Author: Sherief Elsowiny

Title: JavaScript and Python a comparison of the two and their tooling within web applications.

Abstract: An exploration of the two programming languages Python and JavaScript looking under the hood of how it actually runs. Looking into the similarities of the two in terms of syntax and seeing practical applications and construction of a Django project and React project.

Keywords: Python, JavaScript, Django, React, MVC, Components, State, Web Development, Compiled, Interpreted, Static, Dynamic

1. Introduction

The purpose of this study is to see a modern comparison and evolution of the two programming languages, Python and JavaScript, while exploring their engines, syntax, and applications. Much has evolved within the web application atmosphere, and many different development stacks have come and gone with two technologies namely Python and JavaScript still remaining relevant to this day [1]. It's applications are used widely, but for this research, we will explore the popular frameworks built on top of Python and JavaScript and make a comparison of the two with some minor examples. To further refine our exploration, we will be focusing on the web application side of these technologies, as it offers much rich content in exploring not only the evolution, but the differences in applying known design patterns that are opinionated within the communities. This study is important for understanding the comparison of two popular frameworks, their evolution, their applicability and lastly, we will show practical examples of each and combine the two as well to show the power of these frameworks.

1.1 Purpose

Two of the most popular programming languages of the modern age are JavaScript and Python [1]. Both have similarities and both have various applications. With the ever constant change in software development, it is now a good time to grasp a solid understanding of where the current state of programming in application development for the web stands, as well as see some examples of current trends. Many improvements have arisen between Python and JavaScript with ever increasing communities maintaining and building libraries that better abstract the behind the scenes logic that is ever so prevalent across web application development [2, 3] When one learns a new language, a world of endless possibilities presents itself. When understanding multiple languages, one can extend their reach and capabilities much like that which can be seen in programming languages equivalently. We are curious about the differences between two most commonly talked about programming languages [1] and want to see the side by side comparison and applications of each.

To begin our research journey, we will be exploring the intricacies between the languages to get a solid baseline level of understanding of the languages and it's accompanying syntax. In doing so we will have a better understanding when we explore the applications of each of these languages within the context of web application development. We will build mini programs to highlight our understanding of the systems at play to better grasp the practicality of the languages. Finally, after our understanding of the system is complete, we will explore the combination of two languages communicating to one another, and effectively working together.. We will be able to then see how our concepts can be applied and deployed for the world to see.

1.2 Research Questions

What is the difference between Python and JavaScript, specifically what are its features? How can we use them in applications, specifically web applications? What exactly is the difference between React and Django (popular library and framework for these two languages)
What's the difference between SSR and Client side rendering?
Is it possible to use both, and if so.. How?

1.3 Motivation

After scouring through the web to understand modern development with modern tooling, I came across much outdated content pertaining to older versions of these tools. In an effort to understand the latest tooling and the newest latest features, I figured I'd dive a little deeper and explore these concepts at large, from the evolution of the current trends while seeing concrete examples. The question however is what exactly is happening underneath the hood? Better yet what is happening that's being abstracted and why? To better understand this we shall explore two current technologies being used that are built on top of JavaScript and Python. The thing however is, much is different now so we will need to also reference the past design patterns to understand current trends. We start with understanding what's happening in terms of the programming engines and then look into libraries that abstract logic away and allow for beautiful complex applications.

1.4 Limitations

There are rich resources all over the web that explore these topics, languages, and frameworks to an entirely different degree. For the purposes of this research however, we will be limited to exploring the core principles and intermediate capabilities of these languages and frameworks within the context of their primitive features and capabilities, to the current application in web development. Although many other frameworks exist for the two languages Python and JavaScript, React and Django remain to be the most popular of which we will explore.

2. Method

2.1 Structure of research question one

What is the difference between Python and JavaScript, specifically what are its features?
In this exploration we will focus on the common features you will see across programming languages. To be specific, this includes statements of logic or simple structures of data down to entire objects that contain methods. We will explore definitions as well as intricate details.

2.2 Structure of research question two

How can we use Python and JavaScript in applications, specifically web applications(most popular usage)?
For this we will briefly look into what available options are out there that arose since the inception of these languages. We will touch on those antiquitaded tools before diving into two very popular choices which are React and Django and explore these libraries and frameworks.

2.3 Structure of research question three

What exactly is the difference between React and Django(popular library and framework for these two languages) and how do we get it started?
For this research question, we will be getting our keyboard dirty and exploring and visualizing what’s happening in the web development field with these two popular tools.

2.4 Structure of research question four

What's the difference between server side rendering and client side rendering?
We will briefly look into what this means as it pertains to Django and React and see what happens with the server or client.

2.5 Structure of research question five

Is it possible to use both Django and React, and if so.. How?
We will be looking at constructing a very basic API in Django that communicates to a React front end.

2.6 System Setup

A PCor Mac shall be sufficient for this exploration as well as the following packages.

- Python 3 +
- Django
- React 16.8 +
- Node.js & Node Package Manager
- Pip (Python Package Manager)
- Replit (for quick exploring)

“Note that the GitHub with accompanying package information will be available such that this will be outdated and can be referenced if needed later”

3. Overview

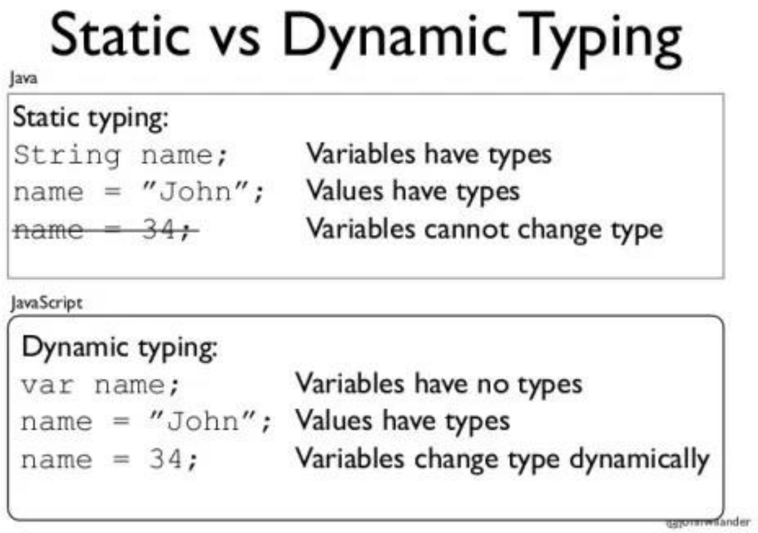
A look into the history.

To begin our journey we will start by defining what we are exploring. And that is the beautiful construct of programming languages and their applications. At the time of this writing much has changed since the adaptation of programming. Better home computing has allowed for modern programming languages to produce some beautiful end user results right in the browser. This contrasts previous models of web application development in which servers process the load and logic of the application and send the end results to the user or client. In more modern years, we see the rise of tools like React and Angular built on top of JavaScript where in the client or browser is able to handle more computabley hard rendering now than ever before. The way in which they do this is through various in depth ways such as evolving the core language, building libraries and standards associated with said libraries. Abstracting nasty repetitive logic that consistently occurs across applications and extending specific features to build better applications.

Contrasting the JavaScript libraries of React and the framework Angular, we have frameworks like Django for Python, which still adopt a server side rendering pattern for its displaying of content. Before we get too ahead of ourselves let’s explore the fundamentals first before we use them to better understand full functioning systems that implement them.

3.1 Static vs dynamic typing

Programming languages can be referred to as either dynamic or statically-typed, depending on whether there is explicit declaration of their variable type and if the variable type is known at runtime. If the interpreter handles the assigning of a data type to the variable at runtime, then it can be thought of as dynamic [12]. In contrast, in a statically-typed language the variable types are known and in most cases such as Java, C, C++, etc, the language requires that the type must be explicitly given [13]. When speaking and dealing with JavaScript and Python, you will come across the fact that both are dynamically typed languages. There are benefits to this of course, such that it allows for you to spend more time worrying about logic in the program and having the interpreter assign that type it believes is correct [16], however some errors may not be prevalent without type checking ahead of time [15]. There exist supersets built on top of JavaScript and Python that we will look into a bit later, that support type checking, in such that it helps to prevent errors that may occur at runtime.



Static vs Dynamic Typing

Java

Static typing:

```
String name;        Variables have types
name = "John";      Values have types
~~name = 34;~~          Variables cannot change type
```

JavaScript

Dynamic typing:

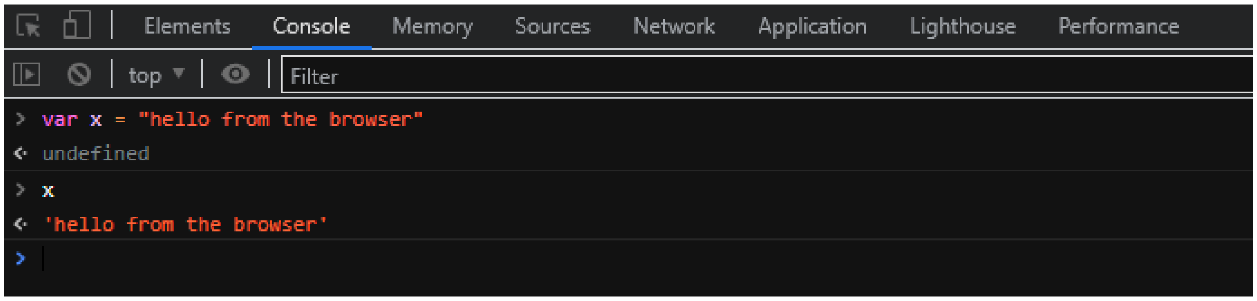
```
var name;           Variables have no types
name = "John";      Values have types
name = 34;          Variables change type dynamically
```

Fig1. Schroeder, Kelly. “The Static around Dynamic Languages.[14]

3.2 Compiled vs interpreted

To get a brief understanding of what it means for a programming language to be compiled or interpreted, has to do with whether the source code of that programming language compiles the source code to machine code, or if there is an interpreter to translate it [10]. JavaScript and Python are both interpreted languages [17, 18]. What happens specifically in how it is interpreted depends on the environment and programming language.

For example, the browser will typically interpret the JavaScript code and actually even do additional steps in order to optimize certain aspects that the browser may deem as “hot” or “warm" in which that particular code is executed a number of times [19]. This can be looked at later and is actually called Just In Time compilation and is how JavaScript is interpreted and run in the browser, with all browsers doing their own implementation [19]. Luckily all major browsers come with a JavaScript engine. This shows how powerful JavaScript is and vital for the web (although it can be disabled). The engine was initially an interpreter but as we’ll see later



```
> var x = "hello from the browser"
<· undefined
> x
<· 'hello from the browser'
>
```

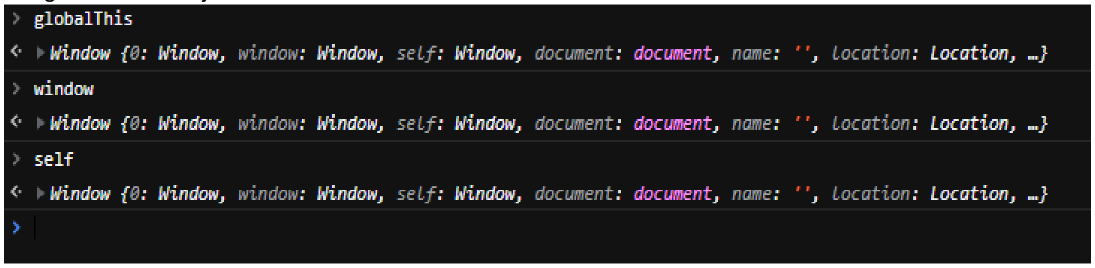
Fig 2 Broswer Console

has additional properties as well. In fact, JavaScript can be viewed and accessed from most browser developer tools. For example in Chrome we can see JavaScript in action here. (To open developer tools and follow along, Option + ⌘ + J (on macOS), or Shift + CTRL + J (on Windows/Linux) and go to Console). Here we can actually write typical JavaScript and have it executed.

When looking at JavaScript a little bit deeper within the browser, we can actually use a property to see the execution context of our global environment. This may be a little confusing, so to make it more clear, when JavaScript is run in the browser, the engine(for that particular browser) creates what is called a Global Execution Context, which is the window [21]. We can actually see this property and the associated variables and functions within it. An execution context is the environment in which the JavaScript code is executed. [21, 22]. There are different execution contexts, pertaining to whether the code is in a specific functional code block, or in our case, the Global execution context which would be the window object in our browser’s case [22]. In either case, we can access the global execution context in three ways: globalThis, window, or self. This all points to the window execution context and we can quite literally see some special features too that come with modern browsers.

Here we can actually see a quick example

Fig 3 Window object

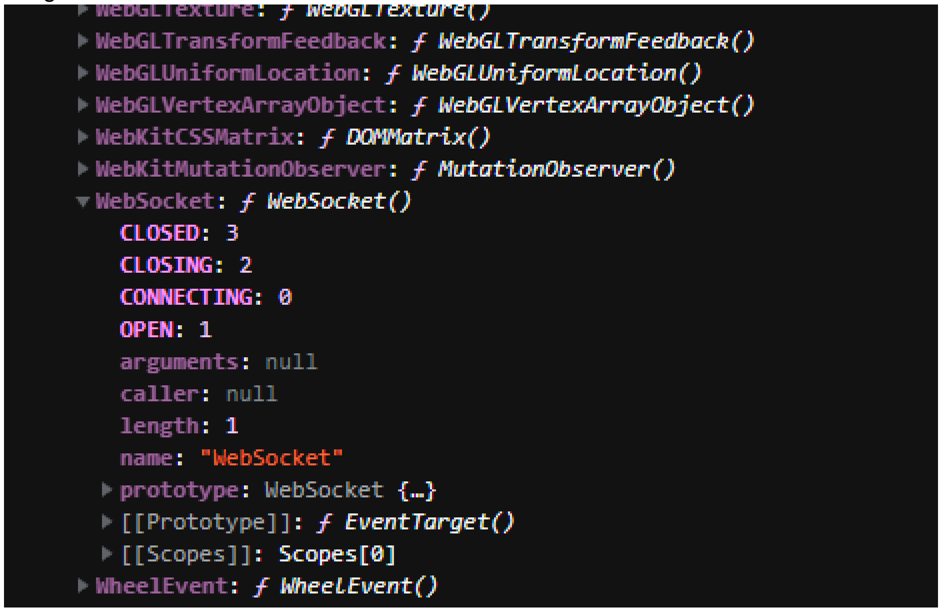


```
> globalThis
< ▸Window {0: Window, window: Window, self: Window, document: document, name: '', location: Location, …}
> window
< ▸Window {0: Window, window: Window, self: Window, document: document, name: '', location: Location, …}
> self
< ▸Window {0: Window, window: Window, self: Window, document: document, name: '', location: Location, …}
>
```

of an advanced tool that our browser has and that is WebSockets. This is accessed by clicking on the window property that we typed in earlier. WebSockets allow us to communicate in a two-way manner instantly and faster than before with server to client. It works by creating a handshake that upgrades to a websocket connection and allows for easier communications with video chatting and instant messaging [24].

When working with modern JavaScript, a lot happens behind the scenes and under the hood that for the average user, wouldn't even realize is happening. We can access it easily and see it from our developer tools however and this allows us to see all the rich features our browser comes with.

Fig 4 Websockets function



```
  ▸WebGLTransformFeedback: ƒ WebGLTransformFeedback()
  ▸WebGLUniformLocation: ƒ WebGLUniformLocation()
  ▸WebGLVertexArrayObject: ƒ WebGLVertexArrayObject()
  ▸WebKitCSSMatrix: ƒ DOMMatrix()
  ▸WebKitMutationObserver: ƒ MutationObserver()
  ▾WebSocket: ƒ WebSocket()
     CLOSED: 3
     CLOSING: 2
     CONNECTING: 0
     OPEN: 1
     arguments: null
     caller: null
     length: 1
     name: "WebSocket"
    ▸prototype: WebSocket {…}
    ▸[[Prototype]]: ƒ EventTarget()
    ▸[[Scopes]]: Scopes[0]
  ▸WheelEvent: ƒ WheelEvent()
```

Python, although called an interpreted language, actually does compile it's code down to something called Python bytecode [25].   When we execute a Python file, I.E a file with a .py extension, Python will compile it into bytecode where rather than being direct machine code for the computer to understand, a Python Virtual Machine is used to interpret and execute it [26].

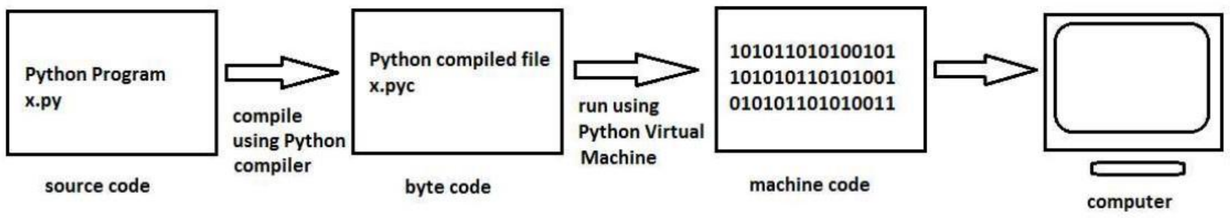
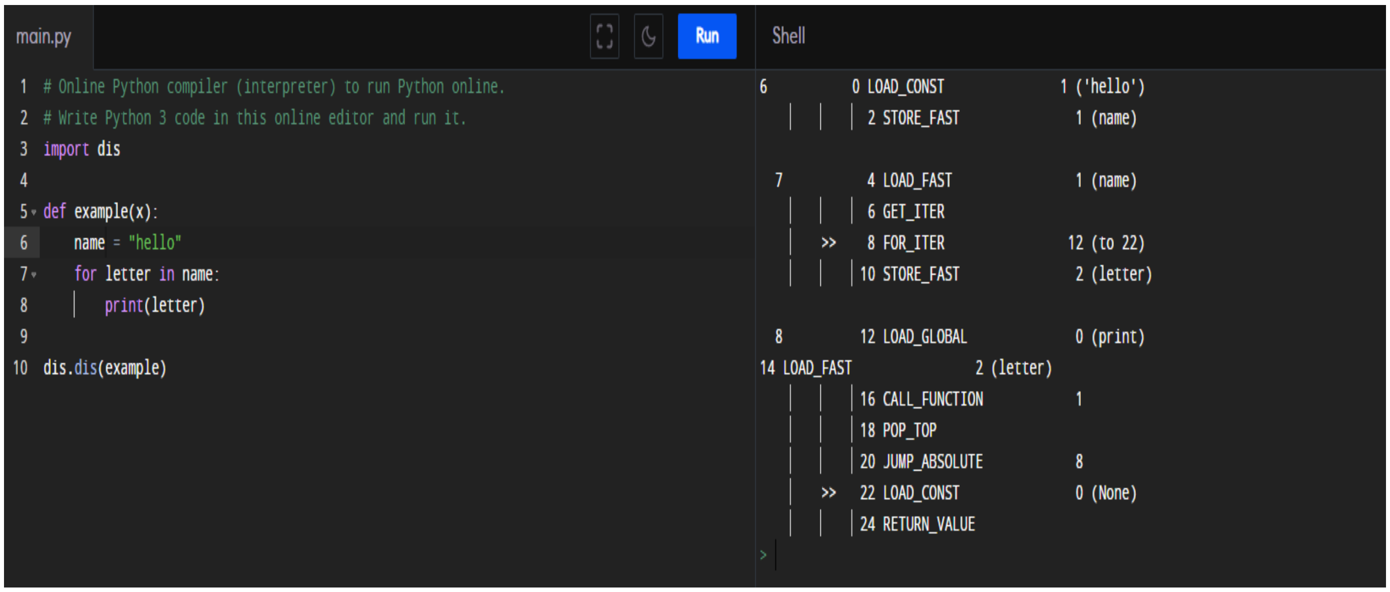


Fig 5. PVM [27]

We can actually see bytecode by using a library called dis, for disassembler [28], and thanks to modern computing advancements we can actually run Python in an online environment, and see what happens.



```
# Online Python compiler (interpreter) to run Python online.
# Write Python 3 code in this online editor and run it.
import dis

def example(x):
    name = "hello"
    for letter in name:
        print(letter)

dis.dis(example)
```

```
6           0 LOAD_CONST               1 ('hello')
            2 STORE_FAST               1 (name)

7           4 LOAD_FAST                1 (name)
            6 GET_ITER
      >>    8 FOR_ITER                12 (to 22)
           10 STORE_FAST               2 (letter)

8          12 LOAD_GLOBAL              0 (print)
14 LOAD_FAST                2 (letter)
           16 CALL_FUNCTION            1
           18 POP_TOP
           20 JUMP_ABSOLUTE            8
      >>   22 LOAD_CONST               0 (None)
           24 RETURN_VALUE
```

Fig 6 Byte code conversion

The Python code on the left is shown in it’s bytecode equivalent on the right. We can see these specific instructions and an accompanying definition from the Python docs here. https://docs.Python.org/3/library/dis.html

For now we can see obvious aspects where LOAD_FAST is used to reference our local variables: name, and letter. We can also see more functions being used where an iterator is used for the call stack and goes about popping and calling the functions.

When working with Python files, you may notice or come across that it compiles the source code to .pyc, or Python's compiled bytecode and when looking at the disassembler library, we can see exactly how this Python code gets turned into bytecode [29].

Similar to how JavaScript has implementations run in the browser with different engines and such, Python also has different implementations for executing its code [30]. In fact Python can be run on a Java virtual machine or even through a C compiler if one of it's implementations are used such as Jython or Cython [30].

There is even a Python implementation of Python itself. It's known as PyPy and extends features of Python where it allows for faster execution of Python thanks to the same thing some JavaScript browsers use and that is Just-In-Time compilation [31].

With the use of the JIT compiler, it allows for the code to be optimized, by having certain parts of the code that are frequently used, pulled out and converted to machine code during runtime and swapped with the previous code[32, 33]

3.3 JavaScript brief history and introduction

JavaScript is a programming language created in 1995 [5] by Brendan Eich at Netscape [4]. It is a dynamic language such that it does not require the explicit declaration of the variable types that may be seen in other programming languages [6]. JavaScript can be used in many different capacities, but most notably it is seen running in the browser [5]. The JavasScript language conforms to the ECMAScript language specifications [4] which releases on occasion a detailed specification for a general purpose language [7].

3.4 ECMAScript

Ecmascript was created by ECMA, an organization dedicated to standardization of information and communication systems [34]. The idea behind ECMAscript was to set a standard for which languages would adhere to. These specifications outline certain behaviors or functionalities that a programming language should have [35]. For JavaScript, and especially the version we will be using in this paper, it is important to note that we will be using JavaScript that conforms to the latest version of Ecmascript at the time of this writing. Ecmascript has seen an evolution of introducing classes, asynchronous/await functions, as well as promises and scoped variable decorations [36...39]. While Ecmascript continues to improve, JavaScript continues to improve its functionality by adopting the standards set forth by Ecmascript. For this, we are able to explore the latest of JavaScript in terms of its features in not only object oriented development, but in more complex tasks such as asynchronous functions or promises.

3.5 TypeScript

It is worth mentioning that there exists a superset of JavaScript that allows for what can be perceived as a limitation of the language and that is it's dynamically typed. Typescript exists to fix this and acts as a wrapper over JavaScript in which it has type support [40]. Typescript compiles down to JavaScript, and acts as a safety wrapper..

3.6 Python brief history and introduction

Python was created in the early 1990's by Guido Van Rossum [8] and can be described as an interpreted, high-level programming language with dynamic typing and object oriented capabilities [49]. Python can be used for a wide range of applications and because of its easy syntax, allows for quick prototyping of ideas.

3.7 Cython

Cython is a compiler and superset of Python that allows for adding static type declarations in Python syntax. It supports calling C functions and declaring your variables with C types [50]

3.8 Frameworks and libraries

Much like programming languages have core sets of features that allow for various applications, there exist frameworks designed and tailored to be the best fit with opinions for designing a particular system. Libraries exist to extend reusable code that abstract away some of the annoying logic that you would have to come across, and allows you to call their functions to get your job done quicker. It abstracts away a lot of the redundancy seen across application development. Two specific libraries and frameworks we will be focusing on are React and Django [2 ,3].

3.9 Engines

JavaScript has various engines [56] in which it runs within the browsers. To briefly see a list of which engines run in which browser we have:

- Firefox - SpideMonkey Engine
- Safari - JavasScriptCore
- Chrome - V8
- Edge - V8/Chromium(after rebuild)

Another popular runtime for JavaScript is Node.js which uses the V8 engine [56]. All these engines conform to ECMAScript and handle the parsing of the JavaScript code as well as the interpreter and compilations that come with transforming different versions of JavaScript [61].

4. Result

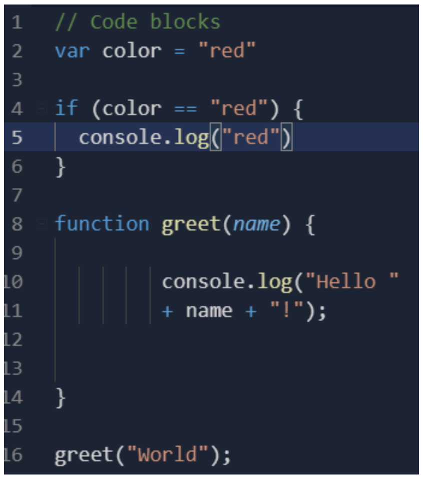
4.1 Syntax and semantics differences

Let us look at a couple of common features and aspects of the two programming languages to get a better understanding of a baseline for later when we dive into the web application aspects. We draw upon this article by FreeCodeCamp, an organization committed to making programming more accessible, and follow some of what they outlined as key differences that should be noted in the syntax. We will also look at advanced features that these languages recently introduced. We will observe these features and key details in an easy to follow replit, which allows us to write JavaScript or Python in the browser [35].

4.1.1 Code blocks

4.1.1.1 JavaScript Code blocks

Fig 7. JavaScript codeblock



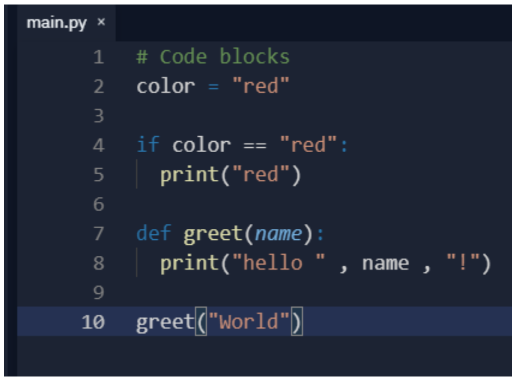
```
// Code blocks
var color = "red"

if (color == "red") {
  console.log("red")
}

function greet(name) {

            console.log("Hello "
            + name + "!");


}

greet("World");
```

Although JavaScript and Java are not similar at all, they are alike in how they group their statements together. This is similar as well to other programming languages in which JavaScript uses curly braces.

JavaScript also allows for having your code on various lines and even indented unevenly. This contrasts Python in which we will see that it uses indentation to signify code blocks. JavaScript, as well as Python both allow for usage of whitespace and doesn’t care for it so long as it stays within what it defines as it’s codeblocks [35]. Nonetheless in our example we see a statement encapsulated by it’s brackets, as well as a function with white space inside, being encased by brackets as well. If you look further you will notice that the semicolons are optional [10]. This must be taken into consideration however, as there can be unintended consequences if not attended to. JavaScript will actually insert a semicolon for you [10], but it is recommended to use them if there is more than one piece of code on that line [11].

4.1.1.2 Python code blocks

Fig 8. Python codeblock



```python
# Code blocks
color = "red"

if color == "red":
    print("red")

def greet(name):
    print("hello " , name , "!")

greet("World")
```

Python strives itself on being as humanly readable as possible.

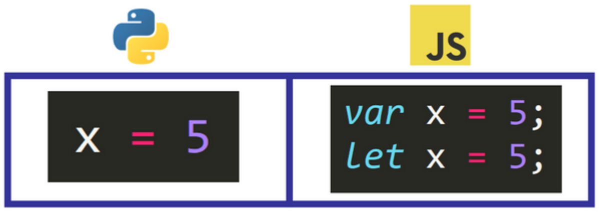
In Python , we group statements and code blocks by using indentations. The question arises with how many spaces to use to signify an indentation, and in reality it doesn't matter so much as long as you stay consistent. However, there is a style guide that actually suggests using 4 spaces and not tabs [43], referred to as the PEP-8 style guide. Some choose to still use tabs and that's fine, but it should be mentioned what is out there.

4.1.2 Variables and naming convention

In Python, we use the snake_case naming style for our variables as according to the PEP-8 style guide[44]. When defining a variable in Python, we simply follow the format below.

<variable_name> = <value>

In the case of JavaScript, we usually see variables using "camelCase" for their variable naming convention [45]. When defining a variable in JavaScript, we simply follow the format below.



```
x = 5
```

```javascript
var x = 5;
let x = 5;
```

Fig 9. Example of variable declarations [41]

var <variableName> = <value>;

JavaScript lets you define a variable by using either var, let, or const. Var creates a variable that is globally scoped or if inside a function, scopes it to that function block. Problems arose with this, and so JavaScript came out with the let keyword for block scope variables [46]. We can also define a constant as something that is immutable or can't change. In JavaScript we achieve this by using the const keyword followed by the variable name in all caps. In Python, we convey a constant by just using all caps (similar in JavaScript) while following the snake_case convention as Python doesn't have any explicit keyword for defining a variable [41].

4.1.3 Data types and primitives

4.1.3.1 JavaScript

In JavaScript, there are 7 primitive data types: string, number, bigint, boolean, undefined, symbol, and null [62]. For calculations on numbers and decimals, we use the same type which is number. The way JavaScript thought of this was that they figured any number could be equivalently computed as a decimal by adding a .0 to it. For example, 1 becomes 1.0 [41]

4.1.3.2 Python

In Python, there are 4 primitive data types: integers, floats, booleans and strings [63]. In Python, we can do calculations on plain integers or decimal equivalents of what is called float [41]. Python also has built in support for complex numbers and calculations with them[47]. Python similarly has a keyword for the null type and that is None [48].

In comparing the two, we see that for indicating that a variable at a particular time in that code block has no value, we use null for JavaScript and None for Python [41].

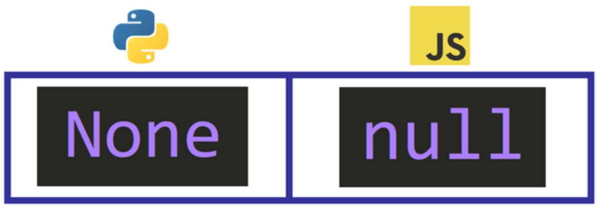
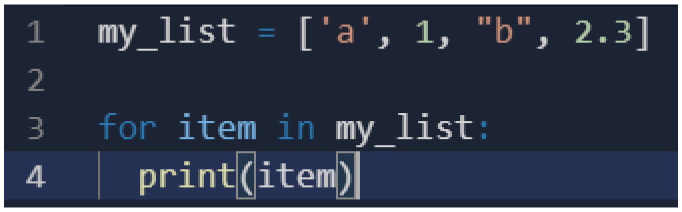


Fig 10. null vs none [41]

4.1.4 Data Structures

When organizing data for your application it is important to consider the obvious data structures that you may need. Some that come to mind are simple lists, or key value structures to denote a keyword to a particular value. Other data structures exist and serve to help with your application such as tuples or sets in Python or JavaScript objects in JavaScript [62, 64].

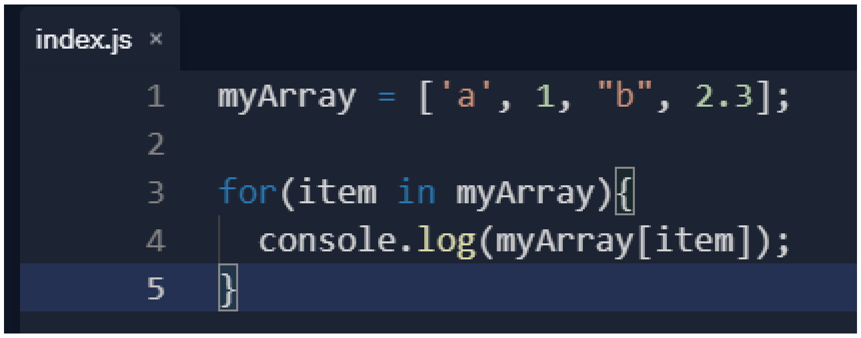


```python
my_list = ['a', 1, "b", 2.3]

for item in my_list:
  print(item)
```

Fig 11.Python List

4.1.4.1 Lists

In Python we have lists which allow us to store items in a sequence [41].
We can store items of any type in this list and don't have to explicitly declare the list either.
Similarly we can do the same in JavaScript calling it an array [41].



index.js ×

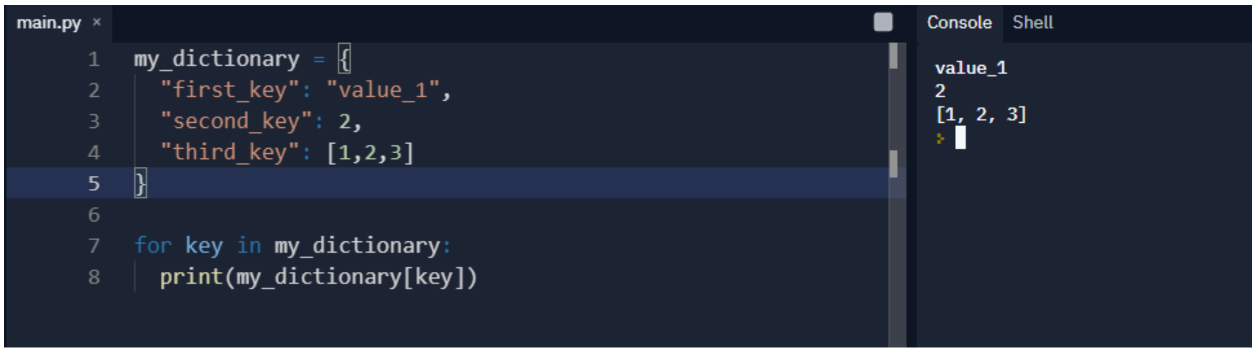
```javascript
myArray = ['a', 1, "b", 2.3];

for(item in myArray){
  console.log(myArray[item]);
}
```

Fig 12 Javascript Array

4.1.4.2 Dictionaries

Dictionaries are powerful in that they allow for key-value mapping. As of Python 3.7, dictionaries are ordered and do not allow duplicates [52]. It acts like an implementation of a hash-map where similarly there's a key-value mapping.

Fig 13 Python Dictionary

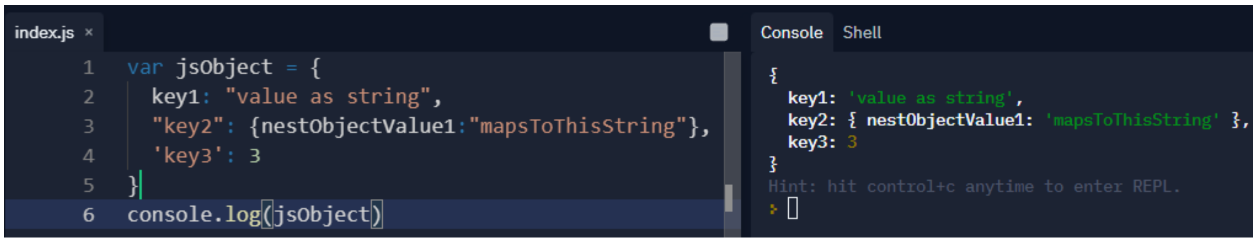


```python
my_dictionary = {
  "first_key": "value_1",
  "second_key": 2,
  "third_key": [1,2,3]
}

for key in my_dictionary:
  print(my_dictionary[key])
```

```
value_1
2
[1, 2, 3]
›
```

Here we can observe the dictionary containing multiple different value types and being accessed by their key. We can also see for the third key that there is a list, and Python even supports nested dictionaries. When accessing items we can reference the actual name of the key as well [52].

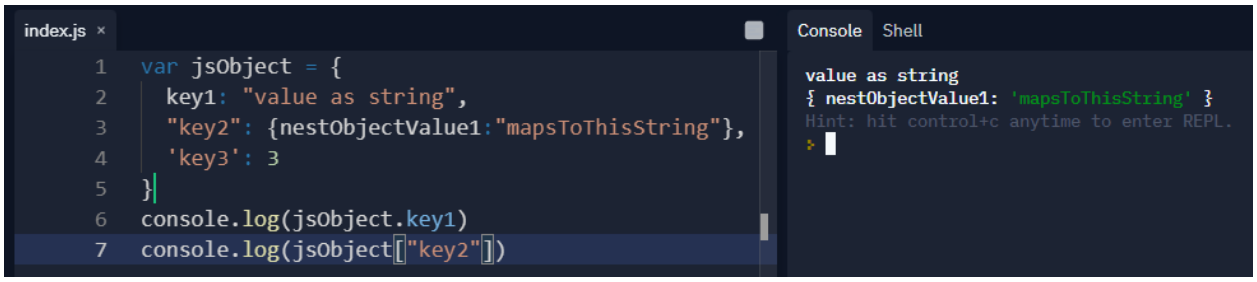
JavaScript is very flexible in it's language and doesn't directly have a dictionary, however it does have objects that can contain multiple properties that map to values [62]. As we'll see later, this notation is so powerful that it is actually commonly used for communicating data between



```javascript
var jsObject = {
  key1: "value as string",
  "key2": {nestObjectValue1:"mapsToThisString"},
  'key3': 3
}
console.log(jsObject)
```

```
{
  key1: 'value as string',
  key2: { nestObjectValue1: 'mapsToThisString' },
  key3: 3
}
Hint: hit control+c anytime to enter REPL.
›
```

Fig 14 JavaScript Object



```javascript
var jsObject = {
  key1: "value as string",
  "key2": {nestObjectValue1:"mapsToThisString"},
  'key3': 3
}
console.log(jsObject.key1)
console.log(jsObject["key2"])
```

```
value as string
{ nestObjectValue1: 'mapsToThisString' }
Hint: hit control+c anytime to enter REPL.
›
```

Fig 15 JavaScript Object Access

applications on the web [65]. The name we're referring to is JSON, and that stands for JavaScript Object Notation. It is a little different than actual JavaScript objects, however it allows for the key-value mapping that is seen in dictionaries.

Here we should make note that the keys are always going to be converted to a string. We could define them with either quotes, single quotes, or not at all. Convention goes with not. We can also observe how there is a nested object in this example as well. We can access properties by objectName.propertyName or objectName["propertyName"]. [54]

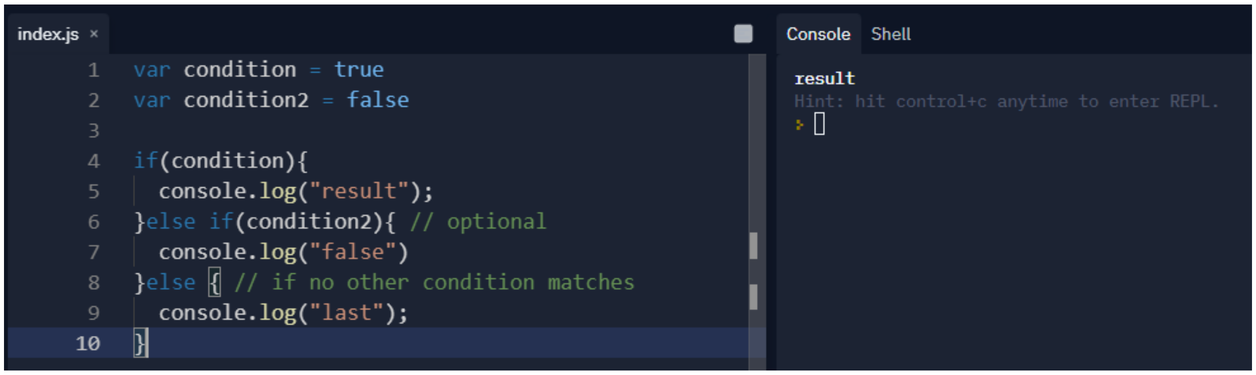
4.1.5 Operators

JavaScript and Python both support the same operations such as the equals sign signifying values being assigned to a variable, as well as the typical addition and multiplication/division [66, 67]. There are slight differences in each. Most notable is JavaScript's edition of weak equality and strict equality where two equal signs don't check for the data's type when making a comparison, so three(===) equal signs are used to equate the types as well [41].
Both languages support assignment operators, comparison, arithmetic and other rich detailed operations.

4.1.6 Conditionals

To see conditionals in Python we use an if/elif/else statement where we follow the convention of Python's syntax to indicate a code block [68].

In JavaScript we explicitly write out else if (rather than elif) and use parentheses to indicate code blocks[41].



```
var condition = true
var condition2 = false

if(condition){
  console.log("result");
}else if(condition2){ // optional
  console.log("false")
}else { // if no other condition matches
  console.log("last");
}
```

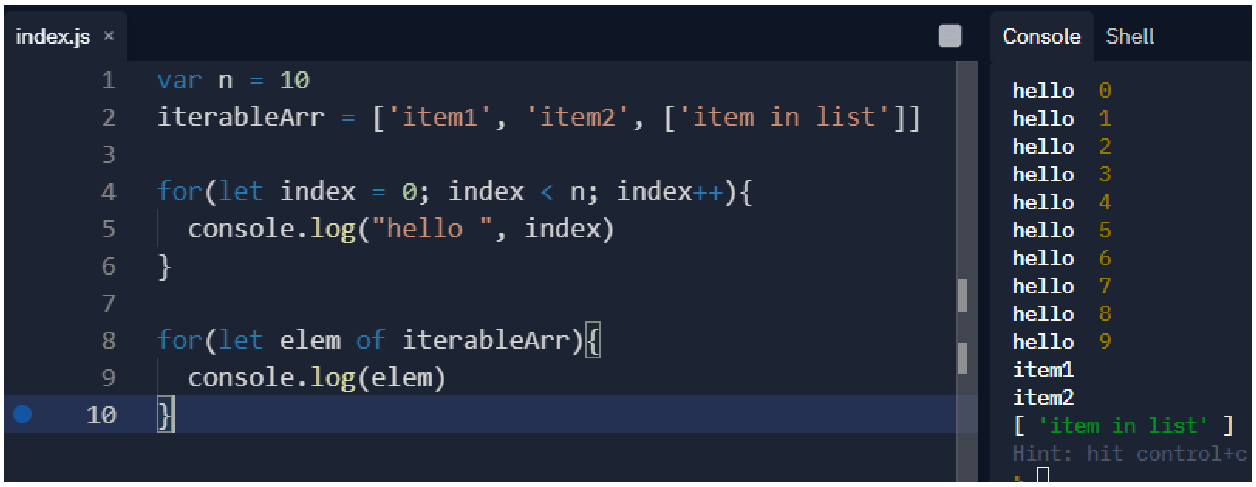
```
result
Hint: hit control+c anytime to enter REPL.
›
```

Fig 16 Javascript conditionals

4.1.7 Loops

We can easily iterate through a list or other iterable by using *for loops* or *while loops*. We can also use loops to continuously perform a task.

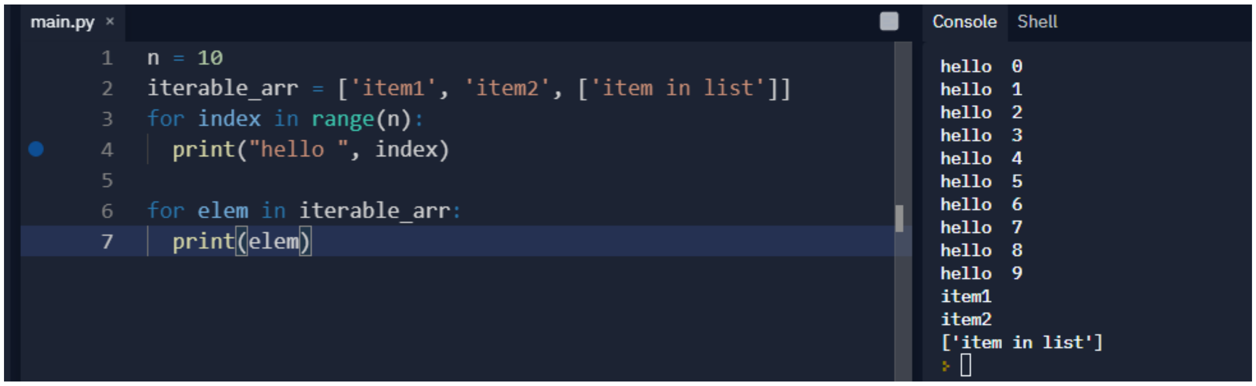
We use an index and range to which we perform a specific task a number of times.



```javascript
var n = 10
iterableArr = ['item1', 'item2', ['item in list']]

for(let index = 0; index < n; index++){
  console.log("hello ", index)
}

for(let elem of iterableArr){
  console.log(elem)
}
```

```
hello  0
hello  1
hello  2
hello  3
hello  4
hello  5
hello  6
hello  7
hello  8
hello  9
item1
item2
[ 'item in list' ]
Hint: hit control+c
```

Fig 17 Javascript iteration and traversing



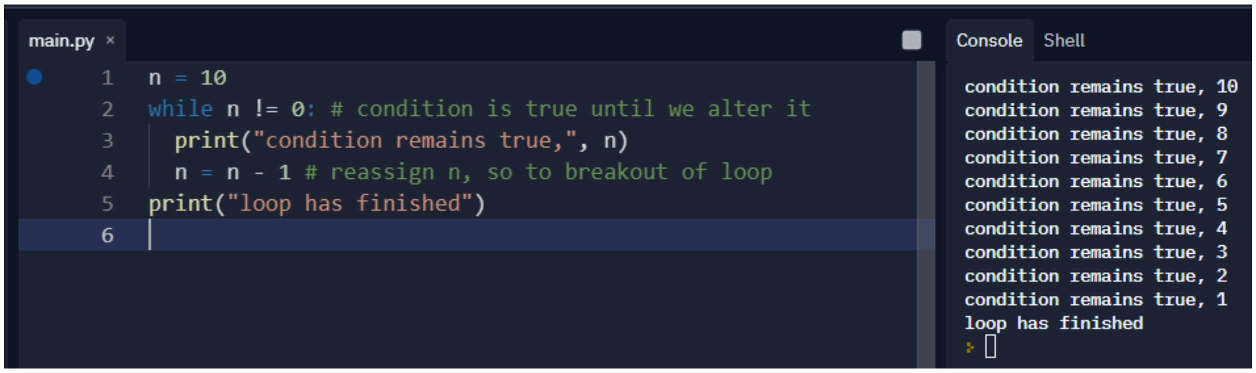
```python
n = 10
iterable_arr = ['item1', 'item2', ['item in list']]
for index in range(n):
  print("hello ", index)

for elem in iterable_arr:
  print(elem)
```

```
hello  0
hello  1
hello  2
hello  3
hello  4
hello  5
hello  6
hello  7
hello  8
hello  9
item1
item2
['item in list']
```

Fig 18 Python iteration and traversing

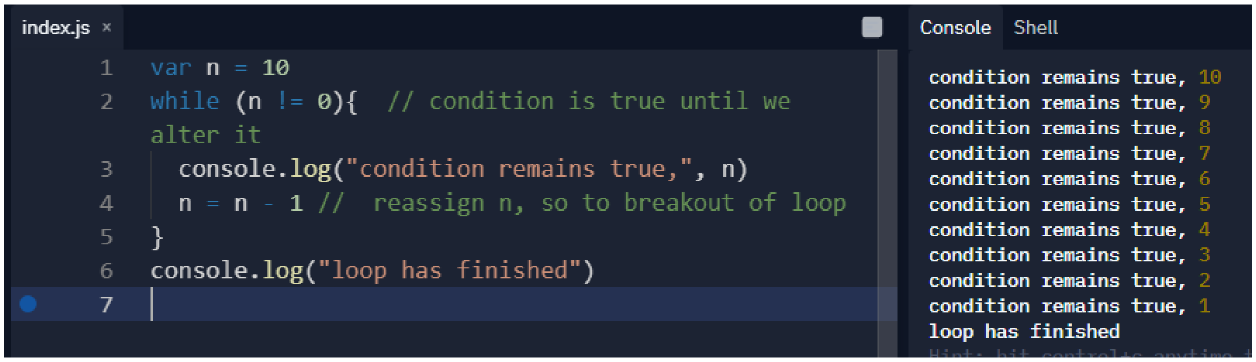
When it comes to while loops, it depends on a specific condition to run. This condition can cause the loop to run infinitely so it's important to have a condition that breaks out of the loop [41].

Fig 19 Python while loop



```python
n = 10
while n != 0: # condition is true until we alter it
  print("condition remains true,", n)
  n = n - 1 # reassign n, so to breakout of loop
print("loop has finished")
```

```
condition remains true, 10
condition remains true, 9
condition remains true, 8
condition remains true, 7
condition remains true, 6
condition remains true, 5
condition remains true, 4
condition remains true, 3
condition remains true, 2
condition remains true, 1
loop has finished
```



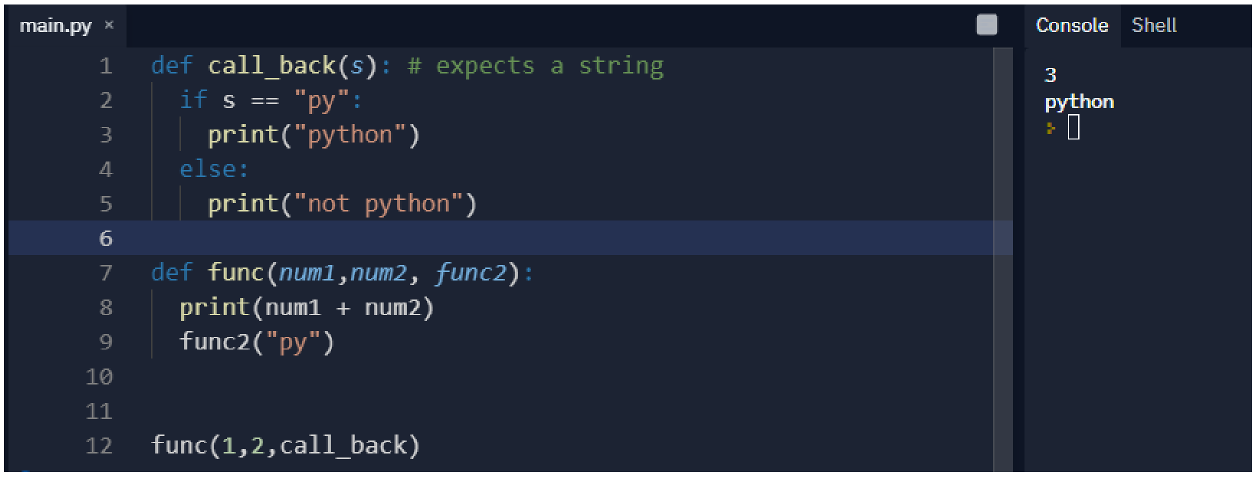
```javascript
var n = 10
while (n != 0){  // condition is true until we alter it
  console.log("condition remains true,", n)
  n = n - 1 //  reassign n, so to breakout of loop
}
console.log("loop has finished")
```

```
condition remains true, 10
condition remains true, 9
condition remains true, 8
condition remains true, 7
condition remains true, 6
condition remains true, 5
condition remains true, 4
condition remains true, 3
condition remains true, 2
condition remains true, 1
loop has finished
```

Fig 20 JavaScript while loop

4.1.8 Functions

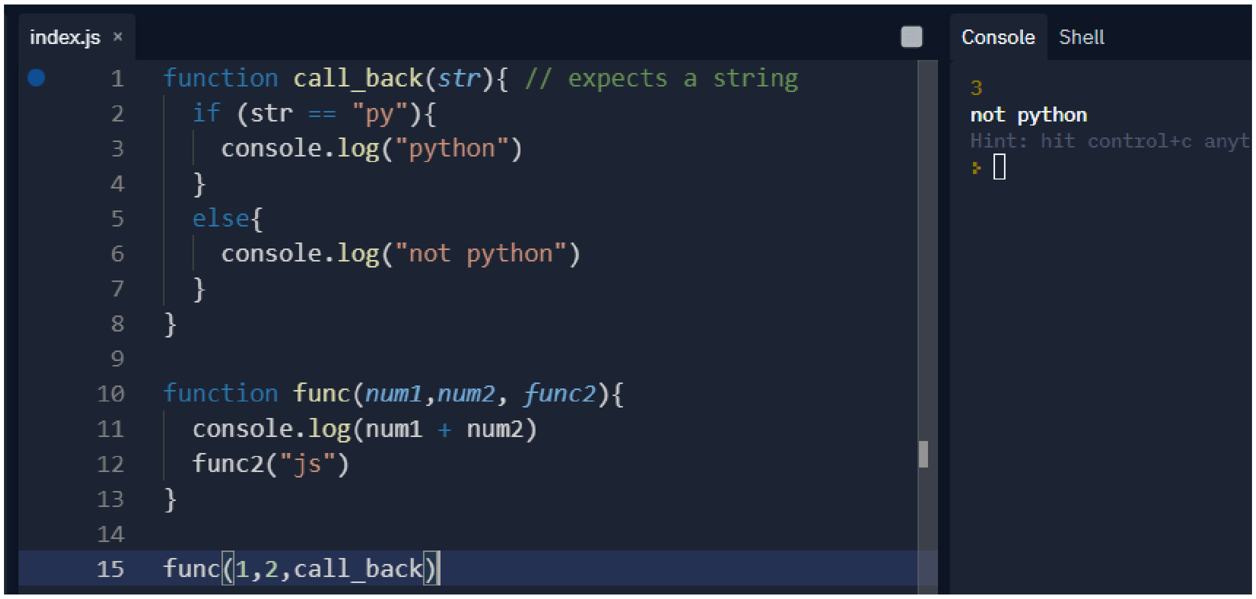
Functions are an important aspect of programming, as they allow us to contain different code into their own block that allows for an operation to happen. This operation could alter a specific value and return something as well, or even set off another function to occu [55]



```python
def call_back(s): # expects a string
  if s == "py":
    print("python")
  else:
    print("not python")

def func(num1,num2, func2):
  print(num1 + num2)
  func2("py")


func(1,2,call_back)
```

```
3
python
```

Fig 21 Python function with callback



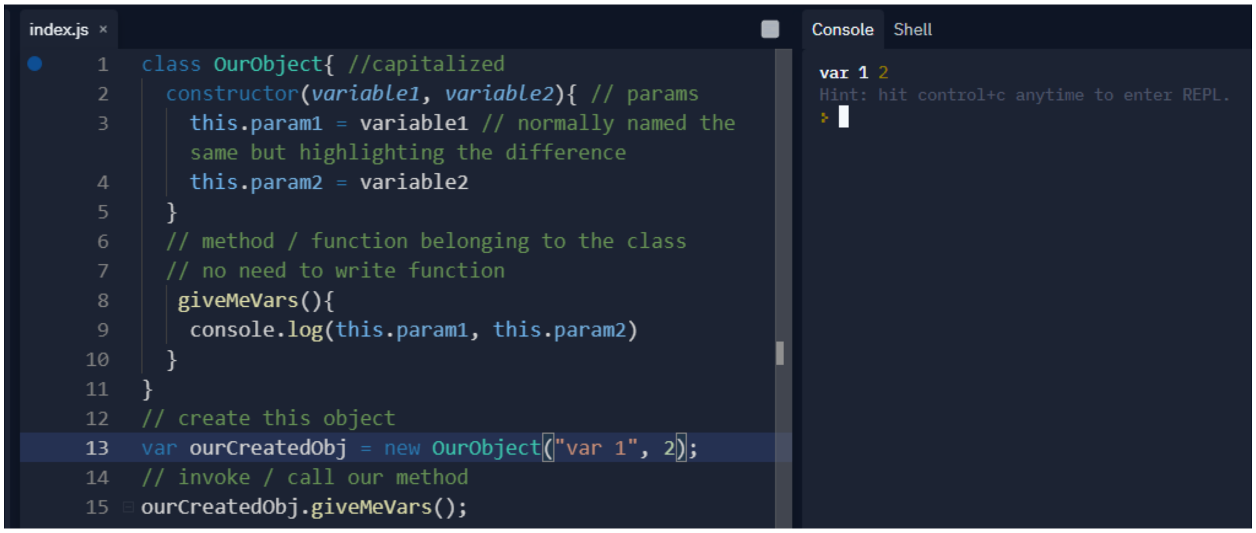
```
function call_back(str){ // expects a string
  if (str == "py"){
    console.log("python")
  }
  else{
    console.log("not python")
  }
}

function func(num1,num2, func2){
  console.log(num1 + num2)
  func2("js")
}

func(1,2,call_back)
```

```
3
not python
Hint: hit control+c anyt
```

Fig 22 JavaScript function with callback

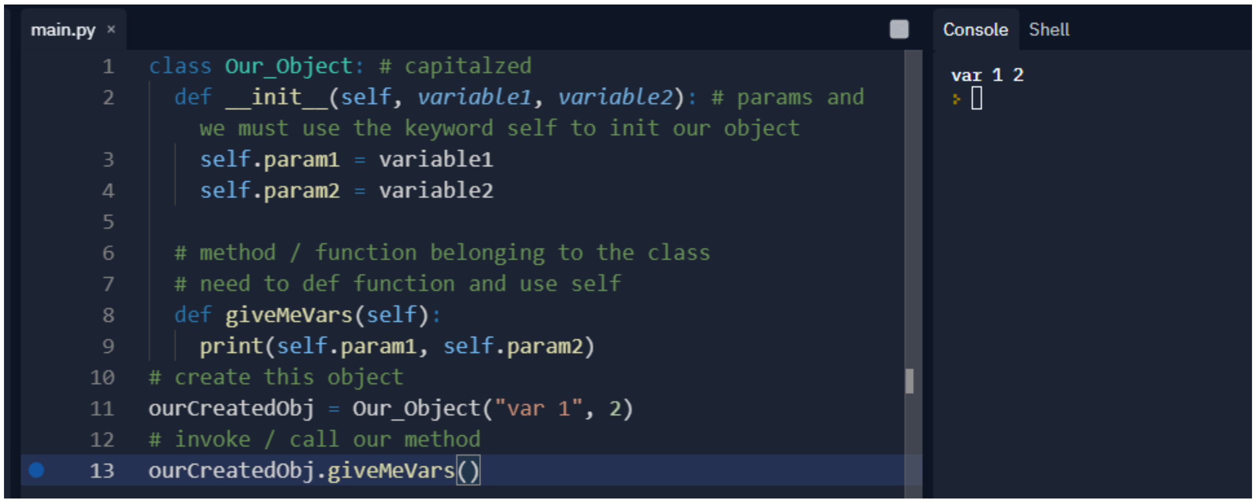
4.1.9 Object-oriented Programming

Both languages support object-oriented programming where a class will define a blueprint for an object. Each class can contain methods which are functions that belong to the class/object. Object oriented programming can go deep, but for a brief example, we will see how to define a class which uses a constructor to construct variables and other parameters that that particular object may need. When the object gets initialized or created for the first time, it will be created with those parameters and take them to construct the object. The object can then contain methods that you can call that behave as functions normally would and can alter that specific object as well [70, 71].



```javascript
class OurObject{ //capitalized
  constructor(variable1, variable2){ // params
    this.param1 = variable1 // normally named the same but highlighting the difference
    this.param2 = variable2
  }
  // method / function belonging to the class
  // no need to write function
   giveMeVars(){
    console.log(this.param1, this.param2)
  }
}
// create this object
var ourCreatedObj = new OurObject("var 1", 2);
// invoke / call our method
ourCreatedObj.giveMeVars();
```

```
var 1 2
Hint: hit control+c anytime to enter REPL.
›
```



```python
class Our_Object: # capitalzed
  def __init__(self, variable1, variable2): # params and we must use the keyword self to init our object
    self.param1 = variable1
    self.param2 = variable2

  # method / function belonging to the class
  # need to def function and use self
  def giveMeVars(self):
    print(self.param1, self.param2)
# create this object
ourCreatedObj = Our_Object("var 1", 2)
# invoke / call our method
ourCreatedObj.giveMeVars()
```

```
var 1 2
›
```

Fig 23, 24 JavaScript/Python class and object creation

4.1.10 Advanced Concepts

JavaScript and Python can get pretty advanced with the improvements in both languages especially now with both supporting interesting features such as asynchronous programming and promises. Promises arose with asynchronous programming and represents the eventual completion of an operation. It can be in three states, pending, fulfilled or rejected, depending on that particular asynchronous operation's results[114]. When considering both Python and JavaScript, they are both single threaded languages, but can achieve the illusion of concurrency through asynchronous programming/event loops or Python threading(although not actually real threading) [72].

When working with concurrency in JavaScript, typically what will happen is there will be an event loop in which the JavaScript will run it’s code in the stack synchronously and then send off the parts that are asynchronous or other specific code to be executed in the loop and awaits its value to be retrieved later [73].

The best example to showcase this within JavaScript is when a user clicks a button on a web page. An event listener listening for that particular event will send a message that gets transmitted to the event queue where it awaits the execution code handling that event [73].

In effect, the same idea can be achieved when looking into Python’s Asyncio library [75].

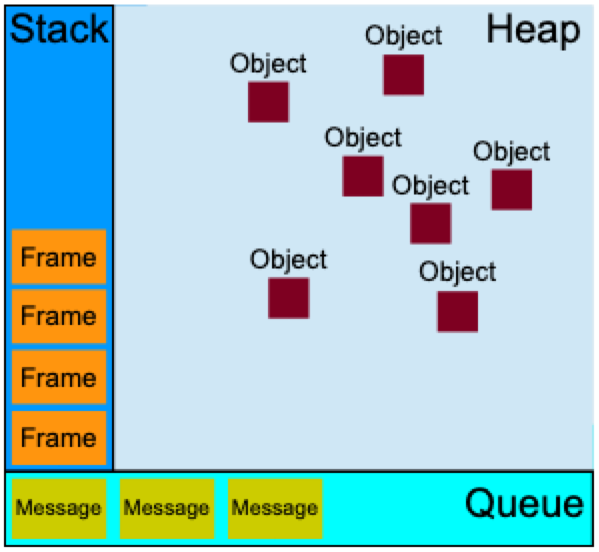


Fig 25. Event loop [73]

The library supports the same features that JavaScript has such as promises and asynchronous tasks.

We could even have our own event loop in which certain code gets executed after a message is sent to the event loop. A message in this instance would be anything from a flag indicating true or false or some event message. We would have the event loop run forever in theory, awaiting for certain actions and executing them in Python's thread [76].

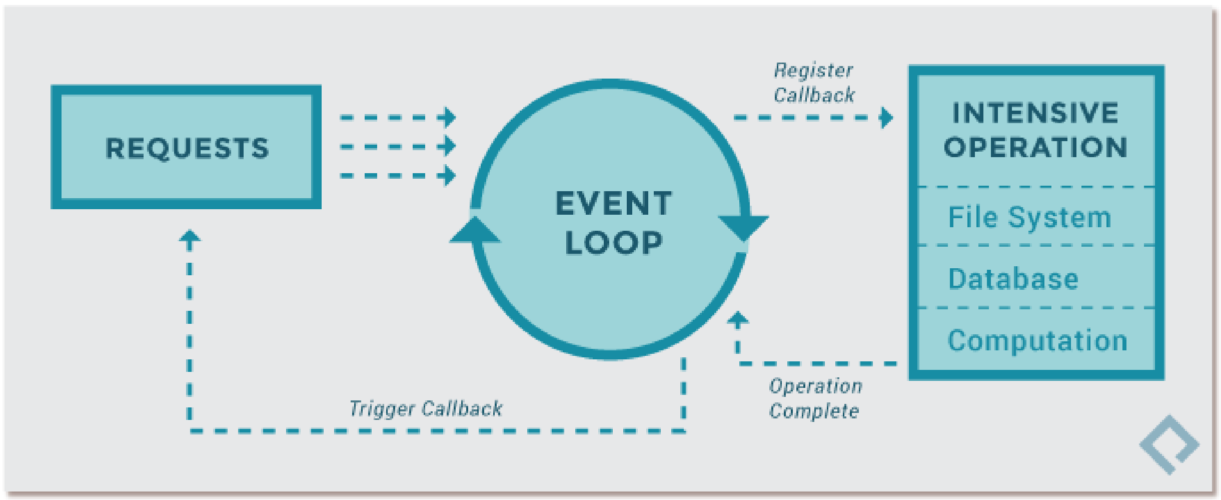


Fig 26. Example of a Python event loop [76]

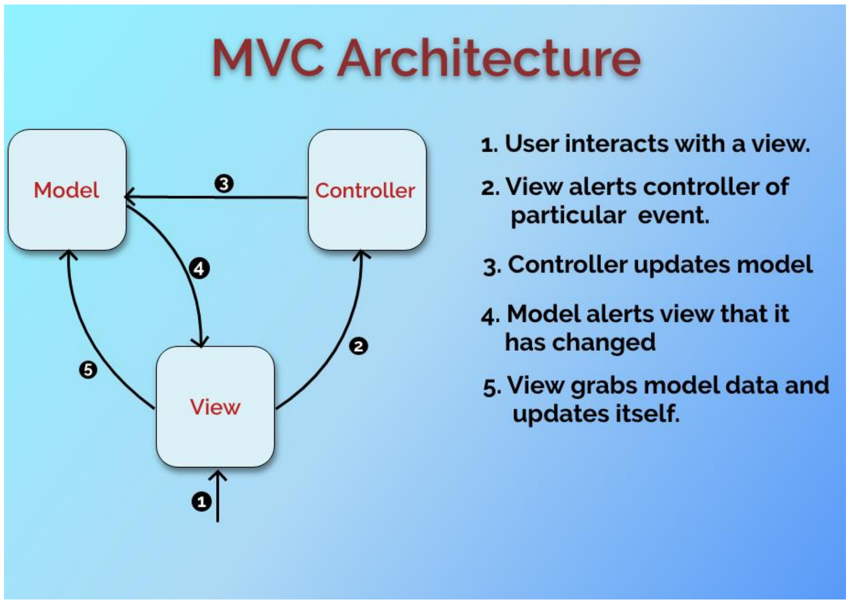
4.2 Practical Libraries and Frameworks

Libraries and frameworks exist to extend capabilities of these programming languages as well as add additional support for extensions into common patterns you may come across. The library will offer functions or methods or even classes which you can call or extend to use in an effort to abstract away a ton of overhead. For example Express.js, a popular JavaScript library built on top of and for Node.js exists as a fast way to launch a JavaScript server without much knowledge into what's happening underneath the hood [77].

When it comes to working with Python you may want to dirty your hands with machine learning or data analysis. When it comes to these tasks, we can use libraries such as TensorFlow or Numpy [57]. Python has an extensive number of libraries including web applications such as Flask for fast lightweight development of web applications or Django which comes with a whole suite of tools [58].

JavaScript is no stranger to extensive libraries and frameworks either. When it comes to working with data analysis or machine learning, it has its own implementation of for example TensorFlow. JavaScript also uses a lot of animations and also has libraries supporting data visualization such as dynamic charts or graphs in libraries such as Chart.js or D3.js[59].

JavaScript has always been a language of the web, so naturally web frameworks and libraries were built on top of this language. You can build full stack applications using just JavaScript with either a full stack framework or a combination of a front-end JavaScript library with a JavaScript back-end. Some examples of popular frameworks are Angular, created by Google and React, a library created by Meta (formally Facebook) [60].

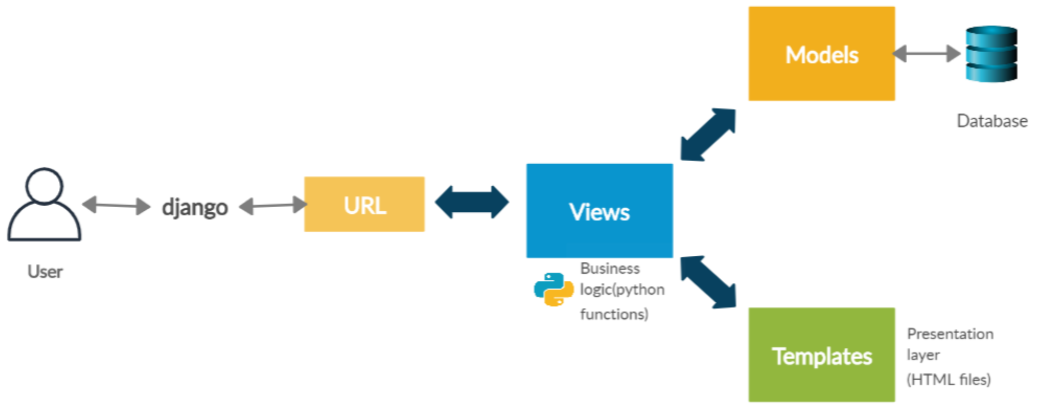


1. User interacts with a view.
2. View alerts controller of particular event.
3. Controller updates model
4. Model alerts view that it has changed
5. View grabs model data and updates itself.

Fig 27 Example of MVC [110]

4.2.1 Django

Django is a very popular framework built on top of Python that loosely follows the MVC architecture [2]. Django abstracts away a lot of overhead that occurs during development of web applications. For example, Django comes with an object relational mapping of its models to a database that you specify. This allows for ease of creation when dealing with new data models and methods pertaining to altering or viewing them [79]. Django is loosely based on MVC since it has templates which are akin to its views and views which act as controllers that handle the requests that come in and what to render.

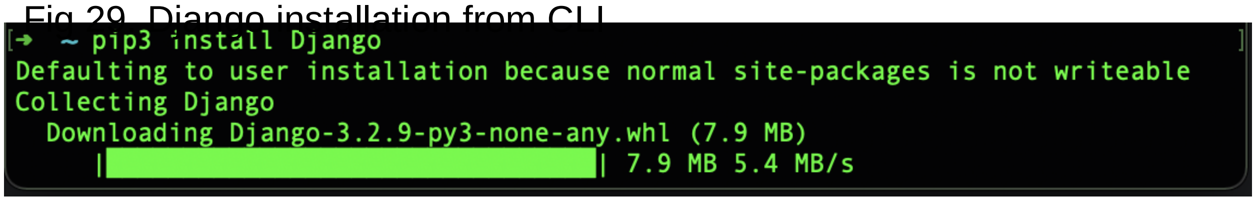

User
django
URL
Views
Business logic(python functions)
Models
Database
Templates
Presentation layer (HTML files)

4.2.1.1 Installation

Fig 28 Example of MVT [1

To get started with Django, make sure to download the latest version of Python and PIP [80]. You should download PIP to help with managing Python packages, and to download Django to it's own virtual environment so as not to mix Python packages up. For installation refer to the site's documentation as it shows a very easy and thorough set up. To get started we simply use PIP to install Django. (Recommended to set up a virtual environment and install Django inside it)

Fig 29. Django installation from CLI

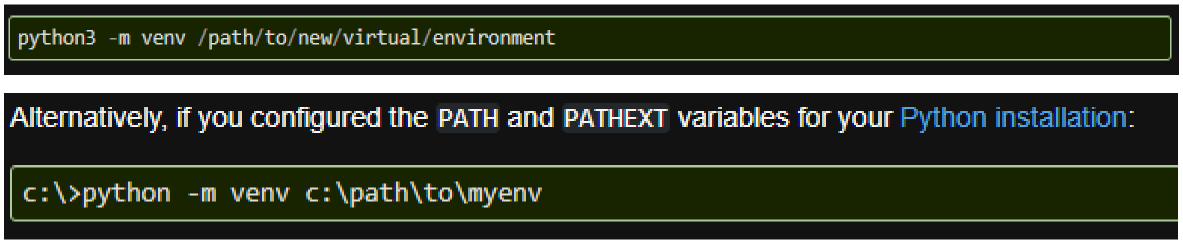


```
➜  ~ pip3 install Django
Defaulting to user installation because normal site-packages is not writeable
Collecting Django
  Downloading Django-3.2.9-py3-none-any.whl (7.9 MB)
     |████████████████████████████████| 7.9 MB 5.4 MB/s
```

4.2.1.2 Virtual environments

We will be using a virtual environment to manage our packages for Django and Python. A virtual environment is just a wrapper to manage our programming environments for our packages on different projects [81]. The best place to start in terms of learning about virtual envs is the docs. https://docs.Python.org/3/tutorial/venv.html

Example of making a virtual environment with a defined folder.

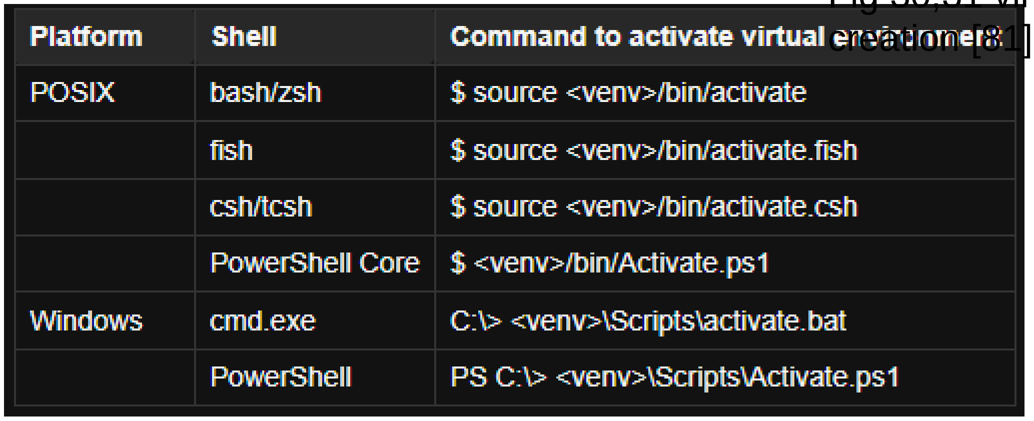


```
python3 -m venv /path/to/new/virtual/environment
```

Alternatively, if you configured the PATH and PATHEXT variables for your Python installation:

```
c:\>python -m venv c:\path\to\myenv
```

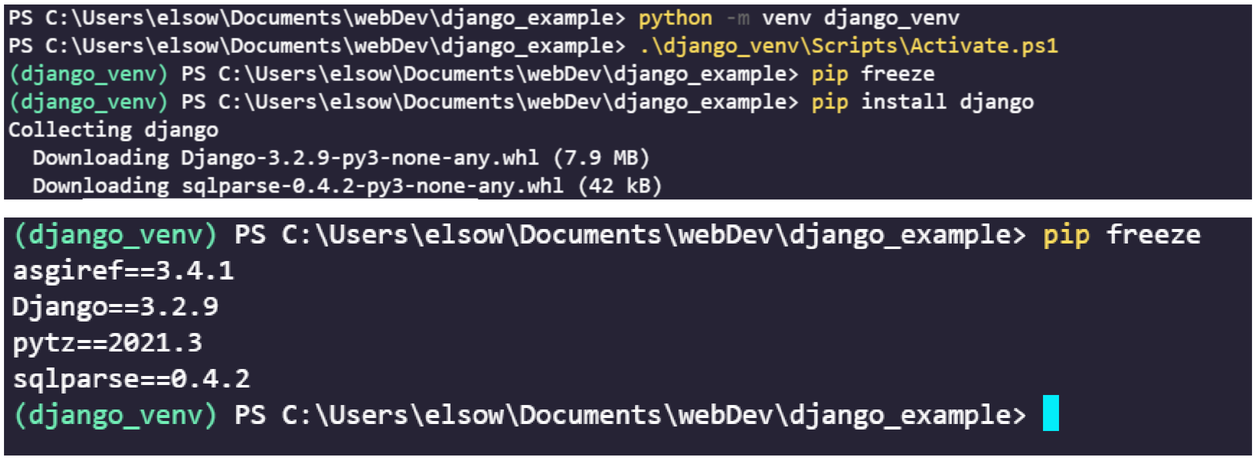
Fig 30,31 virtual environment creation [81]



Platform	Shell	Command to activate virtual environment
POSIX	bash/zsh	$ source <venv>/bin/activate
	fish	$ source <venv>/bin/activate.fish
	csh/tcsh	$ source <venv>/bin/activate.csh
	PowerShell Core	$ <venv>/bin/Activate.ps1
Windows	cmd.exe	C:\> <venv>\Scripts\activate.bat
	PowerShell	PS C:\> <venv>\Scripts\Activate.ps1

Fig 32 virtual environment activation [112]

Example of making a virtual environment , activating it, installing Django from PIP and viewing packages installed for our virtual environment.



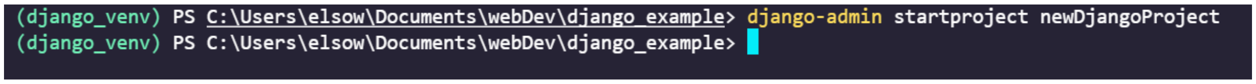
```
PS C:\Users\elsow\Documents\webDev\django_example> python -m venv django_venv
PS C:\Users\elsow\Documents\webDev\django_example> .\django_venv\Scripts\Activate.ps1
(django_venv) PS C:\Users\elsow\Documents\webDev\django_example> pip freeze
(django_venv) PS C:\Users\elsow\Documents\webDev\django_example> pip install django
Collecting django
  Downloading Django-3.2.9-py3-none-any.whl (7.9 MB)
  Downloading sqlparse-0.4.2-py3-none-any.whl (42 kB)
```

```
(django_venv) PS C:\Users\elsow\Documents\webDev\django_example> pip freeze
asgiref==3.4.1
Django==3.2.9
pytz==2021.3
sqlparse==0.4.2
(django_venv) PS C:\Users\elsow\Documents\webDev\django_example>
```

Fig 33,34. Example of virtual environment and commands

4.2.1.3 Install Django

With Django installed in our virtual environment, we now have access to a tool that Django provides and that is the Django-admin tool [82]. The way it works is similar to how we call functions in programming, where the creators of Django decided to extend certain functionalities out of this file and by calling it, we can use different functions and tools that they abstracted away from us to make developing much easier. For example, to start a Django project type in the following.

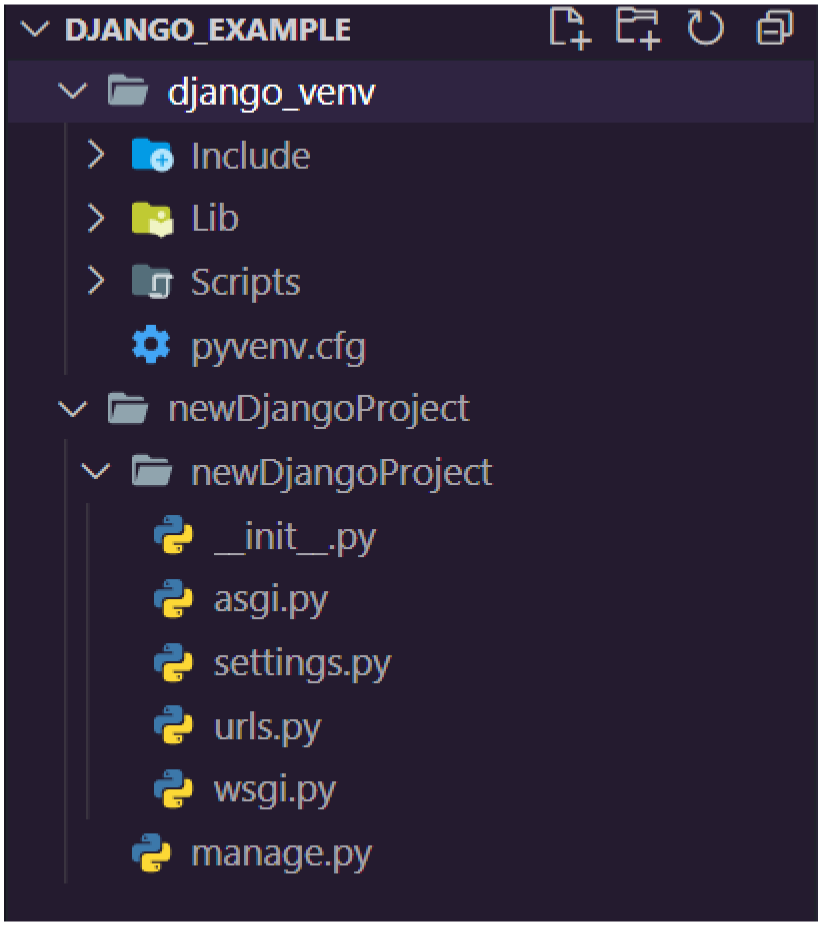


```
(django_venv) PS C:\Users\elsow\Documents\webDev\django_example> django-admin startproject newDjangoProject
(django_venv) PS C:\Users\elsow\Documents\webDev\django_example>
```

Fig 35. Starting a Django Project

With this simple line, we get a new folder that contains various Python files to get us started.

- Outer folder container - Container for your project
- Manage.py - Command line utility that extends abstracted tools for Django
- Init.py - Tells Python this is a Python package
- Settings.py - Settings for your Django Project
- Urls.py - Urls for your site
- asgi/wsgi - deployment files for specific web servers



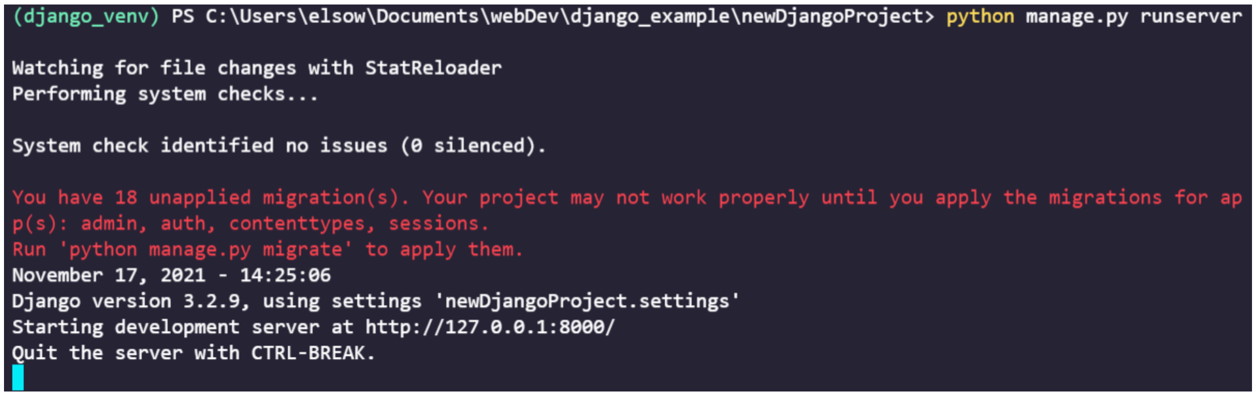
```
DJANGO_EXAMPLE
  django_venv
    > Include
    > Lib
    > Scripts
      pyvenv.cfg
  newDjangoProject
    newDjangoProject
      __init__.py
      asgi.py
      settings.py
      urls.py
      wsgi.py
    manage.py
```

Fig 36. Django folder structure

Django comes with a prebuilt local server and makes it very easy for us to see changes in our local browser [83]. When we go to deploy we need to handle a few more tasks but it’s relatively easy.

To quickly view the page Django gives us with its documentation (highly recommended) run the Django-admin utility tool and give it the param of runserver and optionally provide a port number.

Python manage.py runserver 8080



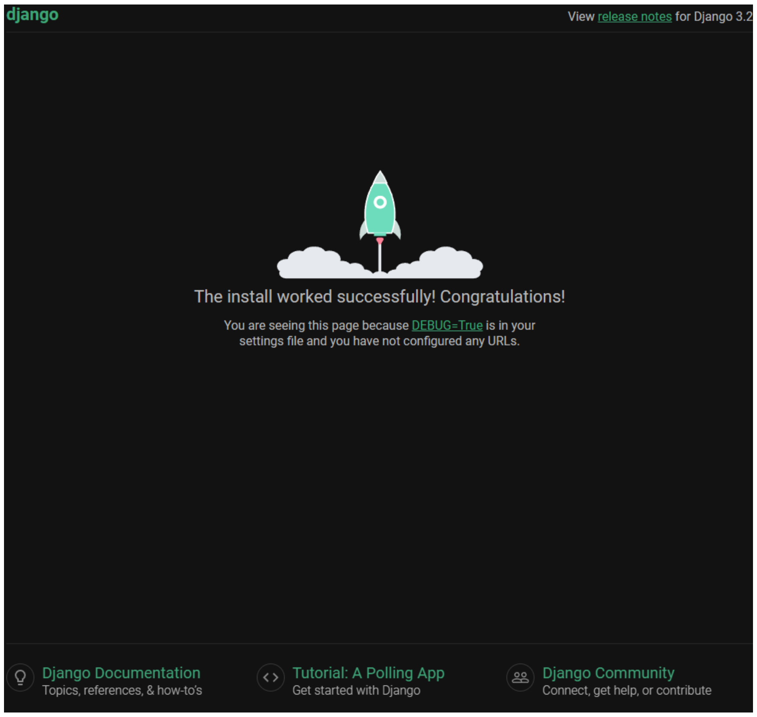
```
(django_venv) PS C:\Users\elsow\Documents\webDev\django_example\newDjangoProject> python manage.py runserver

Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced).

You have 18 unapplied migration(s). Your project may not work properly until you apply the migrations for ap
p(s): admin, auth, contenttypes, sessions.
Run 'python manage.py migrate' to apply them.
November 17, 2021 - 14:25:06
Django version 3.2.9, using settings 'newDjangoProject.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.
```

Fig 37. Django development server

(Migrations have to do with how Django keeps track of database changes, by default some apps use a database that comes with Django, so that is why it mentions the unapplied migrations [83, 84]). The page you should see is on the following page.

Fig 38 Django Sample Page

4.1.2.4 Django projects and apps


django
View release notes for Django 3.2
Django Documentation
Topics, references, & how-to's
Tutorial: A Polling App
Get started with Django
Django Community
Connect, get help, or contribute

The install worked successfully! Congratulations!

You are seeing this page because DEBUG=True is in your settings file and you have not configured any URLs.

When creating a Django project, it can contain multiple apps. An app is simply a web application in which it should be its own component that does its own thing. An example would be a blog or a shop or something similar. When talking about a Django project, it encompasses multiple apps and for example any site could contain a blog app and an e-commerce app. Apps can also belong to multiple projects. The same blog app could be used across various Django projects [83, 85].

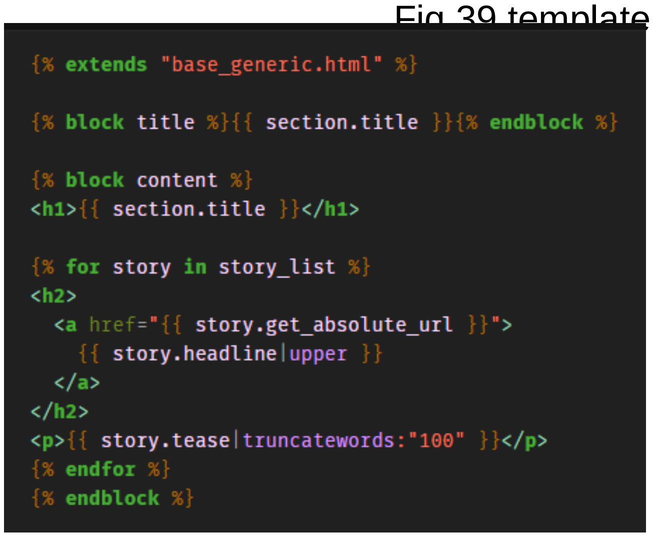
4.1.2.5 Views

Django views are similar to controllers so it could be a bit misleading. Django defines its views as a class that takes in a web request(get, post, put etc) and returns a response [83, 86]. This response could be a template to where a web page is viewed, or it could be for example an authentication response sending back a token.

4.2.1.6 Templates

In order to display dynamic html to the user, Django utilizes templates to insert dynamic Python code into html using Django syntax.
An example can be taken directly from the Django projects documentation. Here we see the template even extended html and also using loops inside of a {% code here %} block [87]

Fig 39 template syntax [87]



```
{% extends "base_generic.html" %}

{% block title %}{{ section.title }}{% endblock %}

{% block content %}
<h1>{{ section.title }}</h1>

{% for story in story_list %}
<h2>
  <a href="{{ story.get_absolute_url }}">
    {{ story.headline|upper }}
  </a>
</h2>
<p>{{ story.tease|truncatewords:"100" }}</p>
{% endfor %}
{% endblock %}
```

4.2.1.7 Models

We will briefly discuss and go over models in Django as they are a very nice feature. Models have to do with databasing and Django actually makes it easy to get by without knowing much of what's going on behind the scenes in terms of the database and queries [79].
The best way to connect to a database for demo purposes would be to use a local database of MySql for example, or there exists remote ones free and paid that allow you to get a database remote connection string which you give to Django to connect to. Django makes it easy to use CRUD operations on your models similar to how you would interface with an object [79] and allows for adding functions and other abstractions to handle your models [79,88].

4.2.1.8 CRUD

CRUD is an acronym that stands for create, read, update and delete.

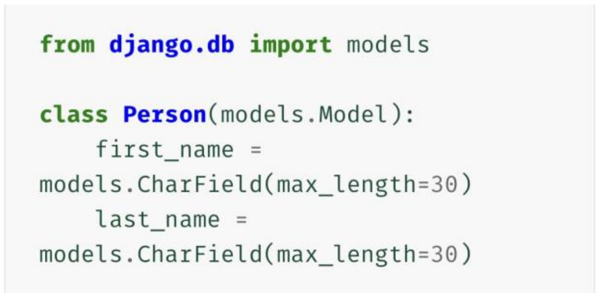
- Create - create a new resource
- Read - read a resource
- Update - update a resource

- Delete - a resource if it exists

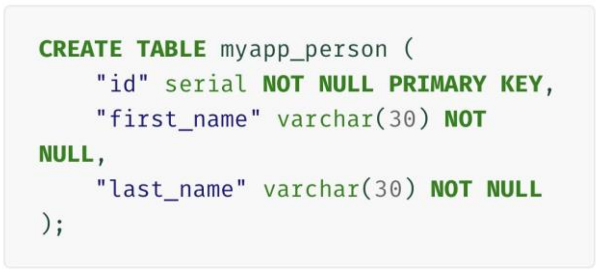
When working with Django models you will certainly have to deal with CRUD operations. These are common operations that any database should interface with. The way this works is that databases have their own operations for doing these operations while each programming language has its own syntax. An interface exists to bridge this gap and allows for you to speak to this database in your favorite programming language. When dealing with Django, it further abstracts it with the use of models and CRUD operations. What Django does is allows you to connect to your database and, on its own, interfaces with the database with its own set of functions allowing you to use Python code to handle database functions [88].

4.2.1.9 Django models under the hood

Here is an example of how Django handles the conversion of its model to a table with columns in



```
from django.db import models

class Person(models.Model):
    first_name =
models.CharField(max_length=30)
    last_name =
models.CharField(max_length=30)
```



```
CREATE TABLE myapp_person (
    "id" serial NOT NULL PRIMARY KEY,
    "first_name" varchar(30) NOT
NULL,
    "last_name" varchar(30) NOT NULL
);
```

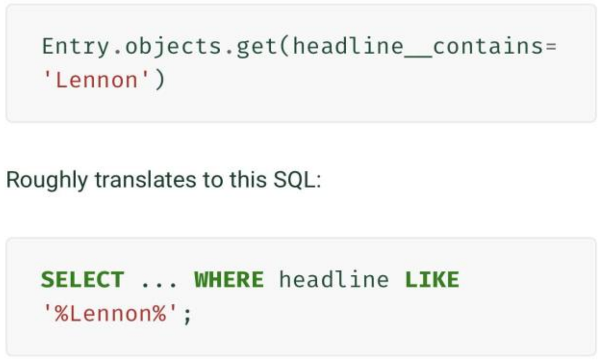
a specified database. You import the model
class and create your data fields which get translated to its equivalent in query language. Models also allow for easy access of CRUD operations such as getting, deleting or updating a model.

4.2.1.10 What is REST?

REST stands for Representational State Transfer which is a standardized way for communicating data between applications [101]. When an application conforms to REST it becomes RESTful.

For web applications, there are common signals we send to the server to communicate what we are intending to do.

Similar to CRUD we expose certain endpoints that allow for crud operations or other restful methods.



```
Entry.objects.get(headline__contains=
'Lennon')
```

Roughly translates to this SQL:

```
SELECT ... WHERE headline LIKE
'%Lennon%';
```

Fig 40,41,42 models under the hood [79]

- Get - Retrieve a resource/ get data from an API endpoint / view a site
- Post - Create a new resource on the server/database
- Put - Replaces resource if exists otherwise creates it
- Delete - Deletes request resource if exists
- Patch - Update resource if exists

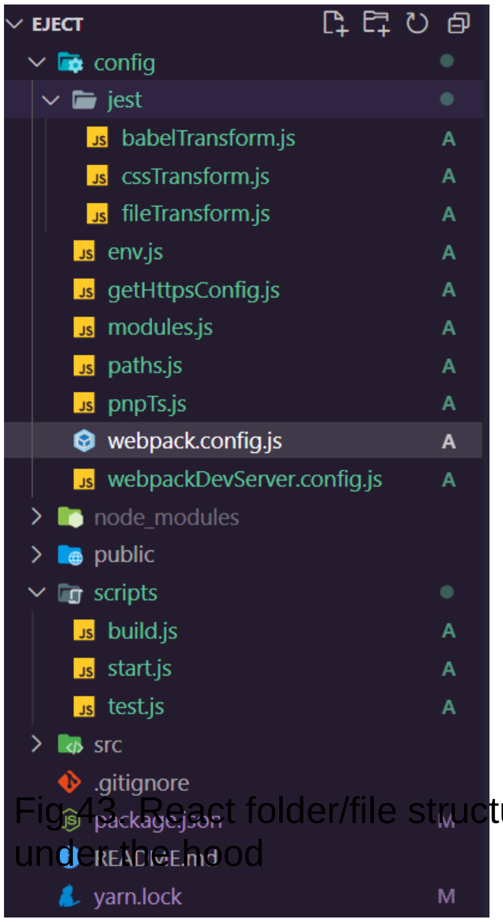
4.2.1.11 Django REST Framework

Since REST is standardized, many libraries build these features of communication to a server where they extend an interface for you to use that allows for interpreting certain requests to your applications or servers and what to do when that request comes in. The Django Rest framework extends this abstraction and handles this logic while letting you define what happens on a GET request or a POST request [89].

4.2.2 React (RQ3a)

React is a user interface library built on top of JavaScript by meta (formally Facebook) for creating composable components that can be reused and combined to create complex sites [3].

React also has state, meaning it allows for data to be contained within its components where any changes re render the component.



```
EJECT
  config
    jest
      babelTransform.js        A
      cssTransform.js          A
      fileTransform.js         A
    env.js                     A
    getHttpsConfig.js          A
    modules.js                 A
    paths.js                   A
    pnpTs.js                   A
    webpack.config.js          A
    webpackDevServer.config.js A
  node_modules
  public
  scripts
    build.js                   A
    start.js                   A
    test.js                    A
  src
  .gitignore
  package.json                 M
  README.md
  yarn.lock                    M
```

Fig 43. React folder/file structure under the hood

Some of the brief features are :

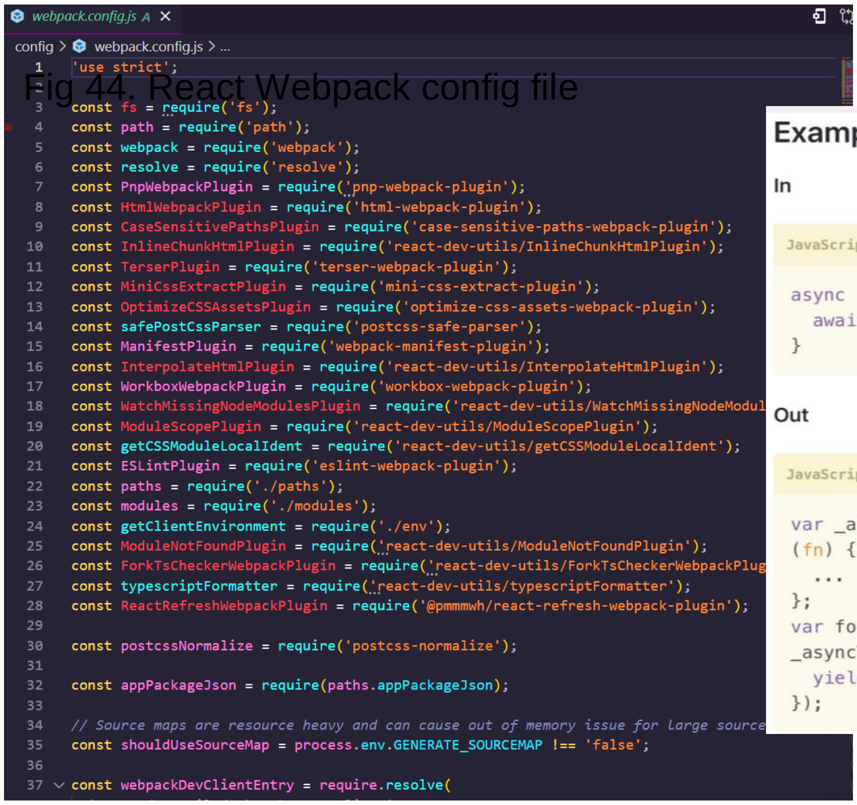
- Has its own syntax (JSX) while also supporting TypeScript and regular JavaScript
- Virtual DOM
- State
- Hooks
- Components

React, although not a framework, provides beautiful features out of the box that allow for rapid development [90]. What React hides is a lot of old patterns that were used in the past when constructing a web application, such as transpiling the JavaScript while minifying it and the state changes of your application. The behind the scenes underneath the hood can be viewed if you eject from a React application which you will find the configuration of all these files including other behind the scenes configuration. What you'll find is some very complex stuff.

4.2.2.1 Babel /Webpack

Because JavaScript evolved so much, a lot of new features came out that are NOT backwards compatible. For example asynchronous programming and certain syntax that allows for different code bases cause incompatible JavaScript when migrating your application to the web for various browser versions to interpret and run their engine on [99]. What Babel does is quite literally recompiles all your JavaScript into a compatible backwards version and Webpack minifies and shrinks it and repackages it. Webpack also handles CSS files as well. The beauty of this happens behind the scenes where React takes care of it [99]. You can see an example of the folder given labeled scripts that React uses to run the development server as well as to build and test your application in the example ejected folder structure.

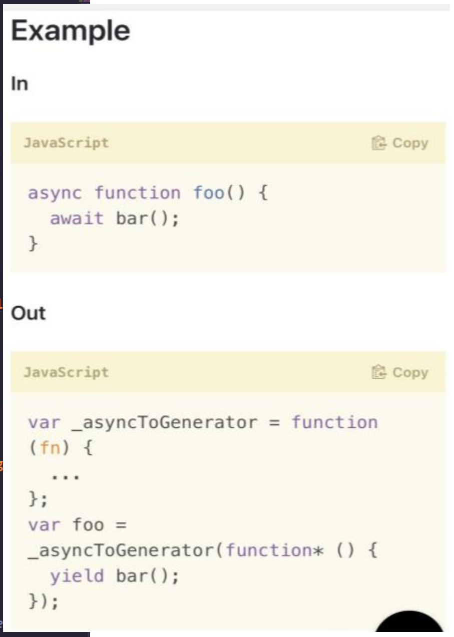
When working with older versions of React you'd have to configure this yourself. When creating a React app from scratch, much has
evolved and the stresses of making this configuration file are no longer something to worry about.


webpack.config.js A ×

config > webpack.config.js > ...

```
'use strict';

const fs = require('fs');
const path = require('path');
const webpack = require('webpack');
const resolve = require('resolve');
const PnpWebpackPlugin = require('pnp-webpack-plugin');
const HtmlWebpackPlugin = require('html-webpack-plugin');
const CaseSensitivePathsPlugin = require('case-sensitive-paths-webpack-plugin');
const InlineChunkHtmlPlugin = require('react-dev-utils/InlineChunkHtmlPlugin');
const TerserPlugin = require('terser-webpack-plugin');
const MiniCssExtractPlugin = require('mini-css-extract-plugin');
const OptimizeCSSAssetsPlugin = require('optimize-css-assets-webpack-plugin');
const safePostCssParser = require('postcss-safe-parser');
const ManifestPlugin = require('webpack-manifest-plugin');
const InterpolateHtmlPlugin = require('react-dev-utils/InterpolateHtmlPlugin');
const WorkboxWebpackPlugin = require('workbox-webpack-plugin');
const WatchMissingNodeModulesPlugin = require('react-dev-utils/WatchMissingNodeModul
const ModuleScopePlugin = require('react-dev-utils/ModuleScopePlugin');
const getCSSModuleLocalIdent = require('react-dev-utils/getCSSModuleLocalIdent');
const ESLintPlugin = require('eslint-webpack-plugin');
const paths = require('./paths');
const modules = require('./modules');
const getClientEnvironment = require('./env');
const ModuleNotFoundPlugin = require('react-dev-utils/ModuleNotFoundPlugin');
const ForkTsCheckerWebpackPlugin = require('react-dev-utils/ForkTsCheckerWebpackPlug
const typescriptFormatter = require('react-dev-utils/typescriptFormatter');
const ReactRefreshWebpackPlugin = require('@pmmmwh/react-refresh-webpack-plugin');

const postcssNormalize = require('postcss-normalize');

const appPackageJson = require(paths.appPackageJson);

// Source maps are resource heavy and can cause out of memory issue for large source
const shouldUseSourceMap = process.env.GENERATE_SOURCEMAP !== 'false';

const webpackDevClientEntry = require.resolve(
```

Fig 44. React Webpack config file


Example

In

JavaScript

```
async function foo() {
  await bar();
}
```

Out

JavaScript

```
var _asyncToGenerator = function
(fn) {
  ...
};
var foo =
_asyncToGenerator(function* () {
  yield bar();
});
```

Fig 45 Babel plugin conversion [113]

4.2.2.2 Node Modules

These can be thought of as libraries containing functions that your app will interface with. React uses Node as it's runtime and will need additional modules or libraries for extra functionality. When we go to include a new library and import it, we will first have to install it. We use node's package manager for this and React is smart enough to put it into your package file and create a package lock file combined with a folder labeled node_modules where these libraries or modules live [99].

4.2.2.3 Node js.

Node is a runtime environment for JavaScript. It has a package manager as well referred to as npm and allows for installing packages that can be placed in your node modules file [102]. When working with Node.js itself, you typically will start with an index file and tell Node to control the event handling of requests to your app/server and how to appropriately respond. In using React, Node handles the starting and stopping of your application, but React typically hides this via its own scripts.

4.2.2.4 Why React?

React was created as a necessity to handle the large code base at Facebook at the time. There were many components that needed updating which were separate from the overall application and so React was created to solve this issue [96, 97].

When React was created, it took a while for it to become popular. An active open source community has led to many rapid improvements and advancements ranging from changing their transpiler to Babel's engine and extending and abstracting React features such as state so that functions can be used to create components rather than classes that followed the pattern of extending the React component class [97].

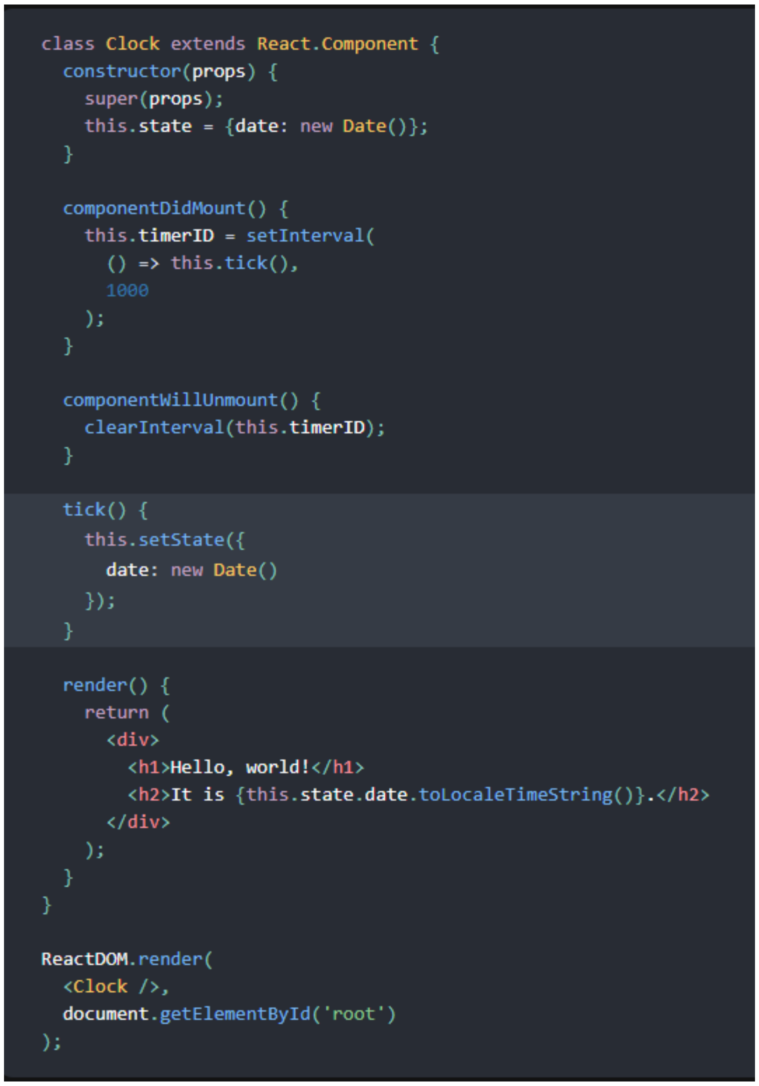
When one looks into React for knowledge, they may come across much old outdated content. React has changed significantly and although many code bases exist that adopted the older habits of react, many companies are now realizing the power behind modern React features such as hooks and other aspects.

4.2.2.5 Virtual dom

React created their own API to interface with the actual dom or UI of the browser. What they did is use their own version called ReactDOM which syncs with the actual DOM. In essence what they do is they'll change the state of the virtual dom and have it appear as though it were what is called a single page application[98].

4.2.2.6 Hooks

After the creation of React, there were class based components where adding state was achieved by extending the React Component class, calling a render function, and setting the state of the class by assigning it to a variable called state [93]. Now the syntax is a little cleaner and React decided to extend what they call hooks which are functions that hook into these specific features of React [92, 94]. When using React versions 16.8+ you can use functional components instead of class based components and use React's hooks to set your state and also hook into the lifecycle methods [92]. React also allows you to make your own hooks, with a couple of rules that exist to make it functional. The most common hooks used are:



```
class Clock extends React.Component {
  constructor(props) {
    super(props);
    this.state = {date: new Date()};
  }

  componentDidMount() {
    this.timerID = setInterval(
      () => this.tick(),
      1000
    );
  }

  componentWillUnmount() {
    clearInterval(this.timerID);
  }

  tick() {
    this.setState({
      date: new Date()
    });
  }

  render() {
    return (
      <div>
        <h1>Hello, world!</h1>
        <h2>It is {this.state.date.toLocaleTimeString()}.</h2>
      </div>
    );
  }
}

ReactDOM.render(
  <Clock />,
  document.getElementById('root')
);
```

Fig.47 State and lifecycle in class based component [93]

- Use state hook - allows for setting multiple state values for your component
- Use effect hook - allows for hooking into the different life cycle methods of a component. I.E mounting, updating, and unmounting.

4.2.2.7 The lifecycle method

React is notorious for having data associated with its components. In the initial designs of React applications, classes were used that extended the React component class [93]. There were specific functions associated with the lifecycle of these components as well. With hooks, the use Effect hook allows for combining these three lifecycle components into one hook, where we can supply a callback function within the use effect hook that will act as a clean up when the component unmounts.To briefly get an idea of the lifecycle methods and their associated React class lifecycle methods, we will list them below.

- Mounting - When the component first mounts to the virtual DOM.
- Updating - When the component is updating (such as rendering new state)
- Unmounting - Component has been removed from the virtual DOM.

React component functions associated with the lifecycle method:

- componentDidMount() - used with mounting lifecycle
- componentDidUpdate() - used when updating state
- componentWillUnmount() - used when cleaning data and in mourning

4.2.2.8 State

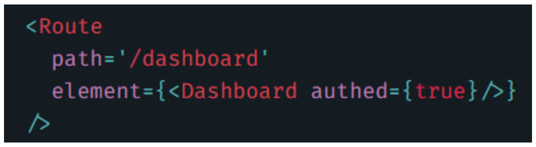
React has state for its components allowing data to be rendered and remembered. We can add and access state by using React’s new hooks and that’s the use state hook.
State has a few important properties as well such as not directly modifying it and having only a set function that handles the mutation of your state. State also flows down between components so a child component could have access to its parents state if the state gets passed [93, 100].

4.2.2.9 Props

Props stand for properties and allow us to pass data between components. This is useful when we may pass state around. We can also pass around objects or functions that callback to the parent component and allow for state to be changed that way [103].

4.2.2.10 Routing

In an effort to get React to render pages fast and gain the name of being a single page application, it uses routing to route components or pages fast and give a smooth experience. You can pass properties through your components this way and render dynamic content depending on what route they hit [104].

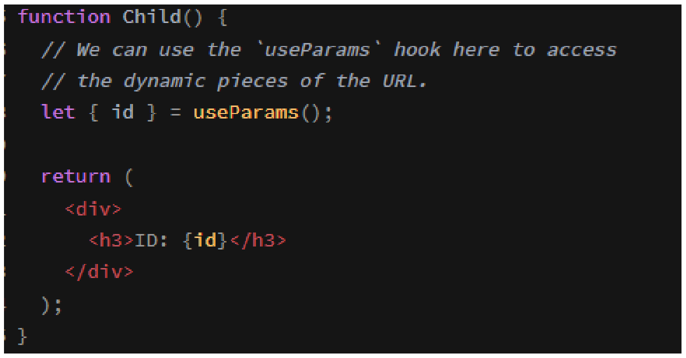


```
<Route
  path='/dashboard'
  element={<Dashboard authed={true}/>}
/>
```

Fig 48 React route [104]

4.2.2.11 Params

In order to effectively use routes, we must have params. In order to navigate dynamically between different pages or render dynamic connect. We can use a hook here that allows us to capture parameters that get passed into the router. We can extract it as well using object destructuring [105].



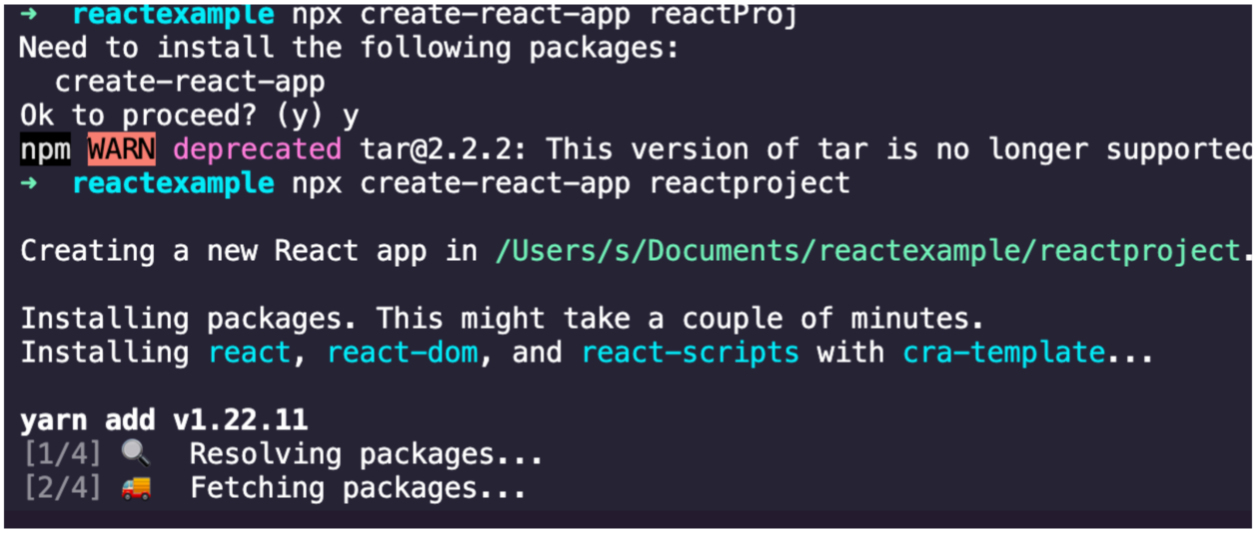
```
function Child() {
  // We can use the `useParams` hook here to access
  // the dynamic pieces of the URL.
  let { id } = useParams();

  return (
    <div>
      <h3>ID: {id}</h3>
    </div>
  );
}
```

Fig 49 use params hook [10

4.2.2.12 Installation and set up

We can use Node’s package executor to start a bootstrapped react project by typing “Npx create react app” in any folder.

Here react uses Yarn package manager to install the necessary files it needs. It also places your folders and files into a structure and gives you features to manage your project such as development to deployment.



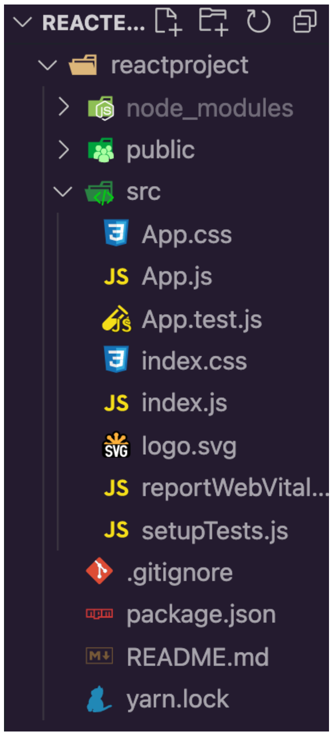
```
➜  reactexample npx create-react-app reactProj
Need to install the following packages:
  create-react-app
Ok to proceed? (y) y
npm WARN deprecated tar@2.2.2: This version of tar is no longer supporte
➜  reactexample npx create-react-app reactproject

Creating a new React app in /Users/s/Documents/reactexample/reactproject.

Installing packages. This might take a couple of minutes.
Installing react, react-dom, and react-scripts with cra-template...

yarn add v1.22.11
[1/4] 🔍  Resolving packages...
[2/4] 🚚  Fetching packages...
```

Fig 50 Creating a React app

Node modules are simply where the packages React and you use are stored. Nothing needs to be done as when you install a package it will go into there so long you install it in the root of the package.

React uses Yarn or npm, so you will typically see a Yarn lock as well as a package.json file which manages your files and versions. Our bootstrap project gives us the following aside from the node_modules and package files:

- **CSS -** All relevant style sheets. React supports scss and there are different methods to go about styling your component or application
- **Public -** Where the public files for your web application will live. Such as HTML, stylesheets or JavaScript, as well as an icon for your site
- **App.js-** entry point for your application where JavaScript is injected or imported and used to display your client side rendered application



```
REACTE...
  reactproject
    node_modules
    public
    src
      App.css
      App.js
      App.test.js
      index.css
      index.js
      logo.svg
      reportWebVital...
      setupTests.js
    .gitignore
    package.json
    README.md
    yarn.lock
```

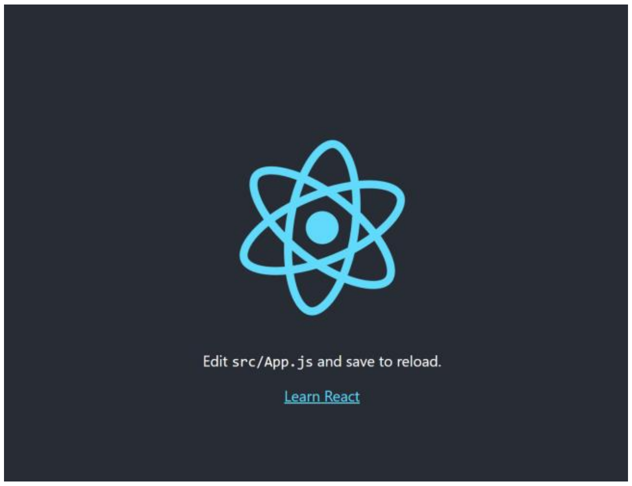
Fig 51.React file/folder structure

- **Test** - Any testing for your main app

4.2.2.13 Development Server

React allows us to view, save, and see our changes immediately with hot reloading. When starting your development server you simply type, npm run start, which tells Node to look into the package.json file and find the script for start, which is linked to the React-scripts library that handles the development server for you. When running your development server you should see the following.

Fig 52. React sample page



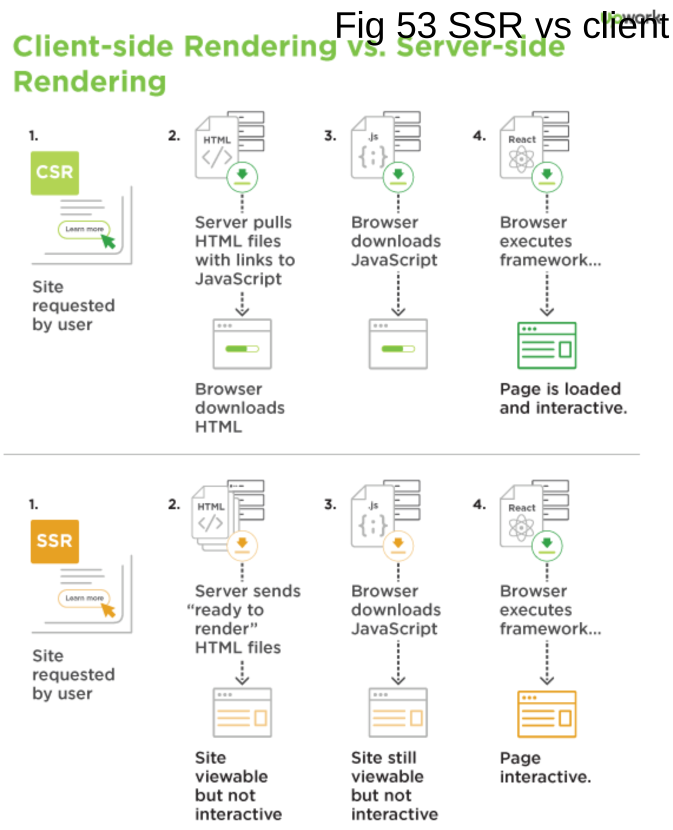
Edit `src/App.js` and save to reload.

Learn React

4.3 SSR vs client Side (RQ4)

When websites initially were displayed, the server would handle the rendering of the html and display it back to the client after the client requests to see it. This is known as server side rendering and has been a limitation of react although react has additional libraries built over it that aid in this. the reason it's a limitation is it affects search engines and the displaying of your site. The search engines can't read the JavaScript that is used in client side rendering and thus sees an empty page and seo is lost [106, 107]

Fig 53 SSR vs client side [107]



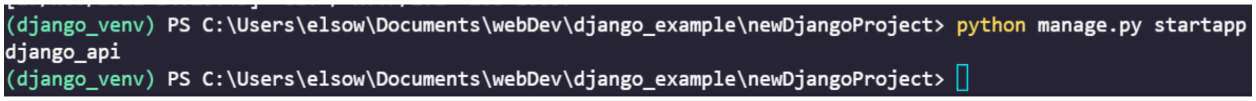
Client-side Rendering vs. Server-side Rendering
1.
CSR
Learn more
Site requested by user
2.
HTML
Server pulls HTML files with links to JavaScript
Browser downloads HTML
3.
Js
Browser downloads JavaScript
4.
React
Browser executes framework...
Page is loaded and interactive.
1.
SSR
Learn more
Site requested by user
2.
HTML
Server sends "ready to render" HTML files
Site viewable but not interactive
3.
Js
Browser downloads JavaScript
Site still viewable but not interactive
4.
React
Browser executes framework...
Page interactive.

4.4 React and Django together

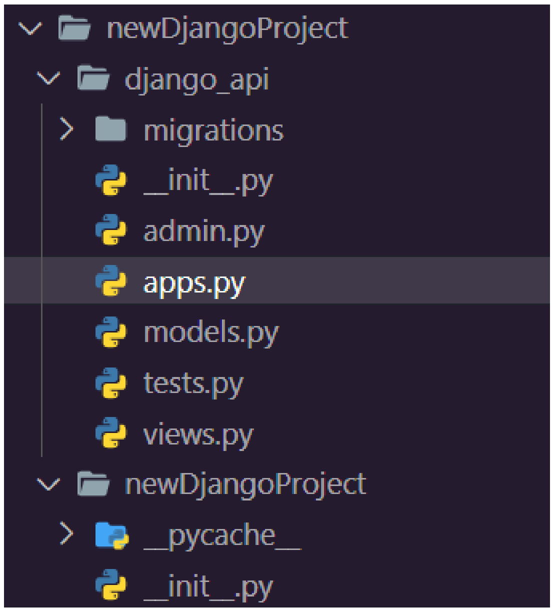
We can actually use React's build tool to build the static files and use Django to serve them. This would in theory combine the two as well as solve the problem of client side rendering.

However, it would be interesting to see a way in which the applications work in their own aspects in a decoupled way as to support better software design. In practicality we could combine the two but for this purpose we will be using Django to mimic an api that sends data in the form of JSON to our front end React application.

To get started we will continue off with our Django project and create a new application which we will call "django_api". Our goal is to send data to the React app we will be creating later. (Remember to view your site use python manage.py runserver)



```
(django_venv) PS C:\Users\elsow\Documents\webDev\django_example\newDjangoProject> python manage.py startapp django_api
(django_venv) PS C:\Users\elsow\Documents\webDev\django_example\newDjangoProject>
```



```
newDjangoProject
  django_api
    migrations
    __init__.py
    admin.py
    apps.py
    models.py
    tests.py
    views.py
  newDjangoProject
    __pycache__
    __init__.py
```

Fig 54,55 . Django new app and file/folder structure

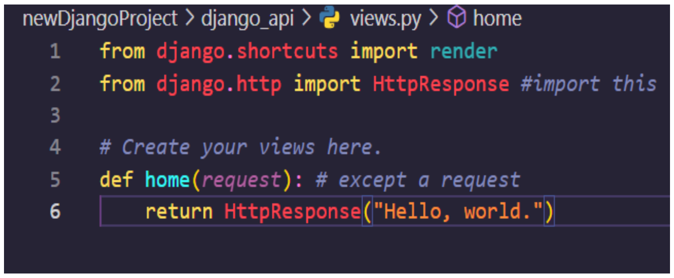
4.4.1 Views configuration

Here we will import the http library from Django and use it to handle a request which we will map to our url. For simplicity's sake, we will be focusing on the GET request.

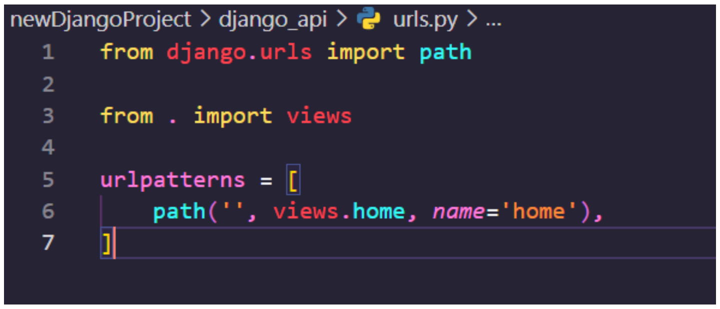
4.4.2 Mapping to url/endpoint

We will be creating a url configuration file to manage our new url where we will link it to our view.

Similar to our main structure where it contains a urls.py file, put one in the new app folder and include the following code.


newDjangoProject > django_api > views.py > home

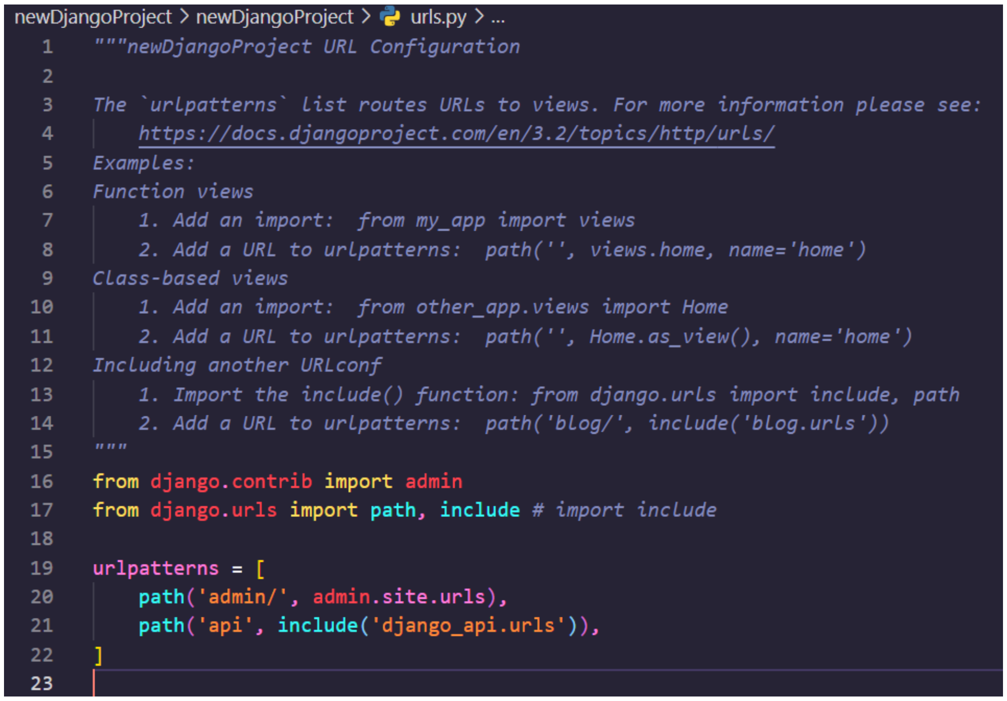
```python
from django.shortcuts import render
from django.http import HttpResponse #import this

# Create your views here.
def home(request): # except a request
    return HttpResponse("Hello, world.")
```


newDjangoProject > django_api > urls.py > ...

```python
from django.urls import path

from . import views

urlpatterns = [
    path('', views.home, name='home'),
]
```

Next we will need to link this to our main urls using an include statement.

If we navigate to localhost and go to /api, we should see the hello world api in action.


newDjangoProject > newDjangoProject > urls.py > ...

```python
"""newDjangoProject URL Configuration

The `urlpatterns` list routes URLs to views. For more information please see:
    https://docs.djangoproject.com/en/3.2/topics/http/urls/
Examples:
Function views
    1. Add an import:  from my_app import views
    2. Add a URL to urlpatterns:  path('', views.home, name='home')
Class-based views
    1. Add an import:  from other_app.views import Home
    2. Add a URL to urlpatterns:  path('', Home.as_view(), name='home')
Including another URLconf
    1. Import the include() function: from django.urls import include, path
    2. Add a URL to urlpatterns:  path('blog/', include('blog.urls'))
"""
from django.contrib import admin
from django.urls import path, include # import include

urlpatterns = [
    path('admin/', admin.site.urls),
    path('api', include('django_api.urls')),
]
```

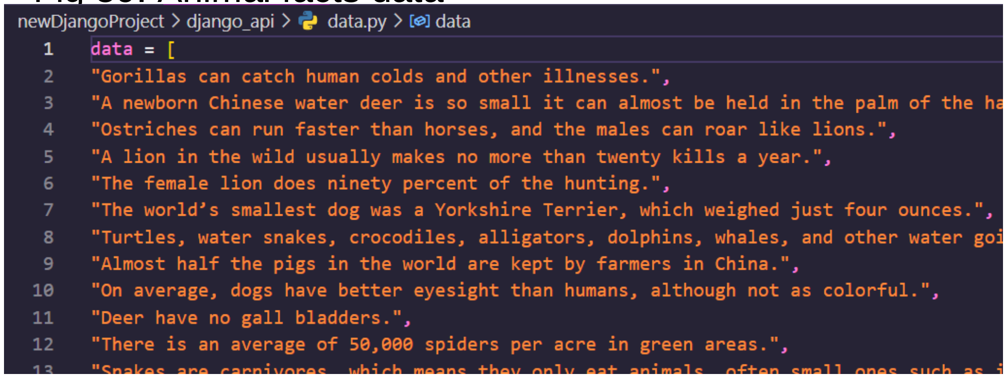
Fig 56, 57, 58.Creating a view and mapping it

4.4.3 Feeding our api data.

We will be using a compiled list of various animal facts that we will serve one at a time from an endpoint.

We will be holding it into a built in data structure of an array or list. Create a file named data.py which will hold an array called data which we will import.

Fig 59. Animal facts data

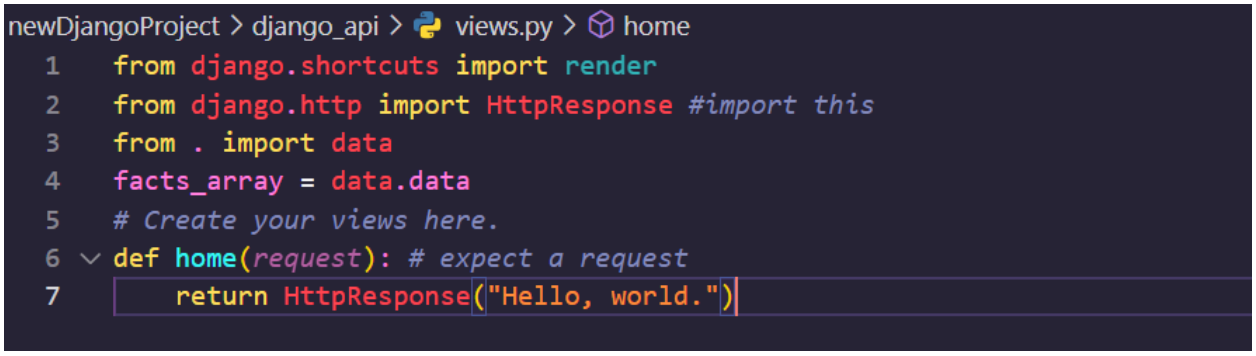


newDjangoProject > django_api > data.py > data

```
data = [
"Gorillas can catch human colds and other illnesses.",
"A newborn Chinese water deer is so small it can almost be held in the palm of the ha
"Ostriches can run faster than horses, and the males can roar like lions.",
"A lion in the wild usually makes no more than twenty kills a year.",
"The female lion does ninety percent of the hunting.",
"The world’s smallest dog was a Yorkshire Terrier, which weighed just four ounces.",
"Turtles, water snakes, crocodiles, alligators, dolphins, whales, and other water goi
"Almost half the pigs in the world are kept by farmers in China.",
"On average, dogs have better eyesight than humans, although not as colorful.",
"Deer have no gall bladders.",
"There is an average of 50,000 spiders per acre in green areas.",
"Snakes are carnivores, which means they only eat animals, often small ones such as i
```

4.4.4 Import the data

Now in order to import the data follow this convention where we assign a variable the overall array.



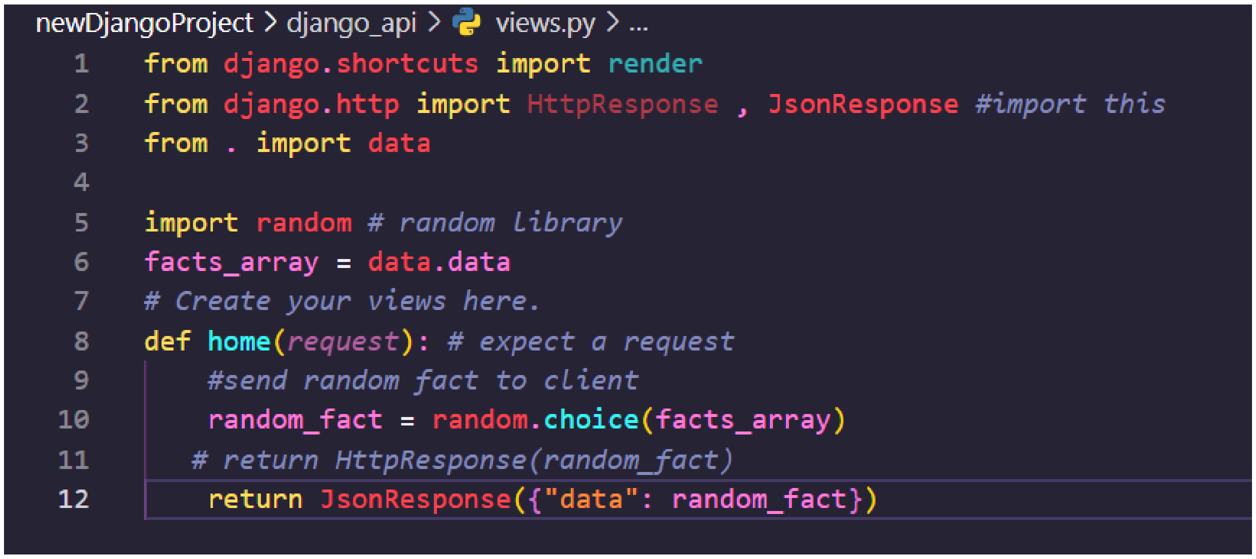
newDjangoProject > django_api > views.py > home

```
from django.shortcuts import render
from django.http import HttpResponse #import this
from . import data
facts_array = data.data
# Create your views here.
def home(request): # expect a request
    return HttpResponse("Hello, world.")
```

Fig 60. Importing data

We will want to use a function that uses this array and gives us a random fact from it. We can use a Python library for this called random, and use random.choice.

Now we return a random fact using JSON with the handy Django utility function.

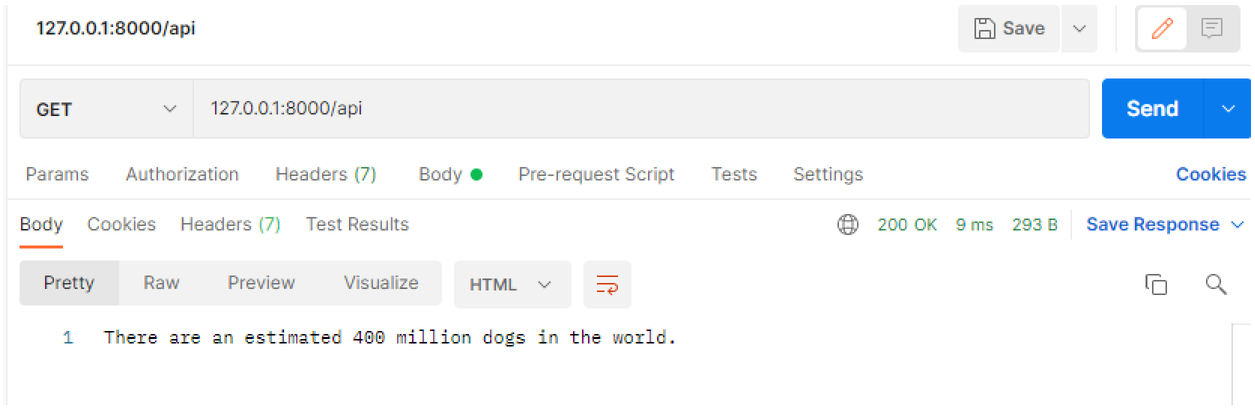


```
newDjangoProject > django_api > views.py > ...
from django.shortcuts import render
from django.http import HttpResponse , JsonResponse #import this
from . import data

import random # random library
facts_array = data.data
# Create your views here.
def home(request): # expect a request
    #send random fact to client
    random_fact = random.choice(facts_array)
  # return HttpResponse(random_fact)
    return JsonResponse({"data": random_fact})
```

Fig 61. Configuring the resp

4.4.5 Testing the response

Use any API tester such as Postman and test that the API returns the data we need.



```
1   There are an estimated 400 million dogs in the world.
```

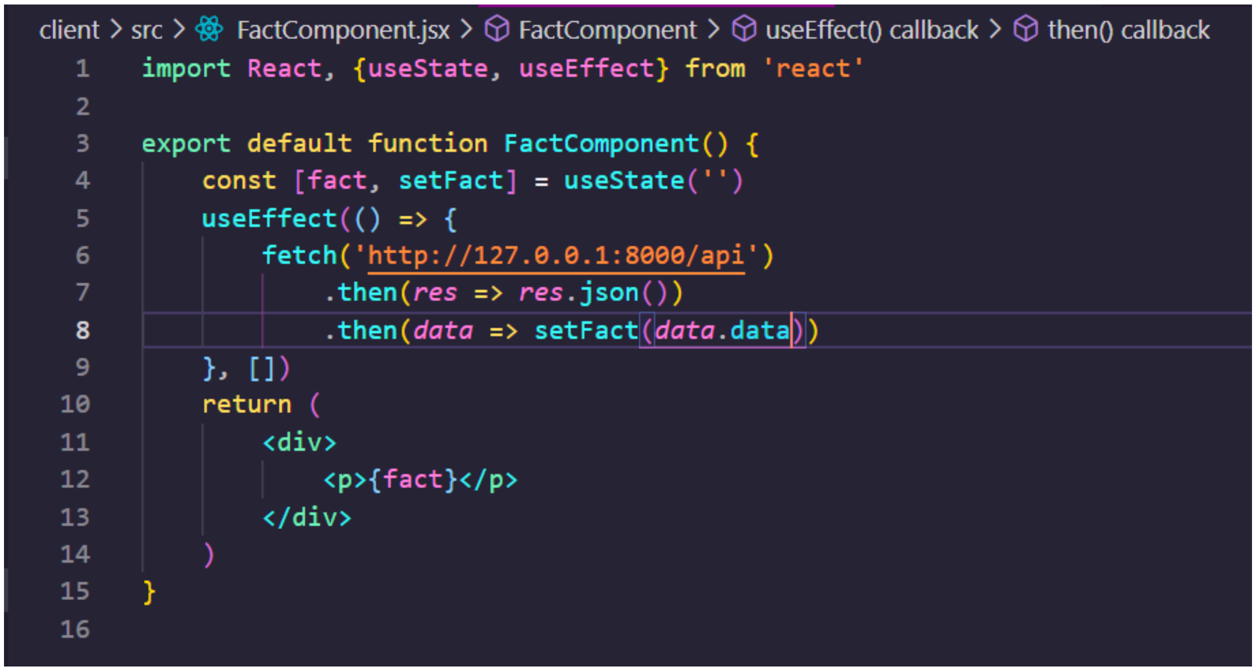
Fig 62. Testing the api

4.4.6 Creating our front end React App

Use the React app created earlier or create a new React app called client. Create a JSX file that will house our functional component that will be used to handle the data collection of our Django API and render it.

4.4.7 Setting up our Hooks/Consuming the api

The way in which React works is we will use the “useEffect” hook that will allow us to use what React calls effects. What we are mimicking from prior versioning is a function that checks when the component has mounted. When the component mounts, React will keep track of the functions



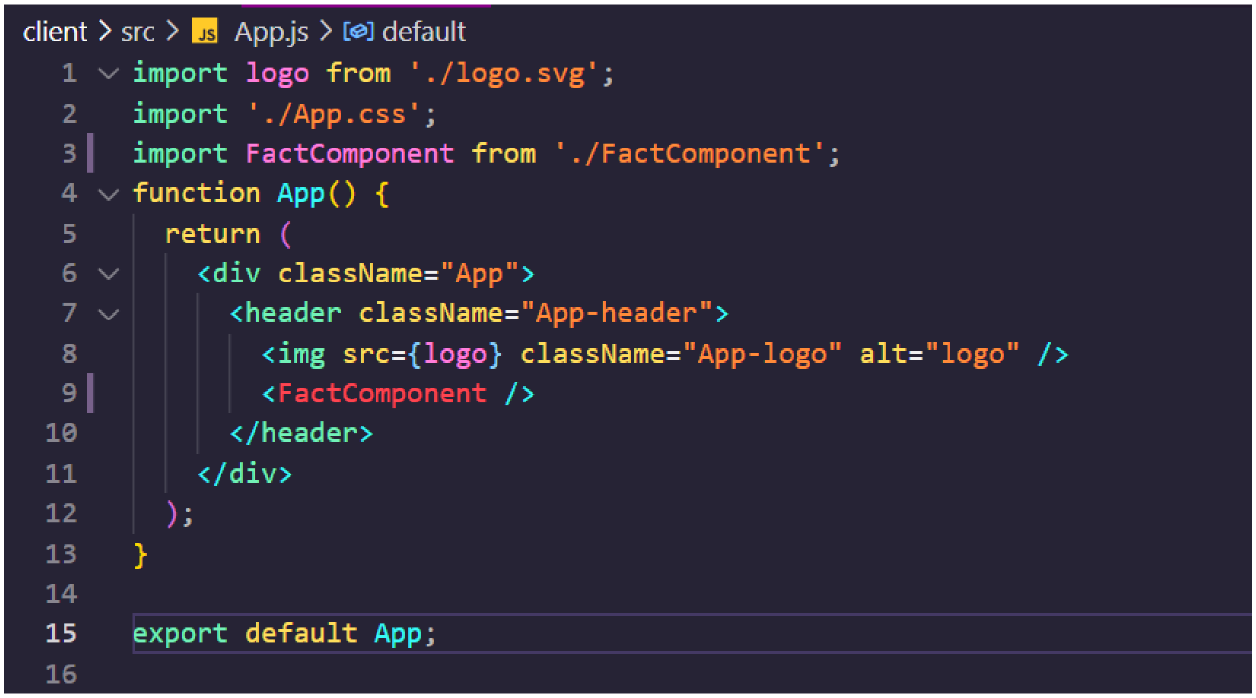
client > src > FactComponent.jsx > FactComponent > useEffect() callback > then() callback

```jsx
import React, {useState, useEffect} from 'react'

export default function FactComponent() {
    const [fact, setFact] = useState('')
    useEffect(() => {
        fetch('http://127.0.0.1:8000/api')
            .then(res => res.json())
            .then(data => setFact(data.data))
    }, [])
    return (
        <div>
            <p>{fact}</p>
        </div>
    )
}

```

Fig 63. Fetching and rendering the data

into the use effect and call them after [108] . Remember though, when React first mounts nothing happens except the mounting of the component. Afterwards, the data gets pulled and then React rerenders since it notices a state change. The new component is rerendered with the new data [93, 108]

4.4.8 Using our component

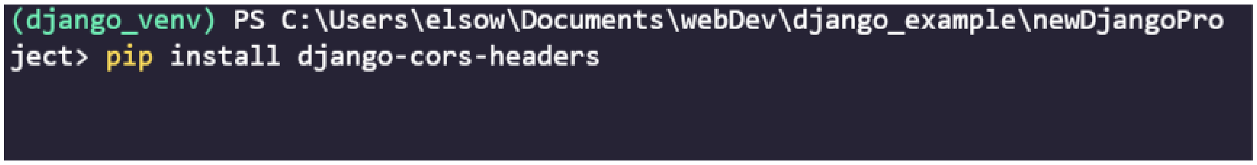


```
client > src > App.js > default
import logo from './logo.svg';
import './App.css';
import FactComponent from './FactComponent';
function App() {
  return (
    <div className="App">
      <header className="App-header">
        <img src={logo} className="App-logo" alt="logo" />
        <FactComponent />
      </header>
    </div>
  );
}

export default App;

```

Fig 64. Importing and using our component

4.4.9 CORS

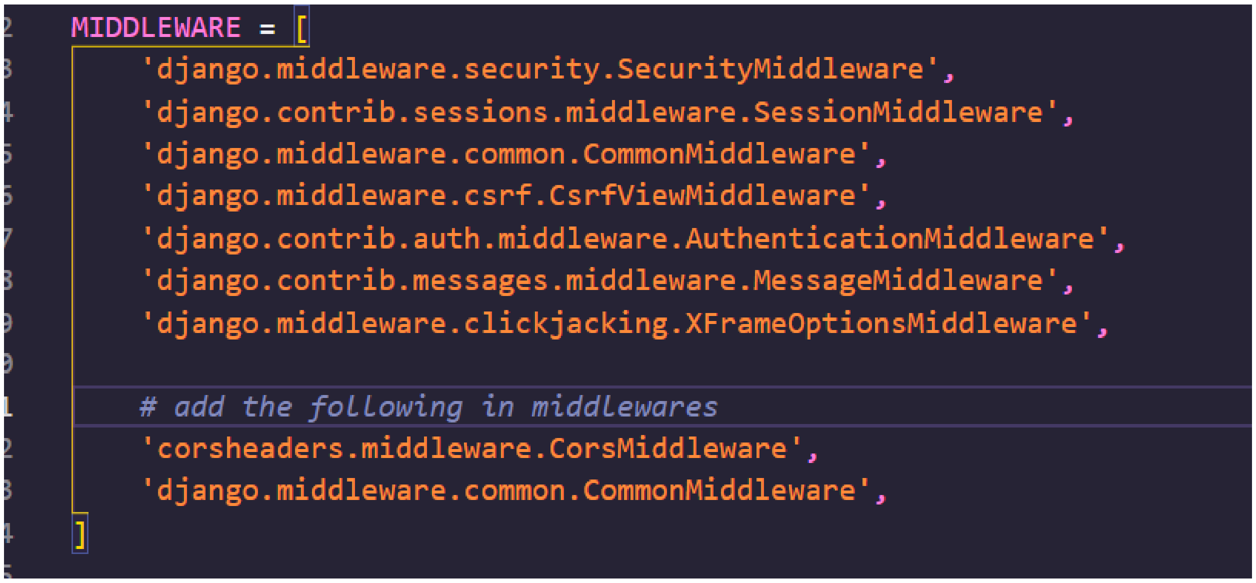
CORS stands for Cross-Origin Resource Sharing and helps to prevent access to the server by specifying the URL and settings of the requesting client [109] . When connecting our Django server to our React front end, we will come across a CORS error. The way to work around this is to either put in the universal acceptor for all URLs which is * or to specify your specific url when deploying.



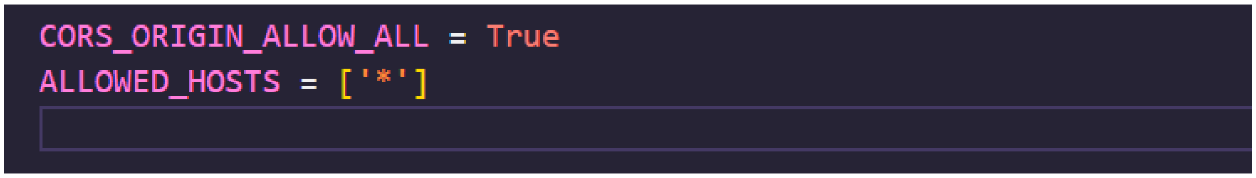
```
(django_venv) PS C:\Users\elsow\Documents\webDev\django_example\newDjangoPro
ject> pip install django-cors-headers
```

Fig 65. CORS download

In settings.py, change the middlewares and add the following lines of code.



```
MIDDLEWARE = [
    'django.middleware.security.SecurityMiddleware',
    'django.contrib.sessions.middleware.SessionMiddleware',
    'django.middleware.common.CommonMiddleware',
    'django.middleware.csrf.CsrfViewMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    'django.contrib.messages.middleware.MessageMiddleware',
    'django.middleware.clickjacking.XFrameOptionsMiddleware',

    # add the following in middlewares
    'corsheaders.middleware.CorsMiddleware',
    'django.middleware.common.CommonMiddleware',
]
```



```
CORS_ORIGIN_ALLOW_ALL = True
ALLOWED_HOSTS = ['*']
```

Fig 66, 67. CORS configuration

4.4.10 Viewing our App

After allowing for cross origin requests from our React app to our Django app, we should be able to successfully retrieve the data we needed.

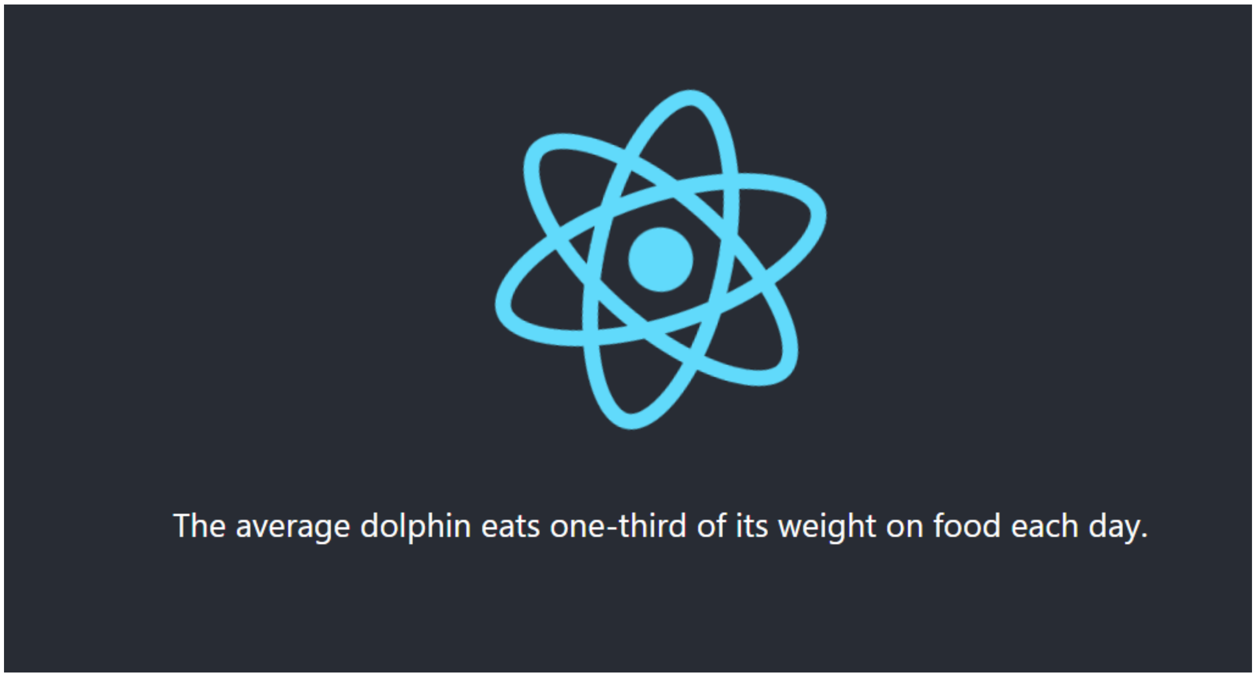

The average dolphin eats one-third of its weight on food each day.

Fig 68. Full App View

5. Discussion

It’s quite interesting to see how JavaScript evolved to where it is at now. The language has conformed to certain standards that continue to improve web computing. Browsers also do a great job of implementing their own version of the JavaScript engine where each does their own job handling certain event queues and memory stacks. It’s also interesting to think that with the evolution of JavaScript, we are able to enjoy the latest features with libraries existing to handle the transpiring of JavaScript. When it comes to Python it’s also packed full of features with such ease of syntax and feature rich libraries that extend so much abstracted complicated logic and

5.1 Discussion of Research Question 1

What is the difference between Python and JavaScript, and what are its features?

Here we saw the main differences in code blocks, syntax, and key libraries that these languages use. We also saw how these languages get run underneath the hood and how outdated libraries get handled. We saw the limitations of the language, specifically dynamic typing, and saw supersets of the languages that aim to solve this issue. We briefly touched on promises and async programming, both associated with advanced concepts. We saw how each language can support these advanced features with supporting libraries and built in features. Although both languages have even more advanced concepts, such as hoisting or decorators, the basics serve as an entry point to understanding key differences and similarities.

5.2 Discussion of Research Question 2

How can we use them in applications, specifically web applications?
Here we saw the ease of Python syntax and with that came the simplification of complex tasks like machine learning or data analysis built into libraries to make it much easier. Python is a general purpose language, much like JavaScript, although it heavily favors more of a scripting/data-analysis programming style. Nonetheless, it can be seen in very popular web frameworks such as light-weight Flask, or full fledged Django. JavaScript, a language of the web, has evolved to multiple frameworks handling the front-end or back-end with libraries supporting ease of creation for servers and runtime environments. JavaScript can also be seen using machine learning libraries and data analysis as well. These topics are pretty advanced and many tutorials and documentation for this can be found online.

5.3 Discussion of Research Question 3

What exactly is the difference between React and Django (popular library and framework for these two languages)?
Here we saw that Django works as a server side rendering language, serving static files and templates as HTML. In React, we learned it’s a UI library, not packed with models or other database methods that a framework would come with such as Django. Nonetheless we saw the

complexity of how a front-end application such as React is handled. With the transpiling of different JavaScript versions or even TypeScript, we see that React abstracts away a lot of this configuration and complexity behind scripts that we run.

5.4 Discussion of Research Question 4

What's the difference between server side rendering and client side rendering?
Here we explored two approaches to serving a website, however a combination of the two does exist where you can accomplish both pre-rendering the static content and then waiting for the dynamic content to load where it “hydrates” the remaining content. This should be explored more in depth, however for this paper we explored Django and React which use both server-side rendering and client side rendering respectively.

5.5 Discussion of Research Question 5

Is it possible to use both, and if so.. How?
We saw that Django can actually serve statically built files from the React build tool. This would result in a combination of the two, however what we saw was the exploration of building an API that submits data to a front-end via a JSON response. This is more standard for the web, however much more should be explored with the RESTful commands such as altering a resource or deleting one. Much better, more detailed documentation does it better justice, as it explores how to connect to a database and see changes made.

6. Contributions

Sherief Elsowiny

7. Lessons learned

Both languages are pretty similar in their appearances when considering the general semantics of what each is doing. The more advanced features such as asynchronous programming and threading are something worth diving into a bit more, as they only give an appearance of concurrency. When it comes to specific applications of each, there are much better resources out there exploring even more advanced concepts within applications. In this paper we focused on specifically Django, React and their features. There are much more advanced concepts to explore that would do it even more justice in exploring the granularity of these languages and frameworks. For example, Redux exists which manages the state of an application when working with React. The exploration of this requires patience as it has an opinionated design, but this should also be worth exploring as even that ecosystem has evolved to make use of more simplistic methods. The exploration of Pythonic data analysis should be worth looking into as well, as the language itself can be known to execute beautiful functions with few lines of code. When looking at how systems work together, there are even more resources to consider. A more practical application would have an automated script set up that handles the running of tests, and for example deployment after changes have been made. In this sense, both these languages can be used to automate these tasks, although tools exist to help. When working with two languages such as Python and JavaScript, there are a myriad of ways of combining the two. Micro applications can be made which communicate to each other or a central entity that handles messages. These Microservices or applications act as mini isolated services that can be made in either JavaScript or Python and can communicate in a number of ways. We saw how Django sends messages to React with JSON responses, but there are also explorations of video chatting or instant messages and other connection tools. When working with data manipulation, both languages can do this easily, and connect to any database and perform queries needed and can abstract away even more complex logic.

8. Conclusion

There is much to explore when looking at the similarities and differences of both these languages. We constrained ourselves to looking at the simple basic features, while focusing on specifically Django and React. When working with both these languages, we see that it can be very complicated with what happens behind the scenes. We were able to see examples of how the JavaScript browser handles the transpiling of JavaScript and the tools out there to convert newer language features. There are a tremendous amount of libraries, tools, and other languages out there worth exploring and diving into. This paper focused on web applications, but the exploration of for example running JavaScript on a mobile device would be worth looking into in the future. Nonetheless, it's interesting to see how both languages have tools built on top of them that make running a web application very easy. When it comes to styling and such, there are resources out there and even templates that make it easier.

9. Code Reference

https://github.com/elsowiny/DjangoReactDemo

10. References

[1] "Stack Overflow Developer Survey 2021." *Stack Overflow*, https://insights.stackoverflow.com/survey/2021

[2] *Django*, https://www.Djangoproject.com/

[3] "React – a JavaScript Library for Building User Interfaces." – *A JavaScript Library for Building User Interfaces*, https://reactjs.org/

[4] “About JavaScript - JavaScript: MDN.” *JavaScript | MDN*, https://developer.mozilla.org/en-US/docs/Web/JavaScript/About_JavaScript

[5] “JavaScript.” *MDN*, https://developer.mozilla.org/en-US/docs/Web/JavaScript

[6] “Dynamic Typing vs. Static Typing.” *Moved*, https://docs.oracle.com/cd/E57471_01/bigData.100/extensions_bdd/src/cext_transform_typing.html#:~:text=First%2C%20dynamically%2Dtyped%20languages%20perform,type%20checking%20at%20compile%20time.&text=If%20a%20script%20written%20in,the%20errors%20have%20been%20fixed

[7] *ECMAScript® 2021 Language Specification*. https://262.ecma-international.org/12.0/

[8] “History and License.” *History and License - Python 3.10.0 Documentation*, https://docs.Python.org/3/license.html

[9] freeCodeCamp.org. “Interpreted vs Compiled Programming Languages: What's the Difference?” *FreeCodeCamp.org*, FreeCodeCamp.org, 28 Apr. 2021, https://www.freecodecamp.org/news/compiled-versus-interpreted-languages/

[10] Copes, Flavio. “Let's Talk about Semicolons in JavaScript.” *FreeCodeCamp.org*, FreeCodeCamp.org, 3 Aug. 2019, https://www.freecodecamp.org/news/lets-talk-about-semicolons-in-JavaScript-f1fe08ab4e53/.

[11] Team, Codecademy. “Your Guide to Semicolons in JavaScript.” *Codecademy News*, Codecademy News, 14 July 2020, https://www.codecademy.com/resources/blog/your-guide-to-semicolons-in-JavaScript/

[12] “Dynamic Typing - MDN Web Docs Glossary: Definitions of Web-Related Terms: MDN.” *MDN Web Docs Glossary: Definitions of Web-Related Terms | MDN*, https://developer.mozilla.org/en-US/docs/Glossary/Dynamic_typing

[13] “Static Typing - MDN Web Docs Glossary: Definitions of Web-Related Terms: MDN.” *MDN Web Docs Glossary: Definitions of Web-Related Terms | MDN*, https://developer.mozilla.org/en-US/docs/Glossary/Static_typing

[14] Schroeder, Kelly. “The Static around Dynamic Languages.” *Medium*, Medium, 30 Oct. 2019, https://medium.com/@kellyrschroeder/the-static-around-dynamical-languages-b52a5d083192

[15] freeCodeCamp.org. “Why Use Static Types in JavaScript? the Advantages and Disadvantages.” *FreeCodeCamp.org*, FreeCodeCamp.org, 8 Dec. 2016, https://www.freecodecamp.org/news/why-use-static-types-in-JavaScript-part-2-part-3-be699ee7be60/

[16] Gros-Dubois, Jonathan. “Statically Typed vs Dynamically Typed Languages.” *Hacker Noon*, 10 Apr. 2017, https://hackernoon.com/statically-typed-vs-dynamically-typed-languages-e4778e1ca55

[17] “Welcome!” *Introduction*, https://web.stanford.edu/class/cs98si/slides/overview.html#:~:text=JavaScript%20is%20an%20interpreted%20language,compiled%20before%20it%20is%20run.&text=Instead%2C%20an%20interpreter%20in%20the,each%20line%2C%20and%20runs%20it

[18] Karani, Dhruvil. “How Does Python Work?” *Medium*, Towards Data Science, 2 Sept. 2020, https://towardsdatascience.com/how-does-Python-work-6f21fd197888

[19] Hiwarale, Uday. “How Does JavaScript and JavaScript Engine Work in the Browser and Node?” *Medium*, JsPoint, 1 Sept. 2020, https://medium.com/jspoint/how-JavaScript-works-in-browser-and-node-ab7d0d09ac2f

[20] “Window - Web Apis: MDN.” *Web APIs | MDN*, https://developer.mozilla.org/en-US/docs/Web/API/Window

[21] “JavaScript Execution Context.” *JavaScript Tutorial*, https://www.JavaScripttutorial.net/JavaScript-execution-context/

[22] Mishra, Rupesh. “Execution Context, Scope Chain and JavaScript Internals.” Medium, Medium, 20 Apr. 2020, https://medium.com/@happymishra66/execution-context-in-JavaScript-319dd72e8e2c

[23] “The Websocket API (WebSockets) - Web Apis: MDN.” *Web APIs | MDN*, https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API.

[24] “WebSockets - a Conceptual Deep Dive.” *Ably Realtime*, https://ably.com/topic/websockets

[25] “Home.” *Internal Working of Python*, https://www.geeksforgeeks.org/internal-working-of-Python/amp/

[26] Bagheri, Reza. “Understanding Python Bytecode.” Medium, Towards Data Science, 5 Mar. 2020, https://towardsdatascience.com/understanding-Python-bytecode-e7edaae8734d

[27] “Python Virtual Machine (PVM).” *GKIndex*, https://www.gkindex.com/Python-tutorial/Python-virtual-machine.jsp

[28] “Dis - Disassembler for Python Bytecode¶.” *Dis - Disassembler for Python Bytecode - Python 3.10.0 Documentation*, https://docs.Python.org/3/library/dis.html
[29] “The Structure of .Pyc Files.” *Ned Batchelder*, 9 Apr. 2008, https://nedbatchelder.com/blog/200804/the_structure_of_pyc_files.html

[30] Singh, Shashwat. “Understanding Python Implementations.” *Medium*, The Startup, 21 July 2020, https://medium.com/swlh/understanding-Python-implementations-53015a3698e5

[31] Team, The PyPy. “Features.” *PyPy*, 28 Dec. 2019, https://www.pypy.org/features.html

[32] Real Python. “PyPy: Faster Python with Minimal Effort.” *Real Python*, Real Python, 26 June 2021, https://realPython.com/pypy-faster-Python/

[33] “How JIT Compilers Are Implemented and Fast: Pypy, Luajit, Graal and More.” *Kipply's Blog*, https://carolchen.me/blog/technical/jits-impls/

[34] “Home.” *Ecma International*, 20 Aug. 2020, https://www.ecma-international.org/

[35] freeCodeCamp.org. “What's the Difference between JavaScript and ECMAScript?” *FreeCodeCamp.org*, FreeCodeCamp.org, 25 Aug. 2017, https://www.freecodecamp.org/news/whats-the-difference-between-JavaScript-and-ecmascript-cba48c73a2b5/

[36] “ECMAScript 2018.” *JavaScript ECMAScript 2018*, https://www.w3schools.com/js/js_2018.asp

[37] “ECMAScript 2017.” *JavaScript ECMAScript 2017*, https://www.w3schools.com/js/js_2017.asp

[38] “JavaScript ES6.” *JavaScript ES6*, https://www.w3schools.com/js/js_es6.asp

[39] *JavaScript ES5*, https://www.w3schools.com/js/js_es5.asp

[40] “JavaScript with Syntax for Types.” *TypeScript*, https://www.typescriptlang.org/

[41] Navone, Estefania Cassingena. "Python vs JavaScript – What Are the Key Differences between the Two Popular Programming Languages?" *FreeCodeCamp.org*, FreeCodeCamp.org, 28 Jan. 2021, https://www.freecodecamp.org/news/Python-vs-JavaScript-what-are-the-key-differences-between-the-two-popular-programming-languages/amp/

[42] "Computer Programming for Everybody." *Python.org*, https://www.Python.org/doc/essays/cp4e/

[43] "PEP 8 -- Style Guide for Python Code." *Python.org*, https://www.Python.org/dev/peps/pep-0008/#indentation

[44] "PEP 8 -- Style Guide for Python Code." *Python.org*, https://www.Python.org/dev/peps/pep-0008/#function-and-variable-names

[45] *JavaScript Style Guide*, https://www.w3schools.com/js/js_conventions.asp.

[46] "Differences Between Var and Let." *StackPath*, https://www.JavaScripttutorial.net/es6/difference-between-var-and-let/

[47] "Built-in Types¶." *Built-in Types - Python 3.10.0 Documentation*, https://docs.Python.org/3/library/stdtypes.html

[48] *Python None Keyword*, https://www.w3schools.com/Python/ref_keyword_none.asp

[49] "What Is Python? Executive Summary." *Python.org*, https://www.Python.org/doc/essays/blurb/

[50] "C-Extensions for Python." *Cython*, https://cython.org/

[51] "Software Framework vs Library." *GeeksforGeeks*, 7 Sept. 2020, https://www.geeksforgeeks.org/software-framework-vs-library/

[52] *Python Dictionaries*, https://www.w3schools.com/Python/Python_dictionaries.asp

[53] Adesoga, Deji. “JavaScript Object VS JSON: Demystified.” *DEV Community*, DEV Community, 28 May 2020, https://dev.to/desoga/JavaScript-object-vs-json-demystified-494j

[54] *JavaScript Objects*, https://www.w3schools.com/js/js_objects.asp

[55] Germain, H. James de St. “Functions.” *Programming - Functions*, https://www.cs.utah.edu/~germain/PPS/Topics/functions.html

[56] *The V8 JavaScript Engine*, https://nodejs.dev/learn/the-v8-JavaScript-engine.

[57] “Top 10 Python Libraries You Must Know in 2021.” *Edureka*, 6 Aug. 2021, https://www.edureka.co/blog/Python-libraries/

[58] “Welcome to Flask¶.” *Welcome to Flask - Flask Documentation (2.0.x)*, https://flask.palletsprojects.com/en/2.0.x/

[59] “Top 10 JavaScript Libraries for Machine Learning and Data Science.” *GeeksforGeeks*, 14 Dec. 2020, https://www.geeksforgeeks.org/top-10-JavaScript-libraries-for-machine-learning-and-data-science

[60] Javinpaul. “10 JavaScript Frameworks and Libraries to Learn in 2020-Best of Lot.” *Medium*, Javarevisited, 11 Dec. 2020, https://medium.com/javarevisited/10-JavaScript-frameworks-and-libraries-to-learn-in-2020-best-of-lot-5f61f86c60b4

[61] Bruijn, Sander de. “A Brief Explanation of the JavaScript Engine and Runtime.” *Medium*, Medium, 22 Feb. 2020, https://medium.com/@sanderdebr/a-brief-explanation-of-the-JavaScript-engine-and-runtime-a0c27cb1a397

[62] “JavaScript Data Types and Data Structures - JavaScript: MDN.” *JavaScript | MDN*, https://developer.mozilla.org/en-US/docs/Web/JavaScript/Data_structures

[63] “Python Data Structures with Primitive & Non-Primitive Examples.” *DataCamp Community*, https://www.datacamp.com/community/tutorials/data-structures-Python.

[64] “5. Data Structures¶.” *5. Data Structures - Python 3.10.0 Documentation*, https://docs.Python.org/3/tutorial/datastructures.html#tuples-and-sequences.

[65] “Introducing Json.” *JSON*, https://www.json.org/json-en.html

[66] *JavaScript Operators*, https://www.w3schools.com/js/js_operators.asp

[67] *Python Operators*, https://www.w3schools.com/Python/Python_operators.asp

[68] “8. Compound Statements¶.” *8. Compound Statements - Python 3.10.0 Documentation*, https://docs.Python.org/3/reference/compound_stmts.html

[69] “4. More Control Flow Tools¶.” *4. More Control Flow Tools - Python 3.10.0 Documentation*, https://docs.Python.org/3/tutorial/controlflow.html

[70] “What Is a Class?” *What Is a Class? (The Java™ Tutorials > Learning the Java Language > Object-Oriented Programming Concepts)*, https://docs.oracle.com/javase/tutorial/java/concepts/class.html

[71] “What Is an Object?” *What Is an Object? (The Java™ Tutorials > Learning the Java Language > Object-Oriented Programming Concepts)*, https://docs.oracle.com/javase/tutorial/java/concepts/object.html

[72] Real Python. “An Intro to Threading in Python.” *Real Python*, Real Python, 19 June 2021, https://realPython.com/intro-to-Python-threading/

[73] “Concurrency Model and the Event Loop - Javascript: MDN.” *JavaScript | MDN*, https://developer.mozilla.org/en-US/docs/Web/JavaScript/EventLoop

[74] “Events, Concurrency and JavaScript • Dan Martensen.” *Dan Martensen on Svbtle*, https://danmartensen.svbtle.com/events-concurrency-and-javascript

[75] “Asyncio - Asynchronous I/O¶.” *Asyncio - Asynchronous I/O - Python 3.10.0 Documentation*, https://docs.Python.org/3/library/asyncio.html

[76] Jun 21, 2017 | By Michael Flaxman. “Python 3's Killer Feature: Asyncio.” *PAXOS*, https://eng.paxos.com/Python-3s-killer-feature-asyncio

[77] “Express/Node Introduction - Learn Web Development: MDN.” *Learn Web Development | MDN*, https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express_Nodejs/Introduction.

[78] “React Native · Learn Once, Write Anywhere.” *React Native*, https://reactnative.dev/.

[79] “Documentation.” *Django*, https://docs.Djangoproject.com/en/3.2/topics/db/models/.

[80] “Pip.” *PyPI*, https://pypi.org/project/pip/

[81] “12. Virtual Environments and Packages¶.” *12. Virtual Environments and Packages - Python 3.10.0 Documentation*, https://docs.python.org/3/tutorial/venv.html

[82] “Documentation.” *Django*, https://docs.Djangoproject.com/en/3.2/ref/contrib/admin/

[83] “Documentation.” *Django*, https://docs.djangoproject.com/en/3.2/intro/tutorial01/

[84] “Documentation.” *Django*, https://docs.djangoproject.com/en/3.2/topics/migrations/

[85] “Django Tutorial Part 2: Creating a Skeleton Website - Learn Web Development: MDN.” *Learn Web Development | MDN*, https://developer.mozilla.org/en-US/docs/Learn/Server-side/Django/skeleton_website

[86] “Documentation.” *Django*, https://docs.djangoproject.com/en/3.2/topics/http/views/

[87] “Documentation.” *Django*, https://docs.djangoproject.com/en/3.2/topics/templates/

[88] “Django Crud (Create, Retrieve, Update, Delete) Function Based Views.” *GeeksforGeeks*, 27 Aug. 2021, https://www.geeksforgeeks.org/django-crud-create-retrieve-update-delete-function-based-views/

[89] Christie, Tom. “Tutorial 2: Requests and Responses.” *2 - Requests and Responses - Django REST Framework*, https://www.django-rest-framework.org/tutorial/2-requests-and-responses/

[90] “ReactJS Features - Javatpoint.” *Www.javatpoint.com*, https://www.javatpoint.com/react-features

[91] Mateusz Piguła Frontend Developer Adrian Senecki Content Creator Mateusz, et al. “How Does the React Component Lifecycle Work? Lifecycle Methods and Hooks.” *The Software House*, 17 June 2020, https://tsh.io/blog/react-component-lifecycle-methods-vs-hooks/

[92] “Hooks at a Glance.” *React*, https://reactjs.org/docs/hooks-overview.html

[93] “State and Lifecycle.” *React*, https://reactjs.org/docs/state-and-lifecycle.html

[94] “Introducing Hooks.” *React*, https://reactjs.org/docs/hooks-intro.html

[95] “React.component.” *React*, https://reactjs.org/docs/react-component.html

[96] FacebookDevelopers, director. *YouTube*, YouTube, 28 Jan. 2015, https://www.youtube.com/watch?v=KVZ-P-ZI6W4&list=PLb0IAmt7-GS1cbw4qonlQztYV1TAW0sCr&index=2. Accessed 11 Nov. 2021

[97] RisingStack Engineering. “The History of React.js on a Timeline.” *RisingStack Engineering*, 11 Oct. 2021, https://blog.risingstack.com/the-history-of-react-js-on-a-timeline/

[98] “Virtual Dom and Internals.” *React*, https://reactjs.org/docs/faq-internals.html

[99] Gasimzada, Gasim. “What Are NPM, Yarn, Babel, and Webpack; and How to Properly Use Them?” *Medium*, Frontend Weekly, 28 Dec. 2019, https://medium.com/front-end-weekly/what-are-npm-yarn-babel-and-webpack-and-how-to-properly-use-them-d835a758f987

[100] “How to Update Parent State in Reactjs ?” *GeeksforGeeks*, 22 Dec. 2020, https://www.geeksforgeeks.org/how-to-update-parent-state-in-reactjs/

[101] Ravan, et al. “What Is Rest.” *REST API Tutorial*, 19 Oct. 2021, https://restfulapi.net/.

[102] “About NPM.” *Npm Docs*, https://docs.npmjs.com/about-npm

[103] “Components and Props.” *React*, https://reactjs.org/docs/components-and-props.html

[104] “How to Pass Props to a Component Rendered by React Router.” *Ui.dev*, https://ui.dev/react-router-pass-props-to-components/

[105] “REACT Router: Declarative Routing for React.” *ReactRouterWebsite*, https://v5.reactrouter.com/web/example/url-params

[106] Hiwarale, Uday. “A Beginner's Guide to React Server-Side Rendering (SSR).” *Medium*, JsPoint, 7 Dec. 2020, https://medium.com/jspoint/a-beginners-guide-to-react-server-side-rendering-ssr-bf3853841d55

[107] Author: Carey WodehouseCarey Wodehouse is an IT/development content writer at Upwork who is dedicated to making the complex world of web development a little easier to navigate. Residing in Richmond, et al. “Can Server-Side Rendering Boost Customer Engagement?” *Business 2 Community*, https://www.business2community.com/brandviews/upwork/can-server-side-rendering-boost-customer-engagement-01919656

[108] “Using the Effect Hook.” *React*, https://reactjs.org/docs/hooks-effect.html

[109] “Cross-Origin Resource Sharing (CORS) - Http: MDN.” *HTTP | MDN*, https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS

[110] “MVC Framework.” *Technoarch Softwares*, https://www.technoarchsoftwares.com/blog/mvc-framework/

[111] Joshi, Sushmita. “The Django Framework!” *Medium*, Medium, 25 May 2020, https://medium.com/@sushmitajoshi22/the-django-framework-cccfc439d865

[112] “Venv - Creation of Virtual Environments¶.” *Venv - Creation of Virtual Environments - Python 3.10.0 Documentation*, https://docs.python.org/3/library/venv.html

[113] “@Babel/Plugin-Transform-Async-to-Generator · BABEL.” *Babel*, https://babeljs.io/docs/en/babel-plugin-transform-async-to-generator

[114] “Using Promises - Javascript: MDN.” *JavaScript | MDN*, https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Using_promises