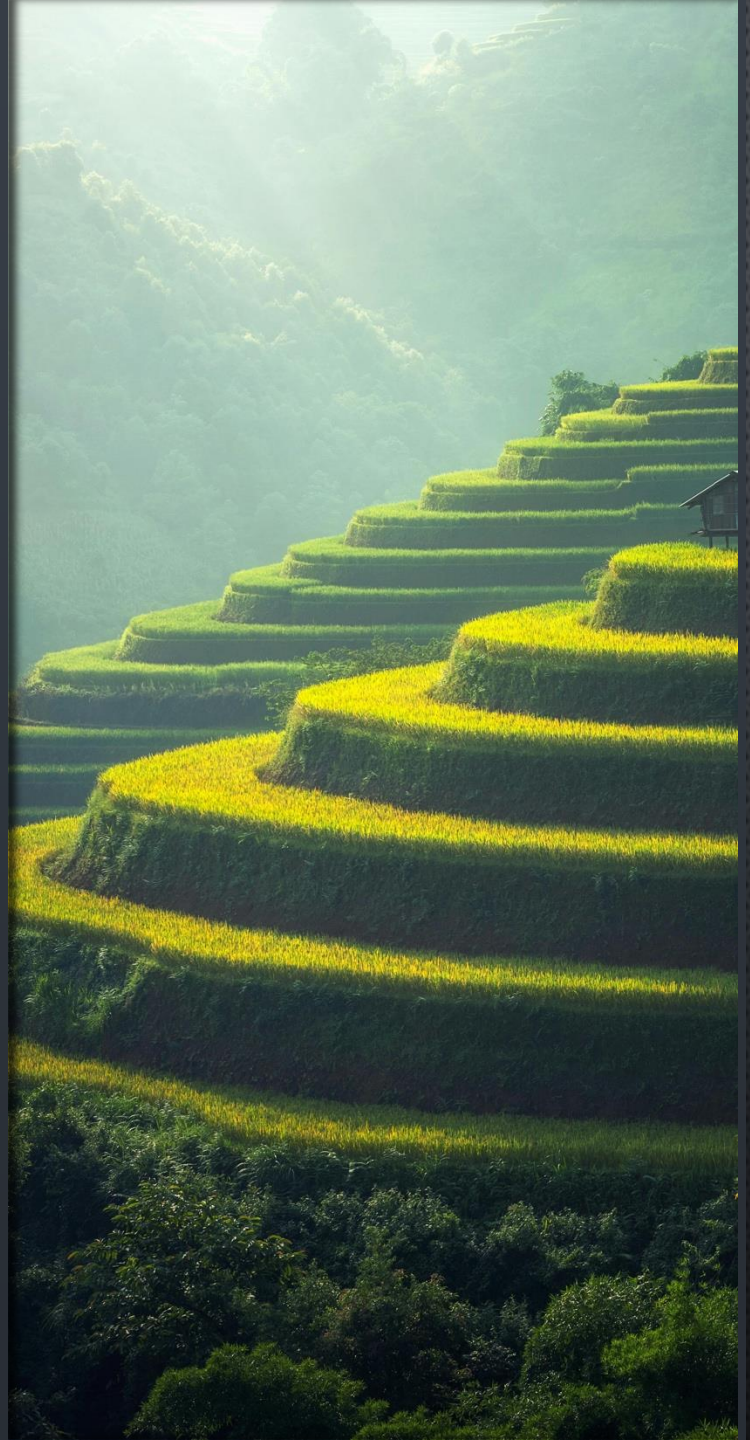
Data Management: Automate API Downloads

- What if you want to populate your database from an external API?
- There are many ways to accomplish this, but with a custom command you can do it all within Django.



HERE'S A DEMO APP THAT POPULATES A DJANGO DATABASE FROM AN EXTERNAL API

WHAT'S THE BEST WAY TO INTEGRATE A JS FRONTEND WITH DJANGO?



THE JAVASCRIPT LANDSCAPE

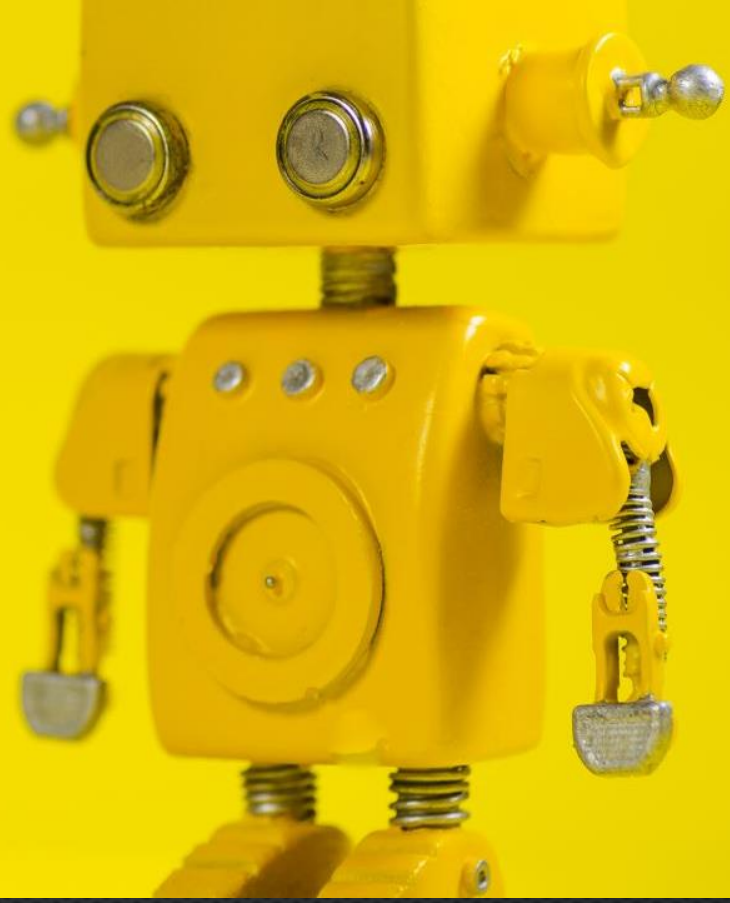
- Django is a fantastic batteries included web framework for rapid deployment of web apps
- JS is EVERYWHERE and is indispensable in web development
- JS was written in 10 days in 1995 by Brendan Eich and its idiosyncrasies are both its strength and weakness
- Plus, there are innumerable libraries/frameworks from which to choose
- As a back-end developer what is the best thing to do?

WELL, YOU CAN DO WHAT WE'VE DONE (VANILLA JS)

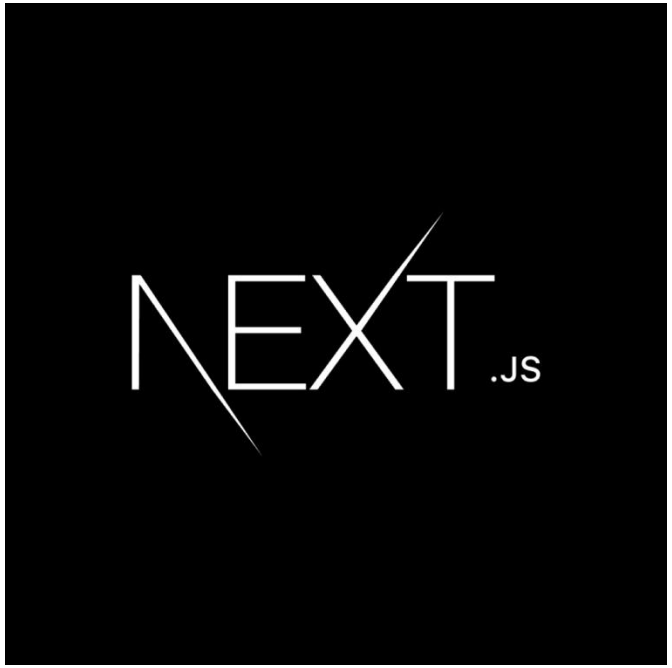
ECMAScript 2023 (ES14)

Imperative programming

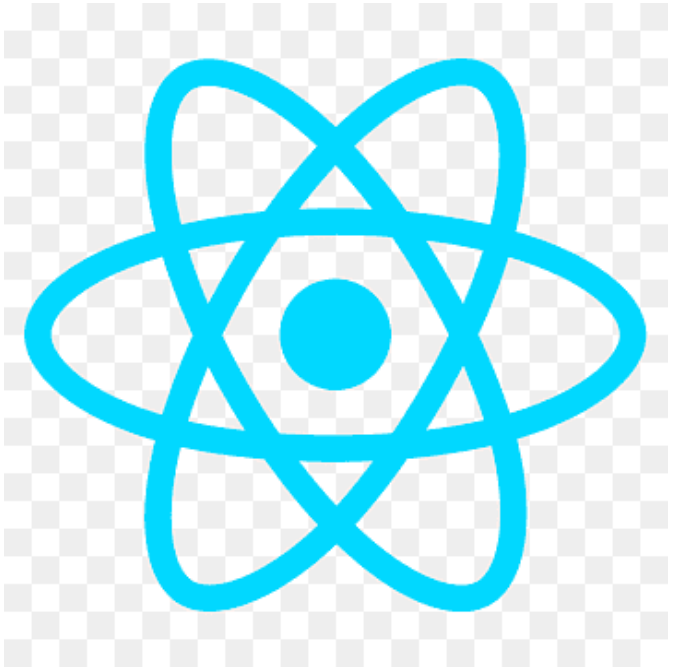
Complex apps can end up having code spread



WHAT KIND OF FRAMEWORKS ARE OUT THERE?



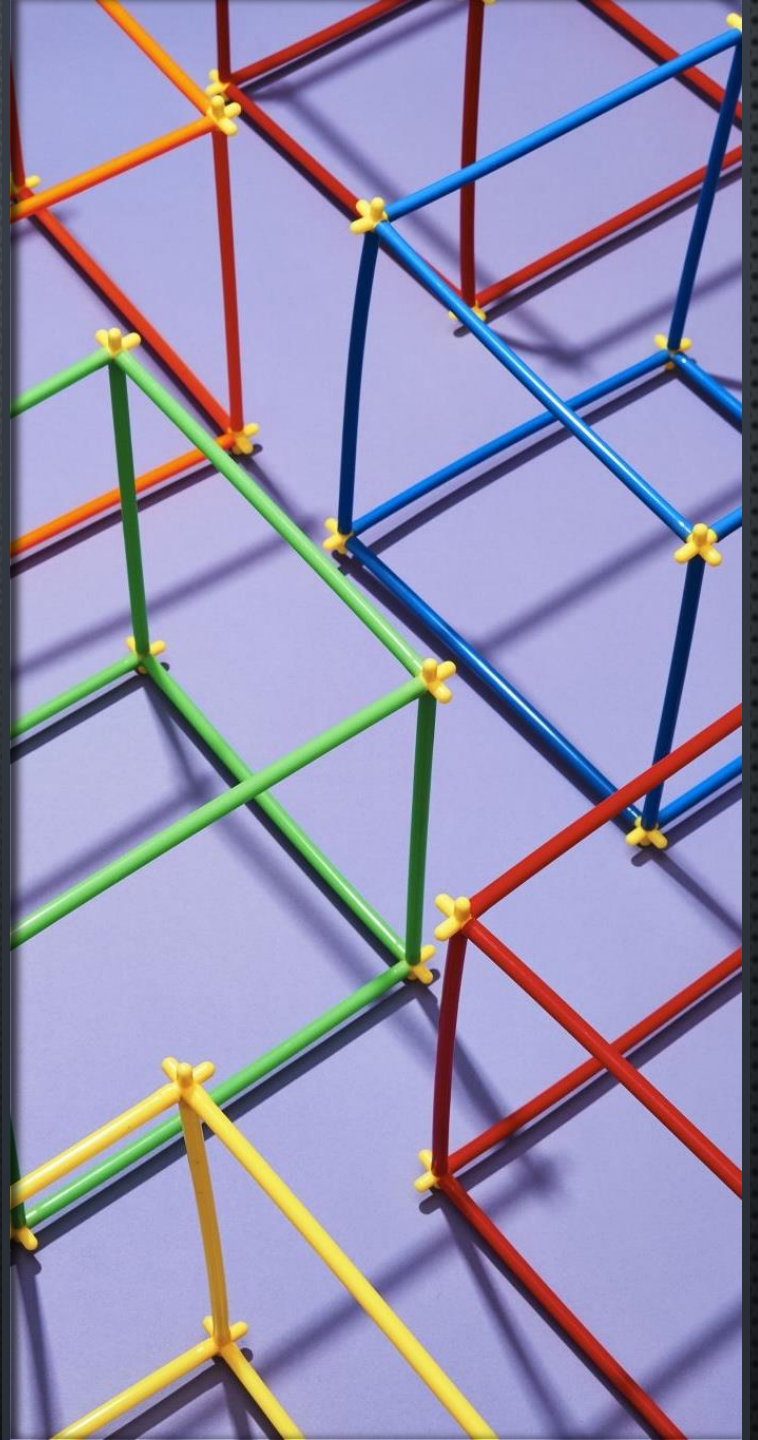
SERVER-FIRST



CLIENT-FIRST

Server-first considerations

- Chosen by backend developers, someone familiar with the framework
- Add JS to the framework to make front-end more interactive with less page-loading
- As app gets bigger, more models, pages with more complex state for JS
- Great at start but beginning to get more complicated as the app grows in sophistication
- So many choices in the JS ecosystem to choose from, it can be paralyzing to know which one is right for a given project



CLIENT-FIRST CONSIDERATIONS

- Usually chosen by front-end developers
- Not able to leverage a lot of Django's goodies
- A different paradigm for Django developers
- Complex development, build and deployment setup
- Backend is completely separate
- Can have two separate code depositories

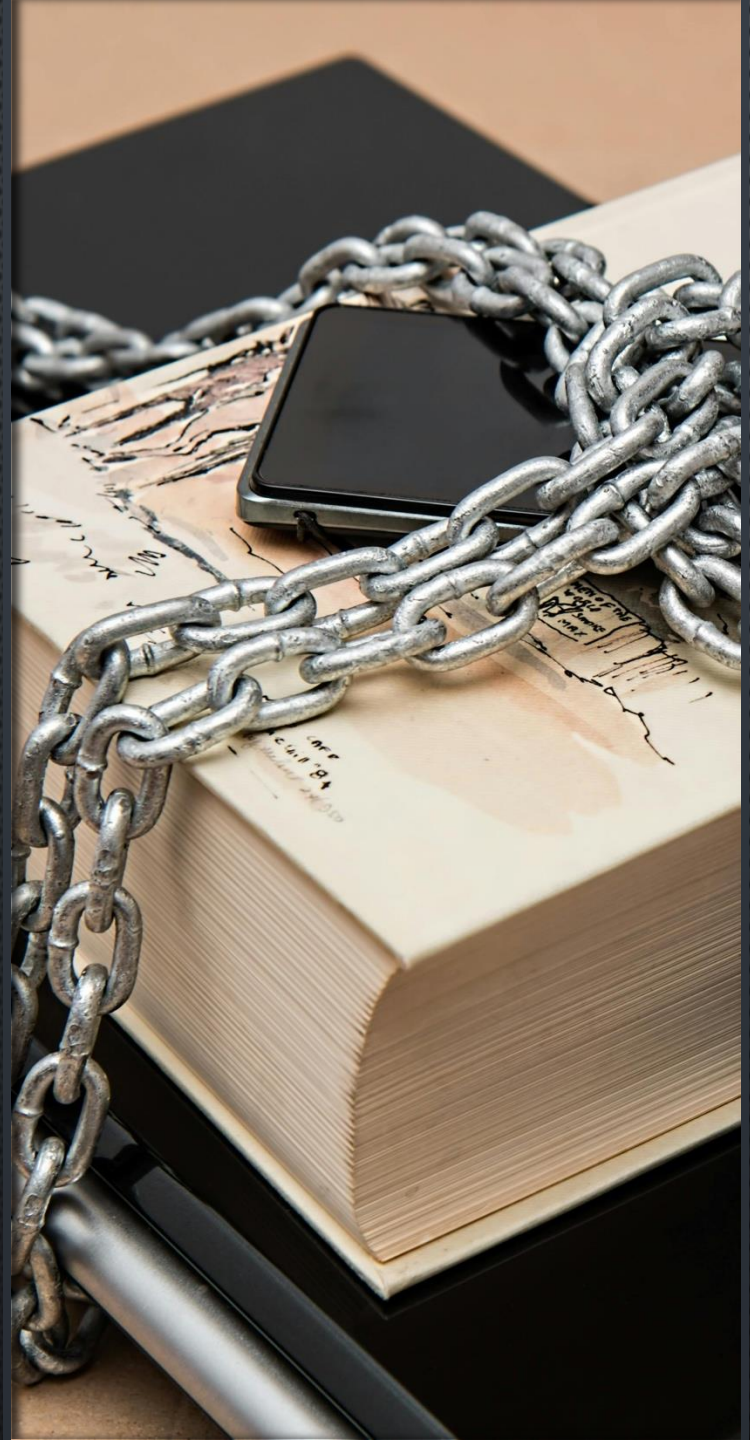
I LOVE DJANGO, BUT ALL MY FRIENDS ARE USING THESE COOL FRONT-END LIBRARIES LIKE REACT AND VUE

- FOMO
- IS THERE A SOLUTION?

A SIMPLE WAY TO INTEGRATE DJANGO AND REACT

- BUILD A REACT APP WITH CREATE-REACT-APP, OR ANOTHER FRONT-END TOOL
- BUILD A BACKEND API IN DJANGO
- REACT HANDLES THE FRONT END
- DJANGO SERVES ONLY AS AN API
- THEY RUN ON SEPARATE SERVERS WITH DJANGO SERVING AS AN API FOR THE REACT FRONT END

HERE'S AN APP WITH A REACT FRONT-END AND DJANGO BACKEND SERVING AS AN API



But, what's wrong with that?

- Nothing really, but…
- You can miss out on many comprehensive, built-in, features which are available out of the box in Django.
  - Template rendering
  - Form Handling and Validation
  - Authentication and Authorization
  - Session Management and CSRF protection
  - Admin Interface
  - URL Routing and Namespacing
  - To name a few…
- We could be having to duplicate on the front-end what Django already does
- Two separate code bases to maintain

Django/React Hybrid Architecture

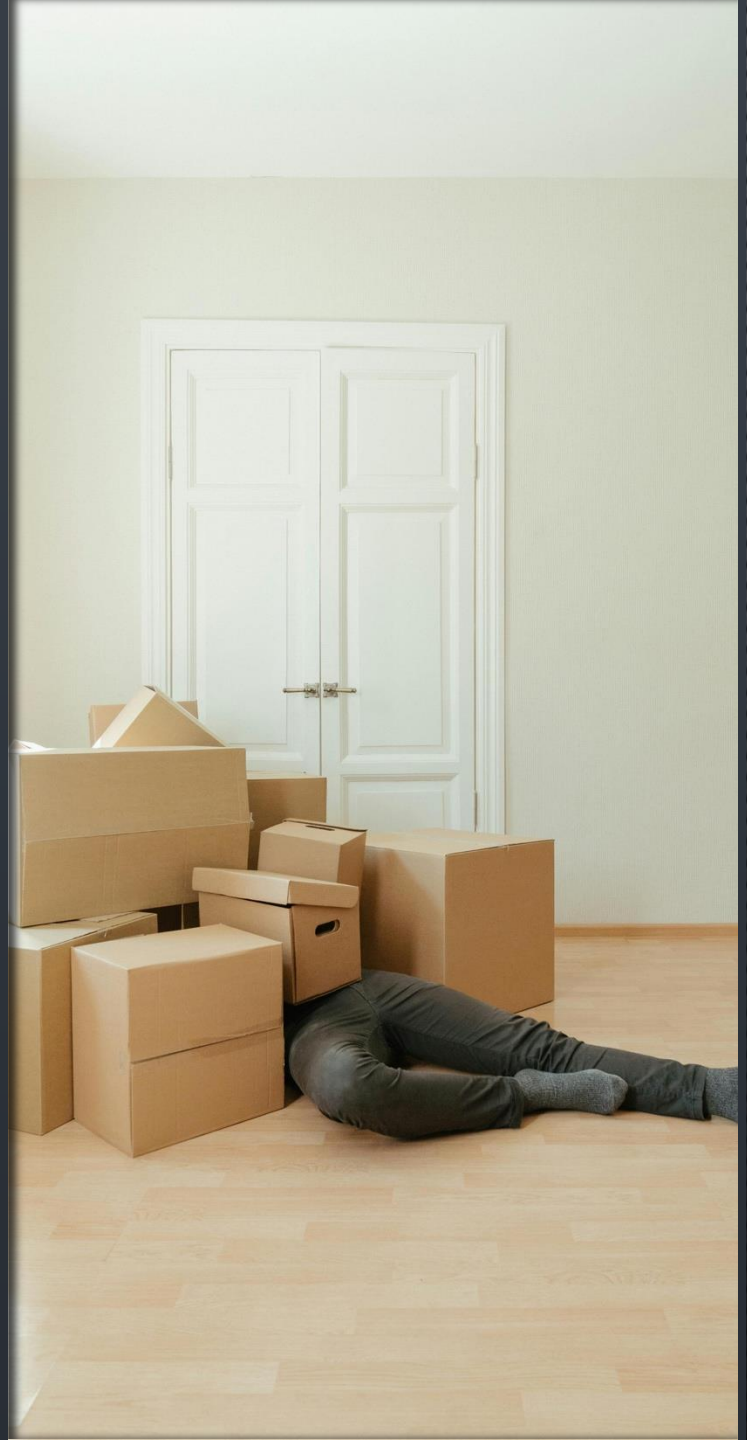
- Instead of having Django and React separate, where Django just functions as an API, a hybrid solution can provide the best of both worlds.
- React runs inside of your Django app

HERE'S A HYBRID DJANGO/REACT APP

A PIANO INVENTORY APP

HERE'S WHAT WE NEED TO BUILD THIS APP

- A Django project
- Package manager
- Bundler
- Compiler
- React
- Django Rest Framework

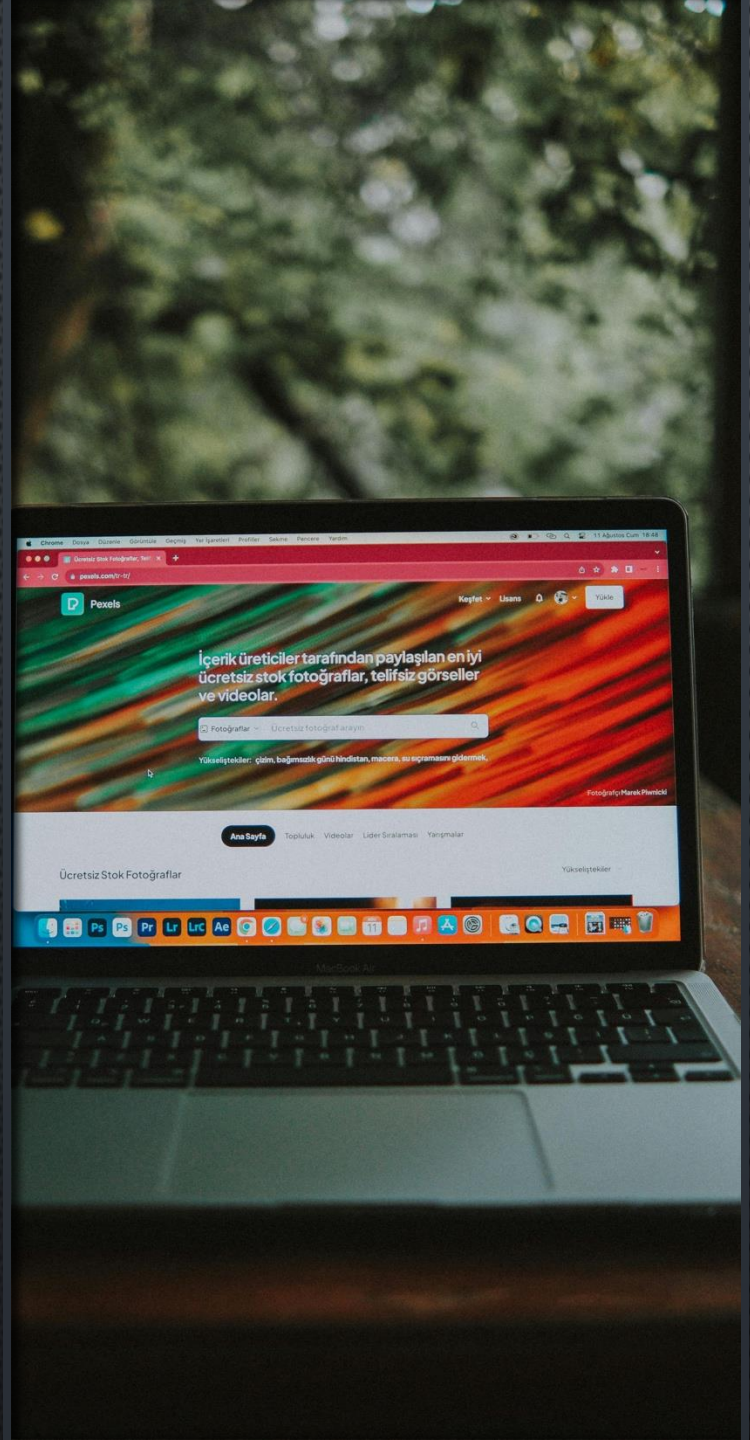


Package Manager

- Imports and uses packages that other people have built
- Basically, pip for JavaScript
- There are 2 popular ones, Yarn and NPM
- I suggest NPM
- Most widely used
- If you have Node.js installed on your computer, npm is included

BUNDLER

- Well, It bundles!
- Gathers the NPM (or Yarn) installed libraries and any code that you've written which may be spread across different files
- Compacts this code into a small number of files
- Enhances performance
- Many choices of bundlers
- I recommend the most popular one, WEBPACK



COMPILER (TRANSPILER OR TRANSCOMPILER)

- BABEL IS A TOOLCHAIN THAT IS MAINLY USED TO CONVERT ECMASCRIPT 2015+ CODE INTO A BACKWARDS COMPATIBLE VERSION OF JAVASCRIPT IN CURRENT AND OLDER BROWSERS OR ENVIRONMENTS.
- THE MOST POPULAR, BABEL
- BUT, WHAT DOES THAT MEAN?

JavaScript

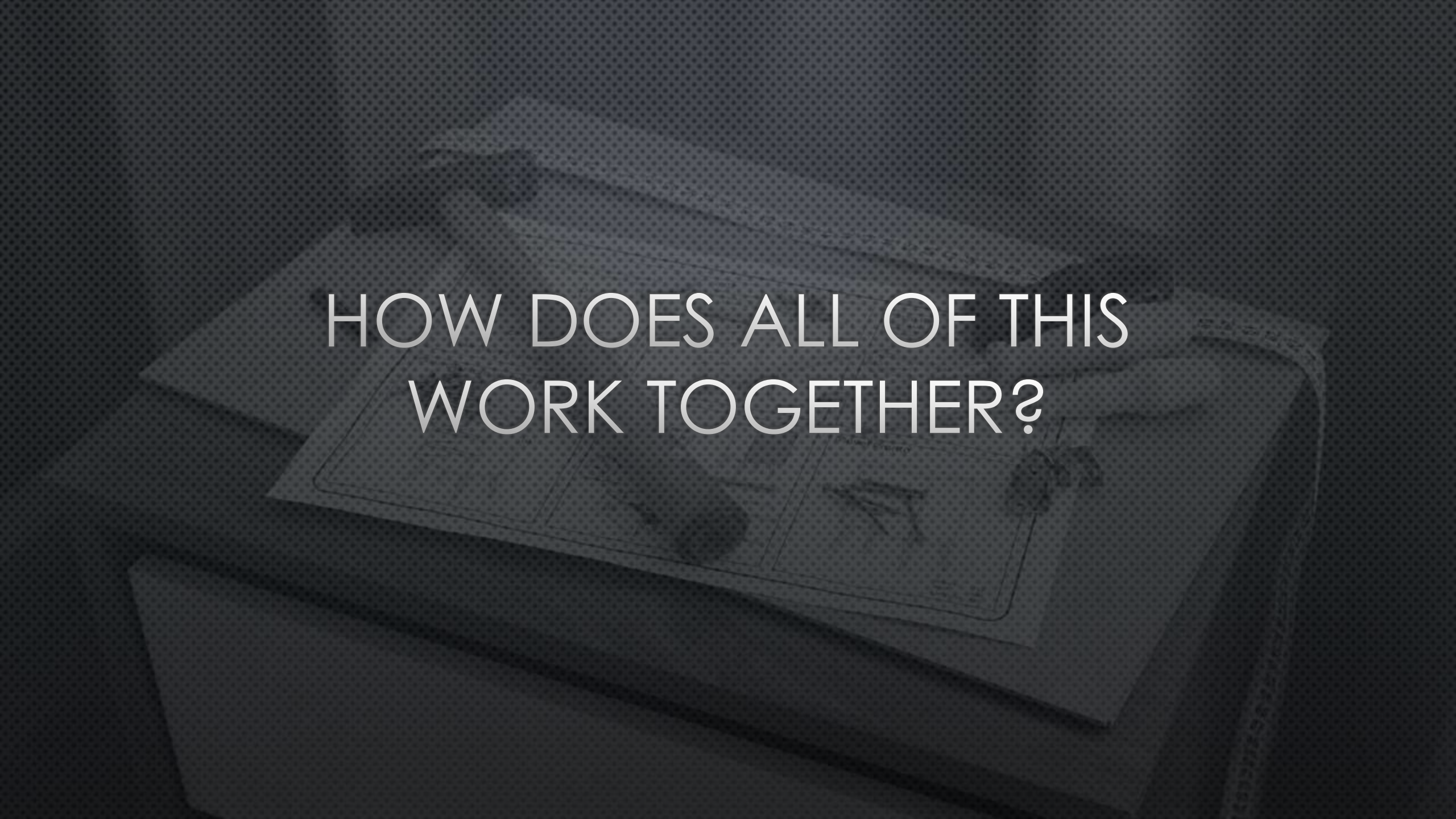
```
// Babel Input: ES2015 arrow function
[1, 2, 3].map(n => n + 1);

// Babel Output: ES5 equivalent
[1, 2, 3].map(function(n) {
  return n + 1;
});
```

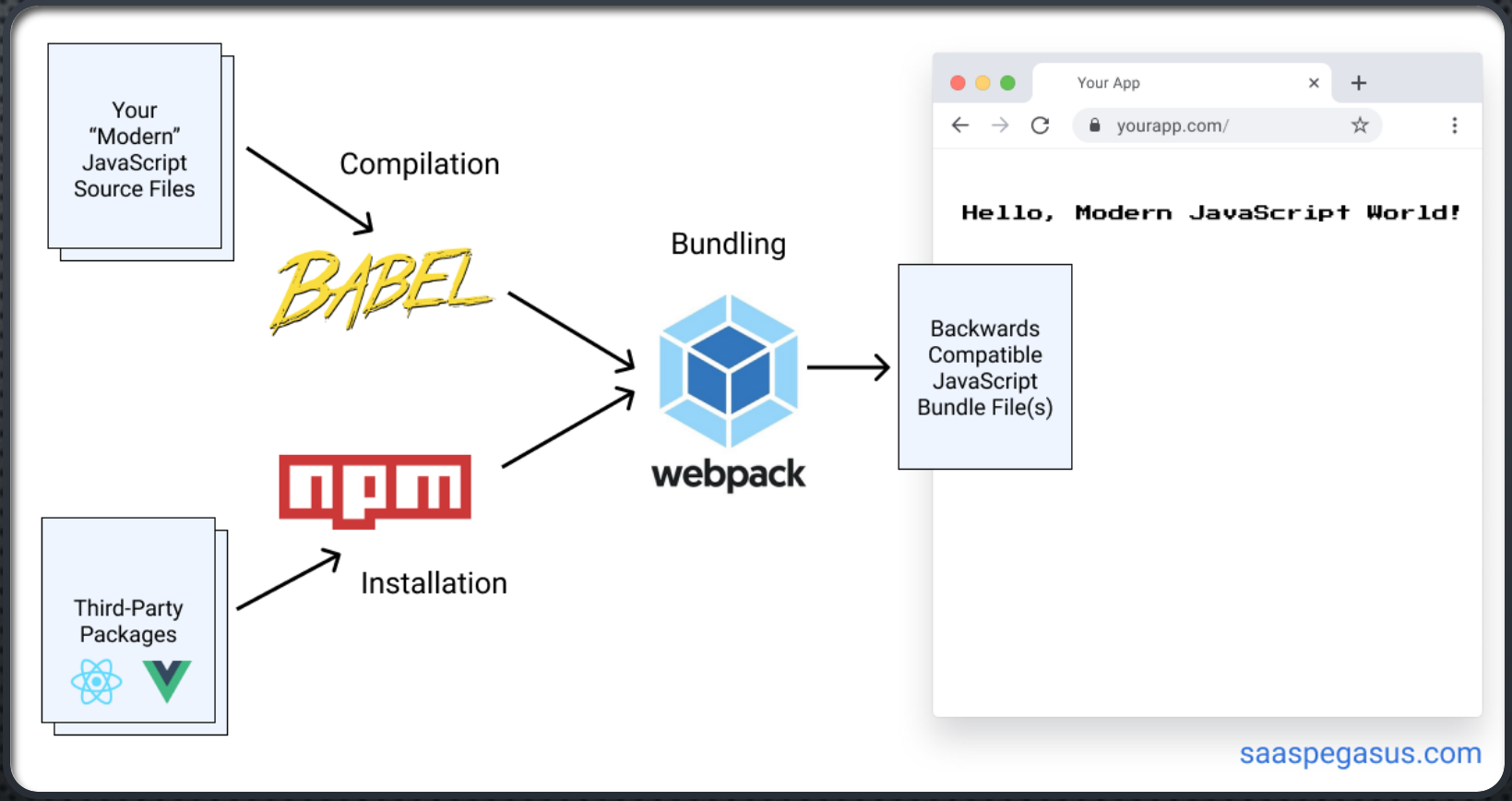
BABEL EXAMPLE

A powerful and flexible toolkit for building Web APIs.

You don't necessarily need this framework as you could roll your own serialization, but it provides a lot of useful abstraction.

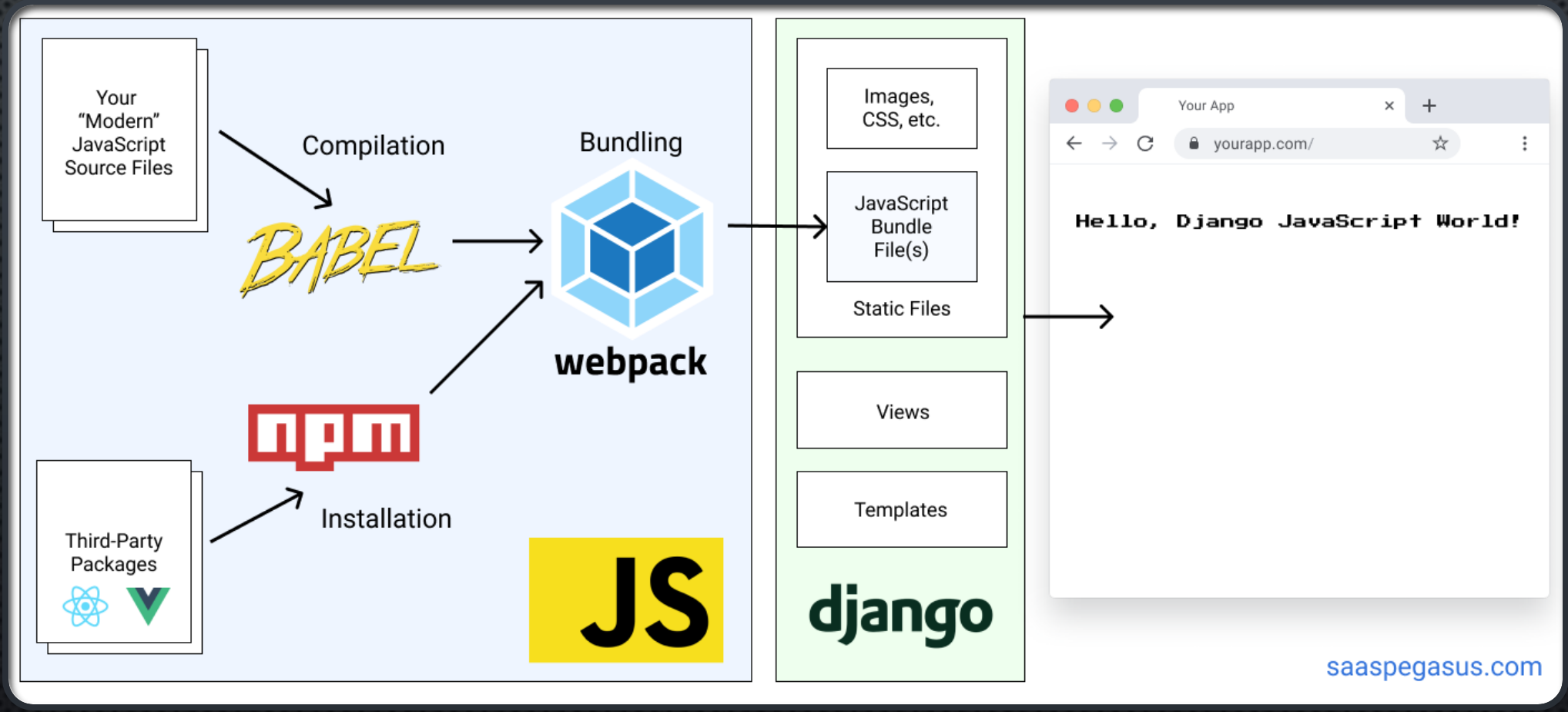

HOW DOES ALL OF THIS
WORK TOGETHER?

OUR JAVASCRIPT PIPELINE



GREAT! SO WHERE DOES DJANGO COME INTO THE PICTURE?

INTEGRATION OF THE JS PIPELINE INTO DJANGO

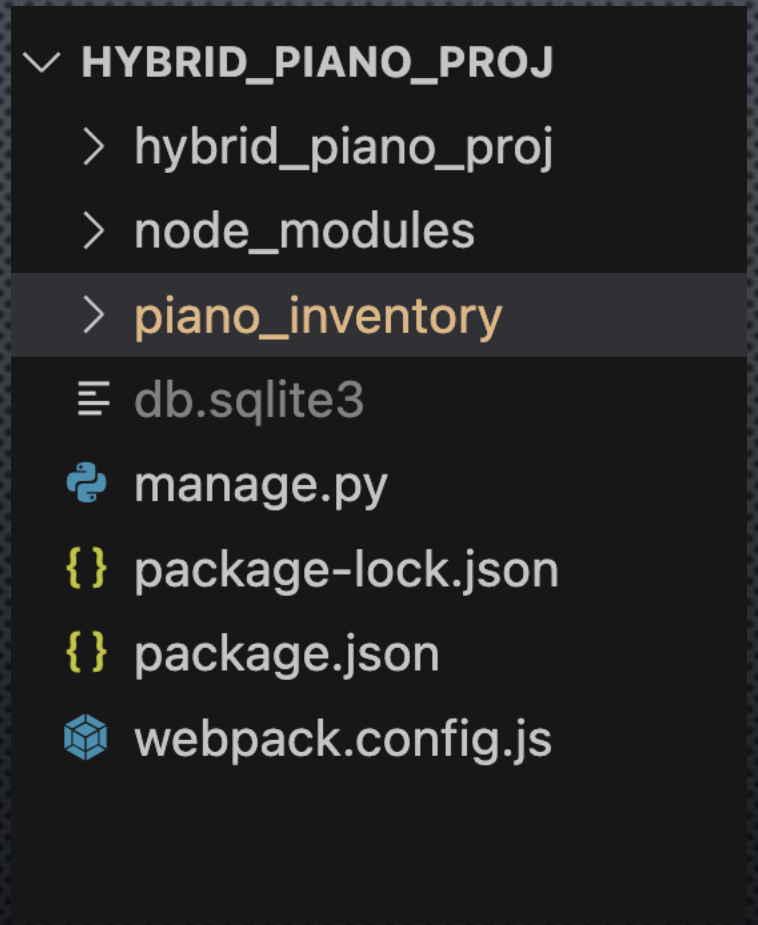


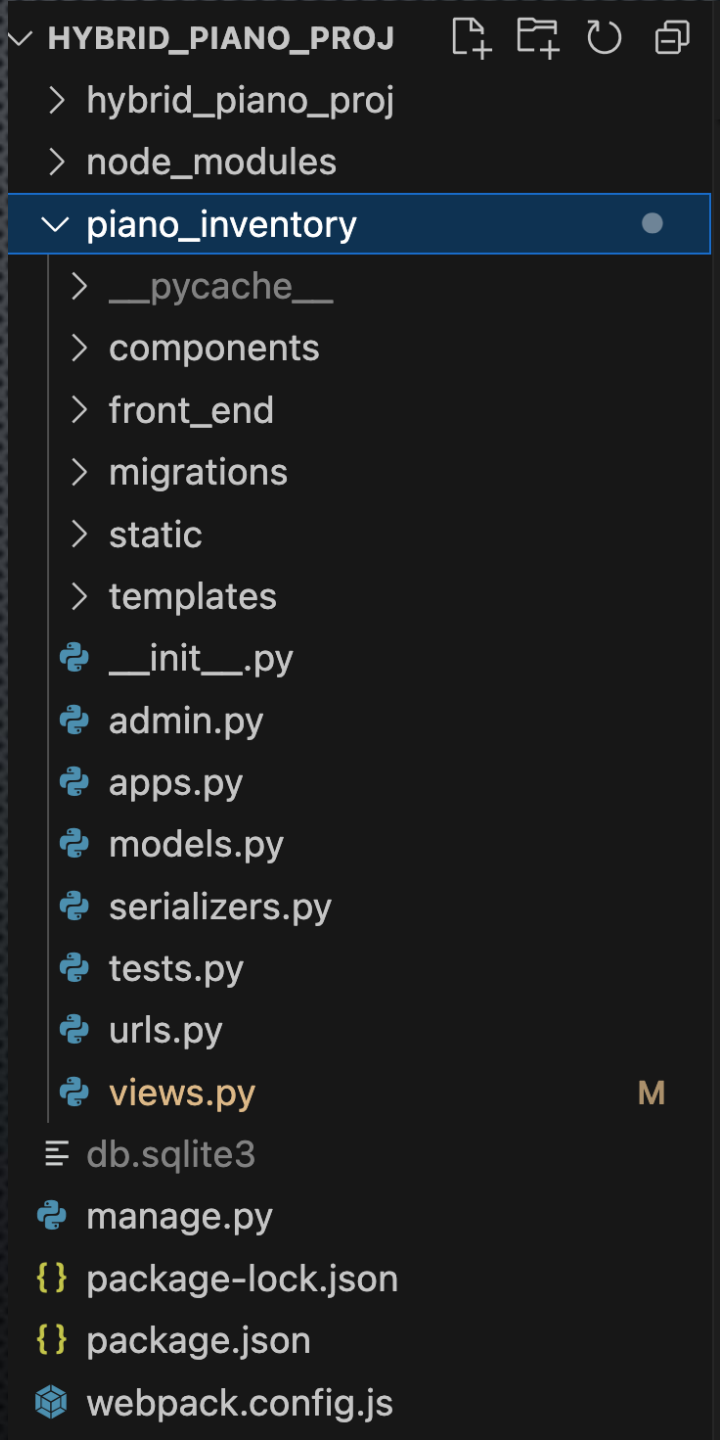
FIRST STEPS

We need to make some changes to our usual Django file structure

- There are various ways to do this, but I've chosen to stick with a structure that closely adheres to the practices of this course and the Django documentation.
- It's the most intuitive for me as it mirrors the static file structure of a normal Django app
- But, there are other ways to structure the app

FILE STRUCTURE OF DEMO APP





Hybrid Django/React app file structure

```
|----manage.py
|----myproject
|         |---- __init__.py
|         |---- settings.py
|         |---- urls.py
|         |---- wsgi.py
|----myapp
|         |---- components
|                   |----myReactComponents
|         |---- frontend
|                   |---- myapp
|                             |---- index.js
|         |---- migrations
|         |----static
|                   |----myapp
|                             |----(JS bundle file)
|                             |----scripts.js
|                             |----styles.css
|         |----templates
|                   |----myapp
|         |----__init__.py
|         |---- admin.py
|         |---- models.py
|         |---- serializers.py
|         |---- urls.py
|         |---- views.py
|----package-lock.json
|----package.json
|----webpack.config.js
|----db.sqlite3
```

Not all files are shown

SETTING UP WEBPACK

INSTALL WEBPACK IN THE ROOT DIRECTORY

- THIS WILL CREATE THE PACKAGE.JSON, PACKAGE-LOCK.JSON FILES AND A NODE_MODULES DIRECTORY WHERE OUR JS LIBRARY DEPENDENCIES GO.
- NODE_MODULES SHOULD BE IN YOUR .GITIGNORE FILE

```
npm init -y
npm install webpack webpack-cli --save-dev
```

WHAT ARE THESE FILES FOR?

- **PACKAGE.JSON** CONTAINS META DATA ABOUT THE PROJECT AND MANAGES ITS DEPENDENCIES (BLUEPRINT FOR THE PROJECT)
- **PACKAGE-LOCK.JSON** IS GENERATED BY NPM WHEN YOU INSTALL DEPENDENCIES. IT LOCKS THE EXACT VERSIONS OF THE DEPENDENCIES.

INSTALL BABEL AND REACT

```
npm install --save-dev babel-loader @babel/core @babel/preset-env @babel/preset-react
```

```
npm install --save react react-dom
```

STEPS TO GET WEBPACK WORKING

- CREATE AN **INDEX.JS** FILE IN YOUR FRONT_END/MYAPP/ FOLDER AN ADD A JS FUNCTION OR RENDER A REACT COMPONENT.
- CREATE THE **WEBPACK.CONFIG.JS** IN THE ROOT DIRECTORY LETTING WEBPACK KNOW WHERE THE SOURCE AND OUTPUT LOCATIONS FOR THE BUNDLED JS ARE
- ADD A SCRIPT TO THE **PACKAGE.JSON** "SCRIPTS" KEY TO RUN WEBPACK IN DEV MODE FROM THE TERMINAL
- NOW WE CAN RUN THE WEBPACK SCRIPT WHICH WILL OUTPUT OUR WEBPACK SCRIPT TO THE LOCATION WE DEFINED IN THE CONFIG FILE

WEBPACK.CONFIG.JS

- We're setting the location of the source JS file and where the bundled output will be
- For the time being, ignore the module key

front_end/piano_inventory/index.js

Let's explore

PACKAGE.JSON

- ADD A SCRIPT TO "SCRIPTS" KEY WHICH WILL RUN WEBPACK IN THE DEVELOPMENT MODE

Run npm dev

- Executes the webpack script
- If no errors, a bundled js file will appear in your static/piano_inventory/ folder
- Note that this is the same folder we've used throughout the semester for our static files.

Setting up Babel

- Add script in **webpack.config.js** to tell webpack how to process our file.
- Add the boilerplate as shown
- Consult the webpack website, bable-loader section for more details: https://webpack.js.org/loaders/babel-loader/

LET'S LOOK AT THE OUTPUT

WE'VE GOTTEN A REACT FRONTEND WORKING IN DJANGO, BUT WHAT ABOUT HOOKING IT UP TO A DATABASE.

CONNECT THE WEBPACK OUTPUT TO DJANGO

- WE'LL DO THAT IN TEMPLATES/PIANO_INVENTORY/INDEX_INVENTORY.HTML
- NOTE THIS IS OUR REGULAR TEMPLATE DIRECTORY
- WE'VE JUST ADDED A TEMPLATE WHICH IS GOING TO RENDER OUR JS.

SERVE THE TEMPLATE

- Define a view in view.py
- Define a route for the view
- Note I'm using a class view

Django class views

- Views in Django can be more than just a function, as we have used in this course, they can be a class
- You can structure your views by harnessing inheritance and mixins
- It's only mentioned here, because in today's demo app, a class-based view is used.

Django REST framework

- The framework is very deep and could easily take up a section (or 2) in and of itself
- The documentation is excellent
- Consult the documentation for a good tutorial on how to get started or detailed explanations for what follows
- A basic implementation is given in the demo that closely aligns with the type of function views we've used this semester

CONNECTING THIS TO DJANGO BACKEND WITH DJANGO REST FRAMEWORK

- LET'S LOOK AT THE RELATIONSHIP BETWEEN **MODELS.PY** AND **SERIALIZERS.PY** AND **VIEWS.PY**
- WE COULD HAVE ROLLED OUR OWN DJANGO API, BUT DRF PROVIDES US WITH A LOT OF BUILT-IN FUNCTIONALITY THAT ABSTRACTS AWAY MUCH OF THE COMPLEXITY

INSTALL USING PIP (DJANGORESTFRAMEWORK)

Add `'rest_framework'` to your `INSTALLED_APPS` setting.

```
INSTALLED_APPS = [
    ...
    'rest_framework',
]
```

How does this work from our React app

- From our App.js component a call is made to the running Django server
- The Django url router receives the request and routes it to the view
- The view receives the request:
  - Finds the Django model object(s)
  - Converts them into python native data types
  - Processes the logic
  - Returns a response in JSON to our React component

One last look at the app

- Feel free to fork the github repo and repurpose it,or use any part of it.
- Please let me know, if in following this tutorial you find an errors.

REMEMBER THE FINAL PROJECT DEADLINE, DEC.14

No extensions!

THANK YOU!

I'VE ENJOYED THESE SECTIONS AND CONGRATULATE ALL OF YOU ON A GREAT SEMESTER!