UNIVERSITY OF CAPE TOWN
IYUNIVESITHI YASEKAPA • UNIVERSITEIT VAN KAAPSTAD

Fintech & Cryptocurrencies Tutorials

MPhil FinTech 2024 Coding Tutorials

Francisco Rutayebesibwa
rtyfra001@myuct.ac.za
April 2024

Slide credits:
Julian Kanjare, MPhil Fintech 2021 Coding Tutorials

CS50's Web Programming with Python and JavaScript

Freecodecamp

How does the Internet work?

Defining key terminologies used in Web Development

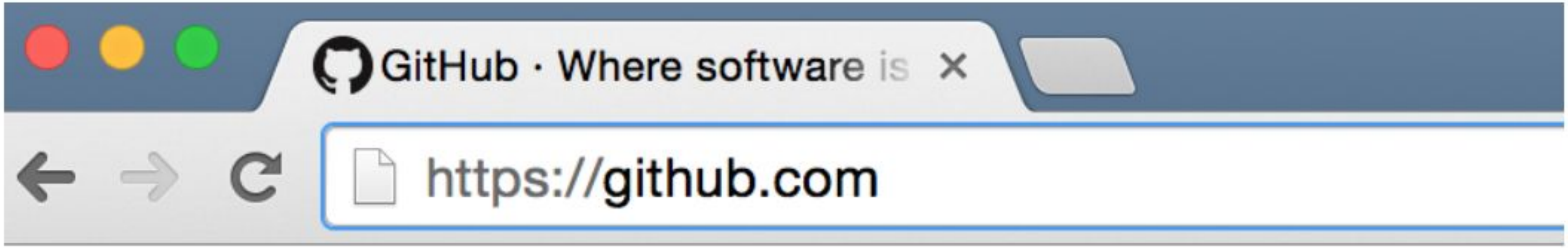
- **Client:** An application, such as Chrome or Firefox, that runs on a computer and is connected to the internet. Its primary role is to take user interactions and translate them into requests to another computer called a web server.
- **Server:** A machine that is connected to the internet and also has an IP address. A server waits for requests from other machines (e.g. a client) and responds to them.
- **HTTP**: Hypertext Transfer Protocol. The protocol that web browsers and web servers use to communicate with each other over the Internet.
- **URL:** Uniform Resource Locators. URLs identify a particular web resource. A simple example is https://github.com/iscoRuta98/
- **TCP/IP:** Transmission Control Protocol/Internet Protocol.
  - Protocol = simply a standard set of rules for doing something
  - TCP/IP is used as a standard for transmitting data over networks.

Defining key terminologies used in Web Development

- **IP address:** Internet Protocol Address. A numerical identifier for a device (computer, server, printer, router, etc.) on a TCP/IP network.
- **ISP:** Internet Service Provider. ISP is the intermediary between the client and servers.
- **Domain Name:** Used to identify one or more IP addresses. Users use the domain name (e.g. www.github.com) to get to a website on the internet. When you type the domain name into your browser, the DNS uses it to look up the corresponding IP address for that given website.
- **DNS:** Domain Name System. A distributed database which keeps track of computer's domain names and their corresponding IP addresses on the Internet.

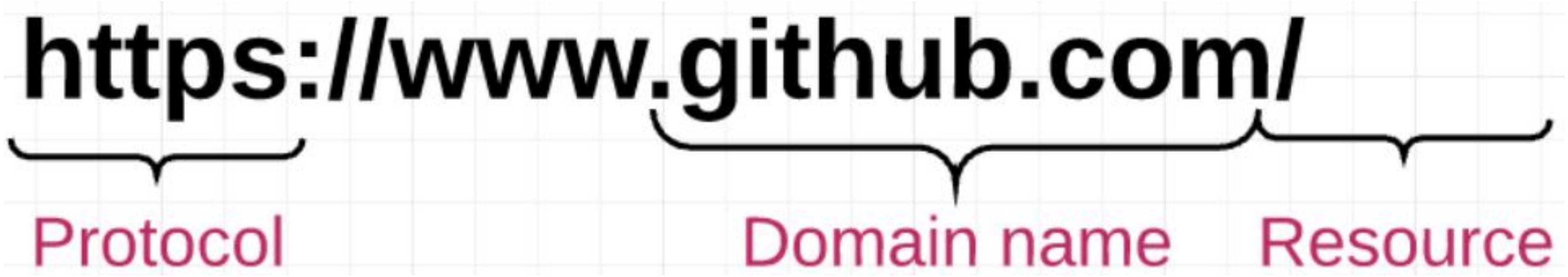
The Journey

1. Type URL into the browser



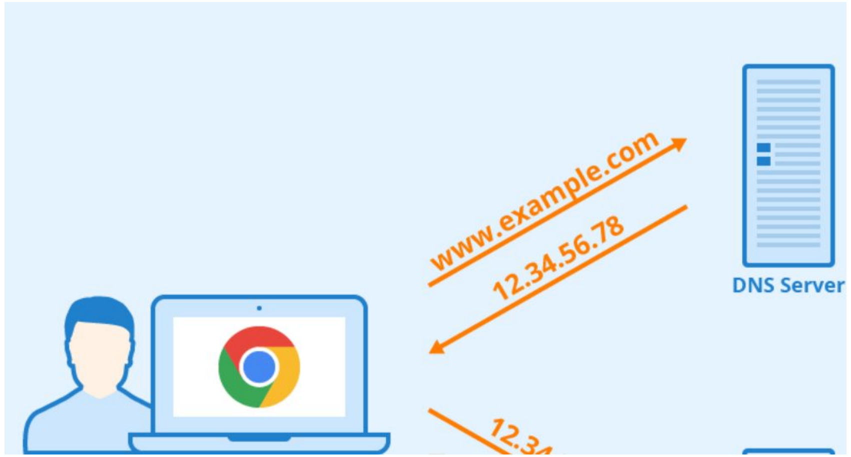
The Journey

2. The browser parses the information contained in the URL



The Journey

3. The browser communicates with your ISP to do a DNS lookup of the IP address for the web server that hosts www.github.com.

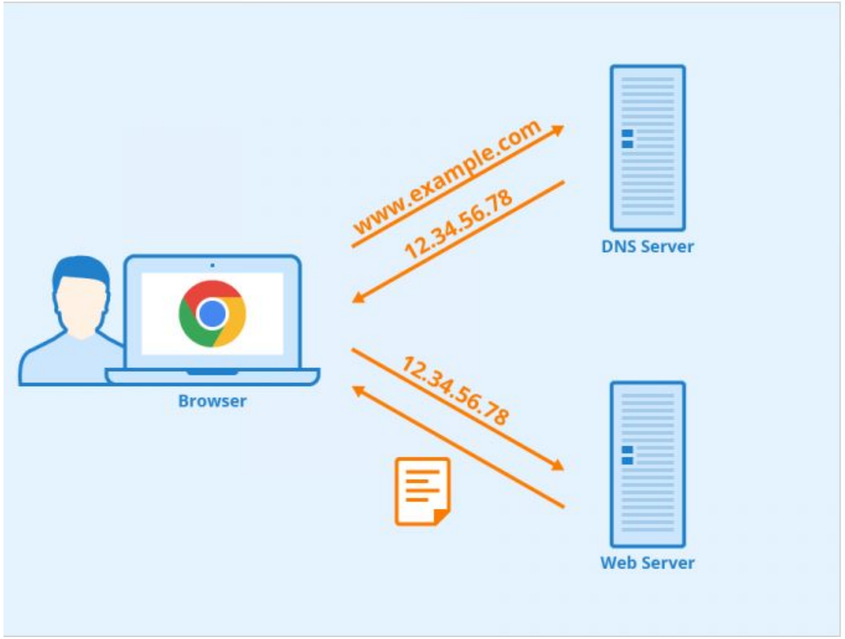


The Journey

3. The browser communicates with your ISP to do a DNS lookup of the IP address for the web server that hosts www.github.com.

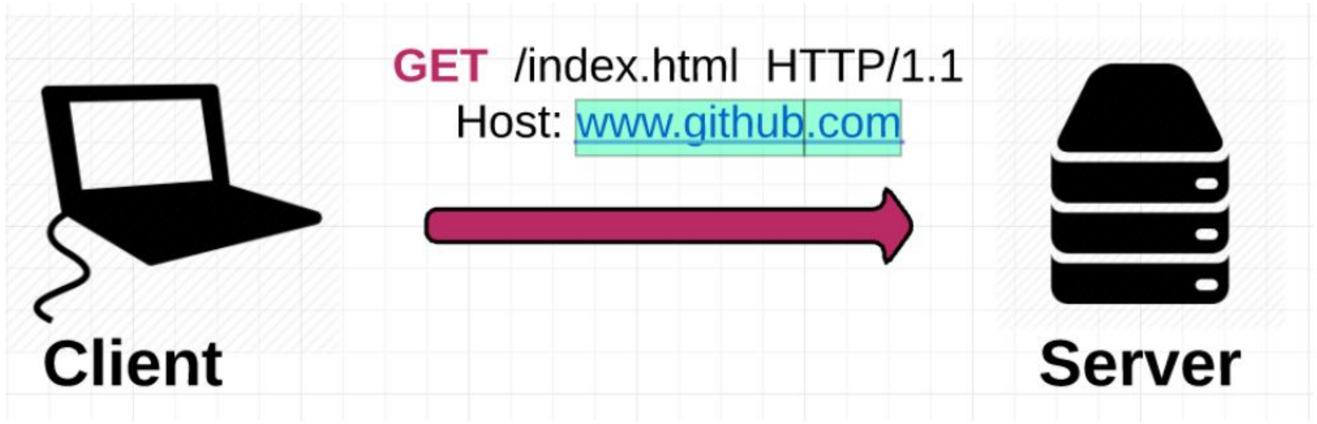
4. Once the ISP receives the IP address of the destination server, it sends it to your web browser.

5. Your browser takes the IP address and the given port number from the URL and opens a TCP socket connection.



The Journey

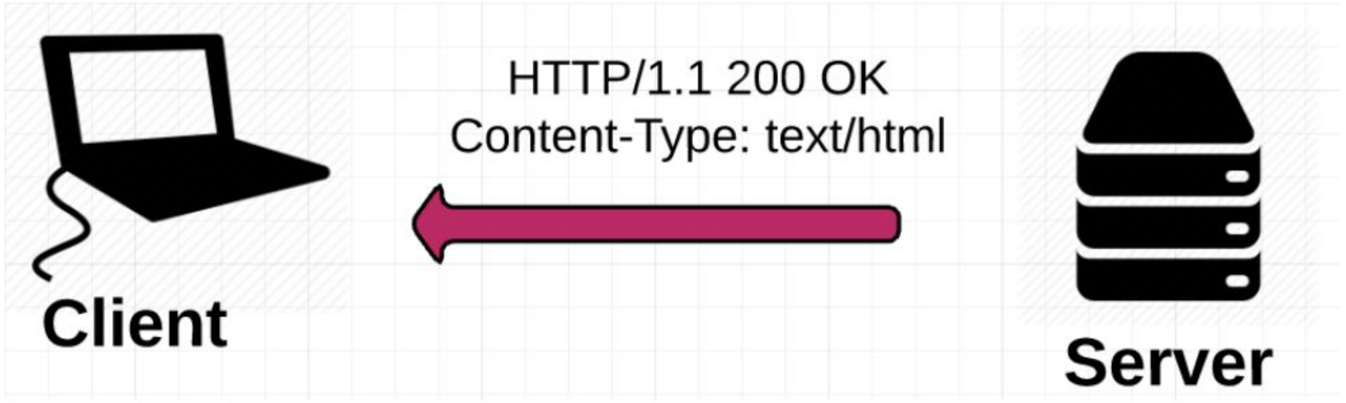
6. Your web browser sends an HTTP request to the web server for the main HTML web page of www.github.com



GET request from Client

The Journey

7. The web server receives the request and looks for that HTML page.

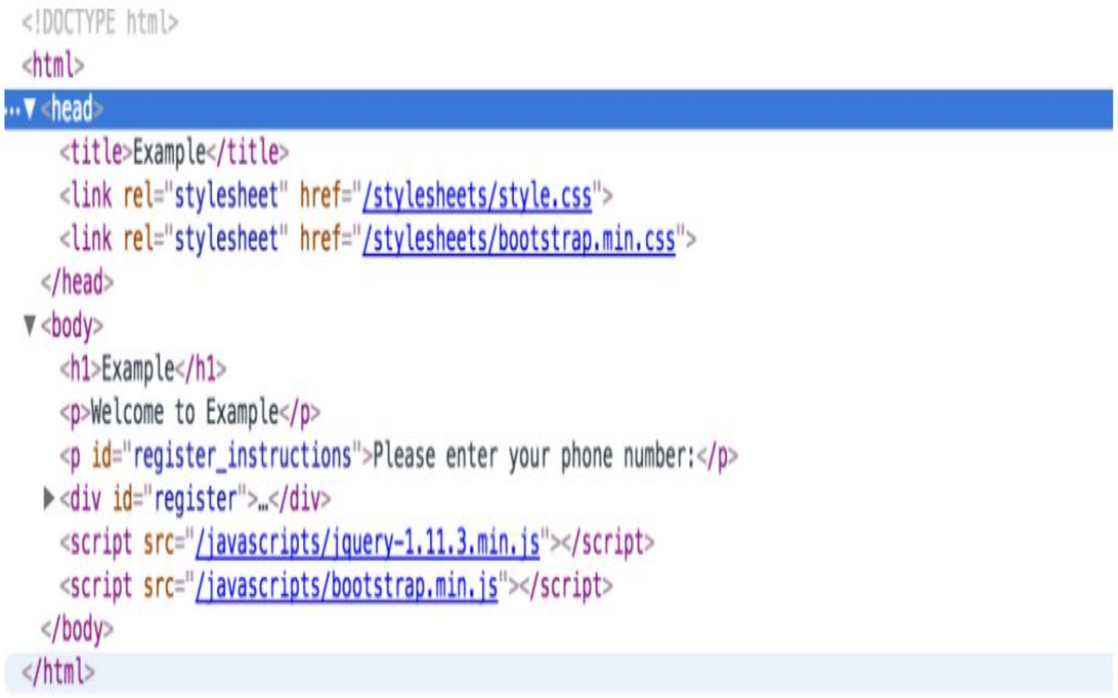


Server response

The Journey

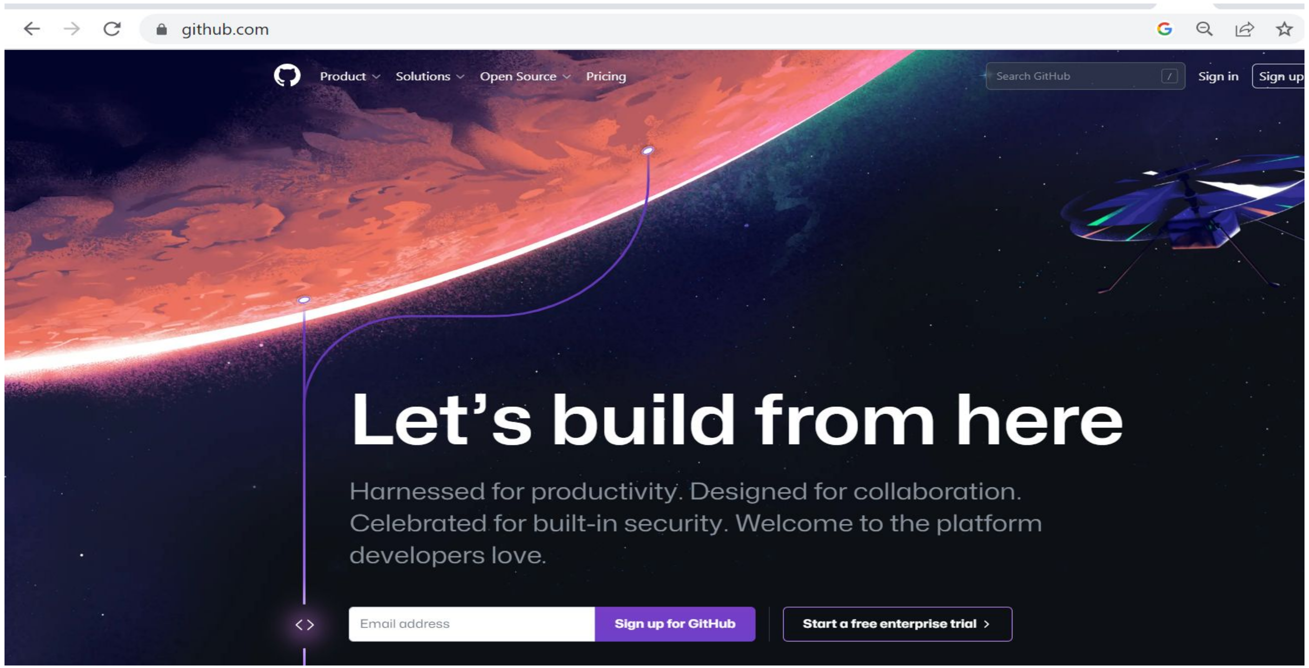
8. The web browser takes the HTML page, parses through it doing a full scan looking for other assets that are listed, such as images, CSS files, JavaScript files, etc.

9. For each asset listed in the HTML page, the browser repeats the entire process above, making additional HTTP requests to the server for each resource.



```
<!DOCTYPE html>
<html>
…▼ <head>
    <title>Example</title>
    <link rel="stylesheet" href="/stylesheets/style.css">
    <link rel="stylesheet" href="/stylesheets/bootstrap.min.css">
  </head>
 ▼ <body>
    <h1>Example</h1>
    <p>Welcome to Example</p>
    <p id="register_instructions">Please enter your phone number:</p>
   ▶<div id="register">…</div>
    <script src="/javascripts/jquery-1.11.3.min.js"></script>
    <script src="/javascripts/bootstrap.min.js"></script>
  </body>
</html>
```

Final result...


github.com

Product Solutions Open Source Pricing

Search GitHub

Sign in Sign up

Let’s build from here

Harnessed for productivity. Designed for collaboration. Celebrated for built-in security. Welcome to the platform developers love.

Email address

Sign up for GitHub

Start a free enterprise trial >

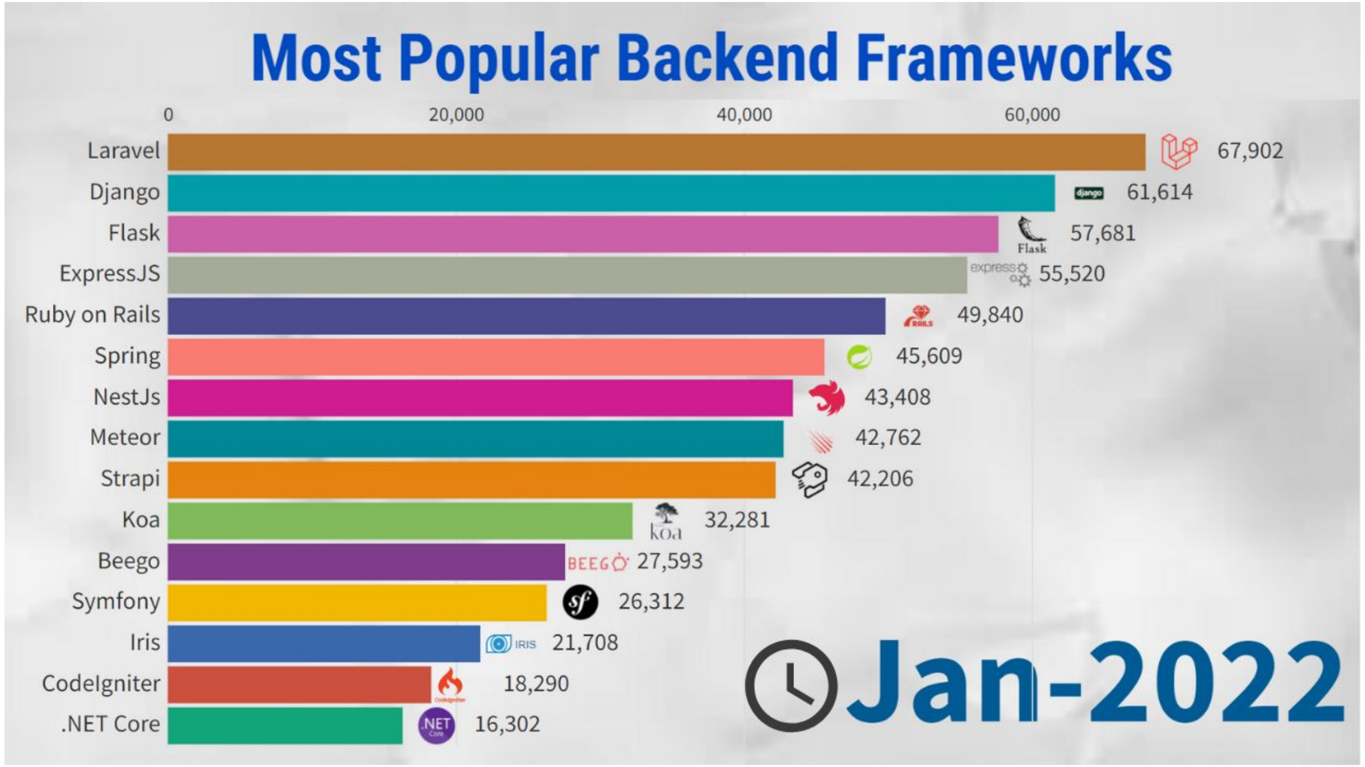
Further readings...

- How the Web Works: A Primer for Newcomers to Web Development (or anyone, really)
- How Does the Internet Work?
- How does the Internet work? | Cloudflare.

Backend/Server-side Development

Backend Development

- The major aim of backend development is to essentially respond to user's request.
- Backend functionalities include: ensuring a website performs correctly, focusing on databases, back-end logic, application programming interface (APIs), architecture, and servers.
- Backend development can be written in different programming languages such as: JavaScript, Python, C#, Go, Rust, etc.



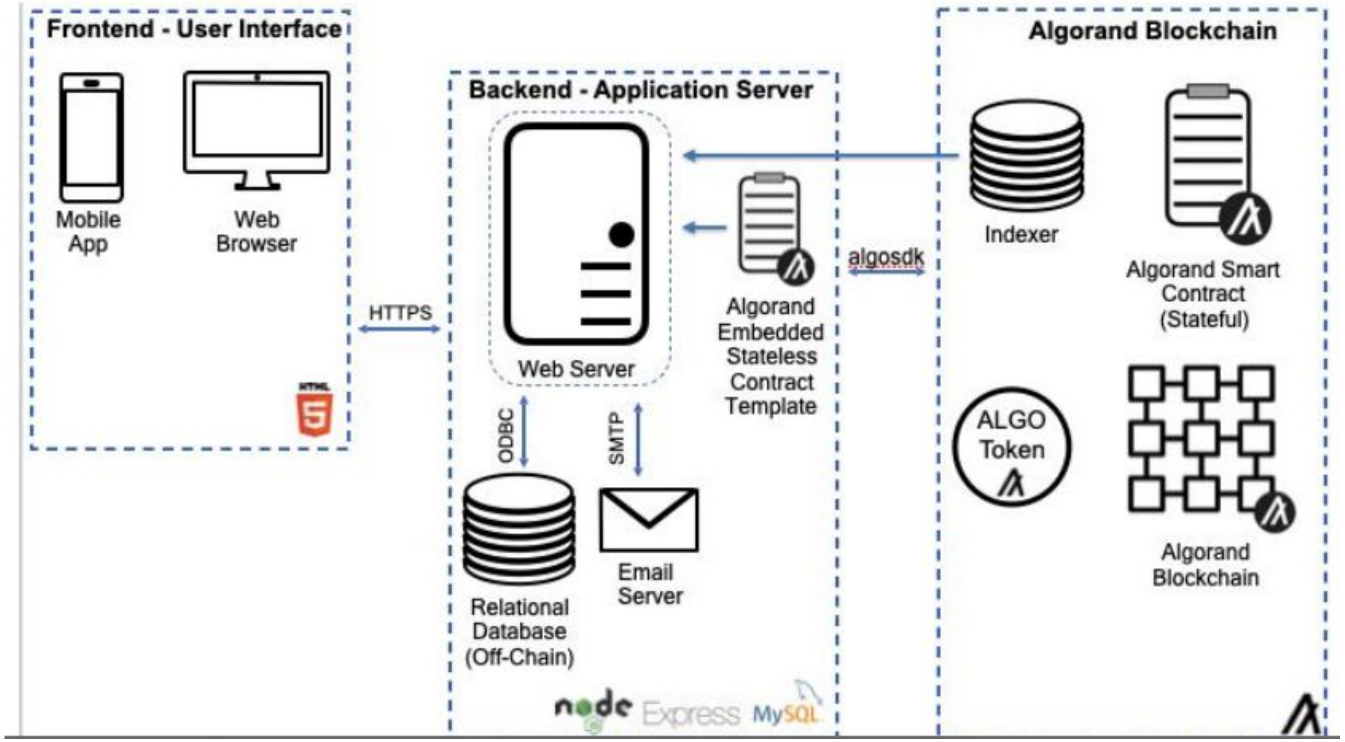
Most Popular Backend Frameworks
0
20,000
40,000
60,000
Laravel 67,902
Django 61,614
Flask 57,681
ExpressJS 55,520
Ruby on Rails 49,840
Spring 45,609
NestJs 43,408
Meteor 42,762
Strapi 42,206
Koa 32,281
Beego 27,593
Symfony 26,312
Iris 21,708
CodeIgniter 18,290
.NET Core 16,302
Jan-2022

Hypertext Transfer Protocol HTTP

What is HTTP?

- **HTTP** definite set of rules, for accessing resources on the web.
  - Resources can be anything from: HTML files, data from database, videos, photos, etc.
- **Client-Server Architecture:** This architecture describes how all web applications work and defines the rules for how they communicate.
  - *Client Application*: The application the user is actually interacting with, that's displaying the content.
  - *Server Application*: The application that *receives* the content, or resource, to the client application. A server application is a program that is running somewhere, listening, and waiting for a request.
- **Client-Server Communication**:
  - Communications between the client and server are almost always initiated by clients in the form of **requests**.
  - These requests are fulfilled by the server application which sends back a **response** containing the resource you requested, among other things

Example Architecture


Frontend - User Interface
Mobile App
Web Browser
HTTPS
Backend - Application Server
Web Server
Algorand Embedded Stateless Contract Template
ODBC
SMTP
Relational Database (Off-Chain)
Email Server
node
Express
MySQL
algosdk
Algorand Blockchain
Indexer
Algorand Smart Contract (Stateful)
ALGO Token
Algorand Blockchain

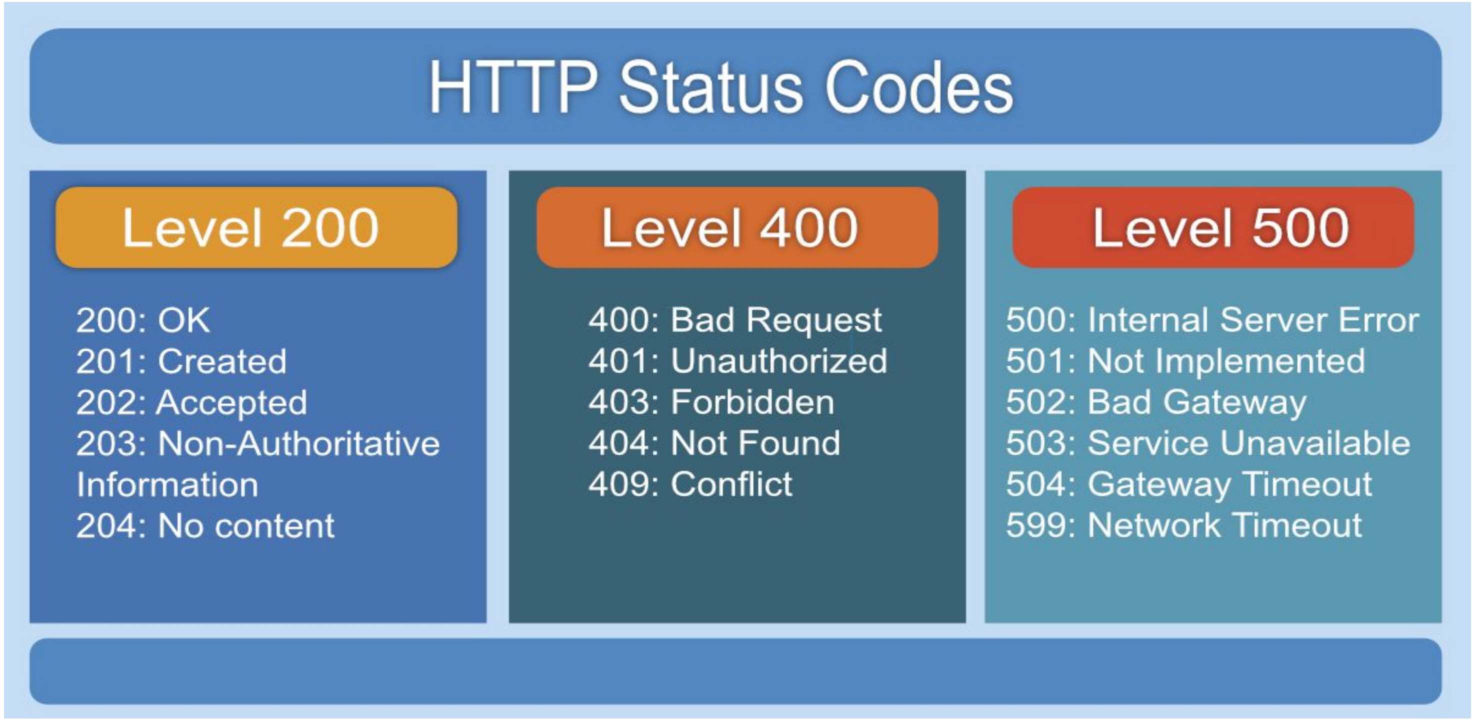
Discussion: Why do we need client-server architecture?

HTTP Requests & Response

- **Anatomy of HTTP Request:** A request must contain the following:
  - An HTTP Method (e.g. GET)
  - A host URL (e.g. https://api.spotify.com/
  - An endpoint path (e.g. v1/artists/{id}/related-artists
  - Additionally, HTTP Request can also optionally have:
    - Body, Headers, Query strings, and HTTP version.
- **Anatomy of HTTP Response:** A request must contain the following:
  - Protocol version (e.g. HTTP/1.1)
  - Status code (e.g. 200)
  - Status text
  - Headers
  - Additionally, HTTP response can also optionally have:
    - Body.

HTTP Requests Methods

- **POST** Request**:** create a new resource/content.
  - A POST request requires a body in which you define the data of the entity to be created.
  - A successful POST request can either be 200 (OK) or 201 (CREATED).
- **GET** Request: Use it to read or retrieve a resource.
  - A successful GET returns a response containing the information requested by the client.
- **PUT** Request: We use this request to modify a resource.
  - PUT updates the entire resource with data that is passed in the body payload.
- **PATCH** Request: This request modifies a part of a resource/data.
  - With PATCH, the client only needs to pass in the data that it wants to update.
- **DELETE** Request: This request deletes a resource.



HTTP Status Codes

Level 200	Level 400	Level 500
200: OK	400: Bad Request	500: Internal Server Error
201: Created	401: Unauthorized	501: Not Implemented
202: Accepted	403: Forbidden	502: Bad Gateway
203: Non-Authoritative Information	404: Not Found	503: Service Unavailable
204: No content	409: Conflict	504: Gateway Timeout
		599: Network Timeout

Source: 8 Most Common HTTP Error Codes

HTTP Requests & Network Tab

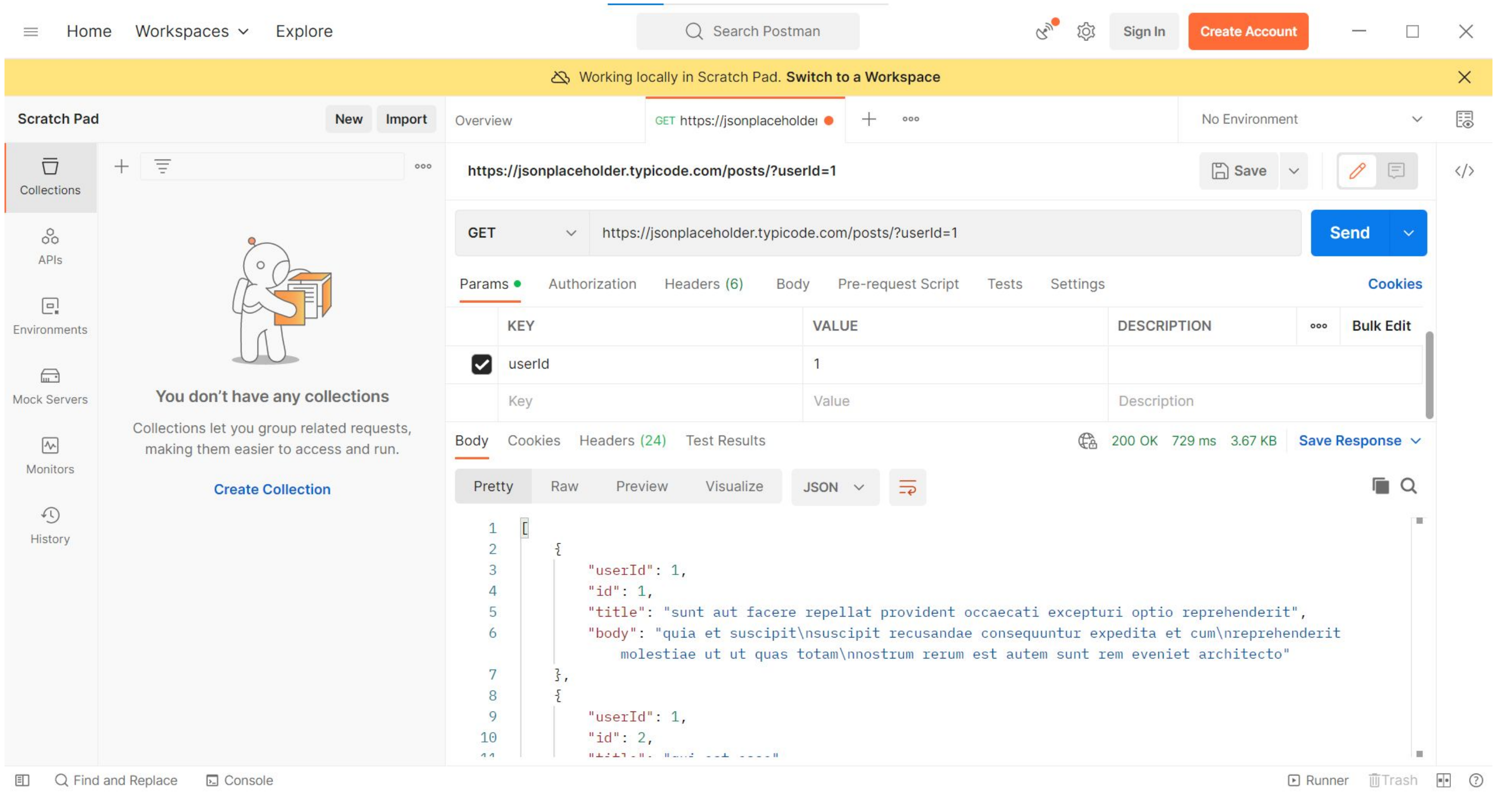
- When you type a website, such as www.google.com, browsers (like Google Chrome, Mozilla Firefox & Brave) have a network tab that keeps track of HTTP requests being sent to a server.
- For instance, when you go to Google, you are sending a **GET** request.
- By opening the network tab, you can see what is in the request headers, and the resources fetched by your request.
- When you click on a resource, you can check the **status code** and **status text** of the response.

Application Programming Interface (API)

What are APIs?

- APIs enables developers to access specific features and data of a software application or service.
    - Formal definition: *An API is a set of rules that define how applications or devices can connect to and communicate with each other.*
- Public (i.e. external) APIs are made to be used by anyone, whilst private APIs needs the user/developer to be authenticated before .
- Types of APIs:
    - Representational State Transfer (**REST**) APIs:
        - popular and easy-to-use APIs which use *HTTP* requests to get or change data.
    - Simple Object Access Protocol (**SOAP**) APIs
    - **GraphQL**
    - **Webhooks**
    - **Websockets**

Postman: API Testing framework



https://jsonplaceholder.typicode.com/posts/?userId=1

GET https://jsonplaceholder.typicode.com/posts/?userId=1

	KEY	VALUE	DESCRIPTION
✓	userId	1	
	Key	Value	Description

200 OK 729 ms 3.67 KB

```
[
    {
        "userId": 1,
        "id": 1,
        "title": "sunt aut facere repellat provident occaecati excepturi optio reprehenderit",
        "body": "quia et suscipit\nsuscipit recusandae consequuntur expedita et cum\nreprehenderit
            molestiae ut ut quas totam\nnostrum rerum est autem sunt rem eveniet architecto"
    },
    {
        "userId": 1,
        "id": 2,
```

Some APIs you can explore

- There are several public (free) APIs that you can explore:
    - JSON Placeholder API
    - Random User Generator API
    - Github API
    - Chuck Norris API
    - The Movies DB API
    - This link has a list of further APIs that are free to explore.

Python

Python

- Python is a high-level, interpreted programming language.
  - Interpreted programming languages are executed line-by-line by an interpreter, rather than being compiled into machine code beforehand.
- Python is known for its simplicity, readability, and ease of use.
  - Makes it easy to understand what is happening.
- Python's syntax is relatively easy to understand, and provides a repository of standard libraries.

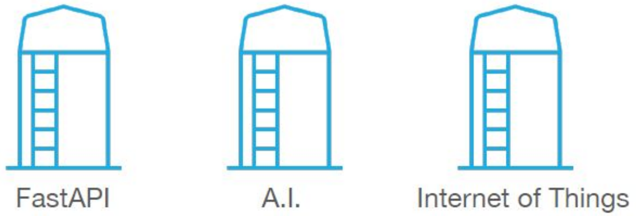
What can you build with Python

- Server-side web development using libraries like: Flask, Django, or FastAPI
- Scientific computing: Python provides libraries such as: numpy, SciPy, and Pandas to perform numerical computations, and data analytics.
- Machine Learning: Python libraries such as: TensorFlow, PyTorch, and Scikit-Learn enables developers to build, train, and deploy Machine Learning Models.
- Data Visualization: Python provides libraries such as: Matplotlib & Seaborn to create visualizations of data.
- Automation: Python can be used to automate tasks such as: testing, and web scraping.

Virtual Environments

Virtual Environments

- A virtual environment is a Python environment that is isolated from other Python environments.
- When working on python projects, it is highly recommended to setup a python environment that will handle the project's dependencies.
    - Using the same python environment in several projects may cause conflicts in dependencies, hence deploying different virtual environments for different projects.

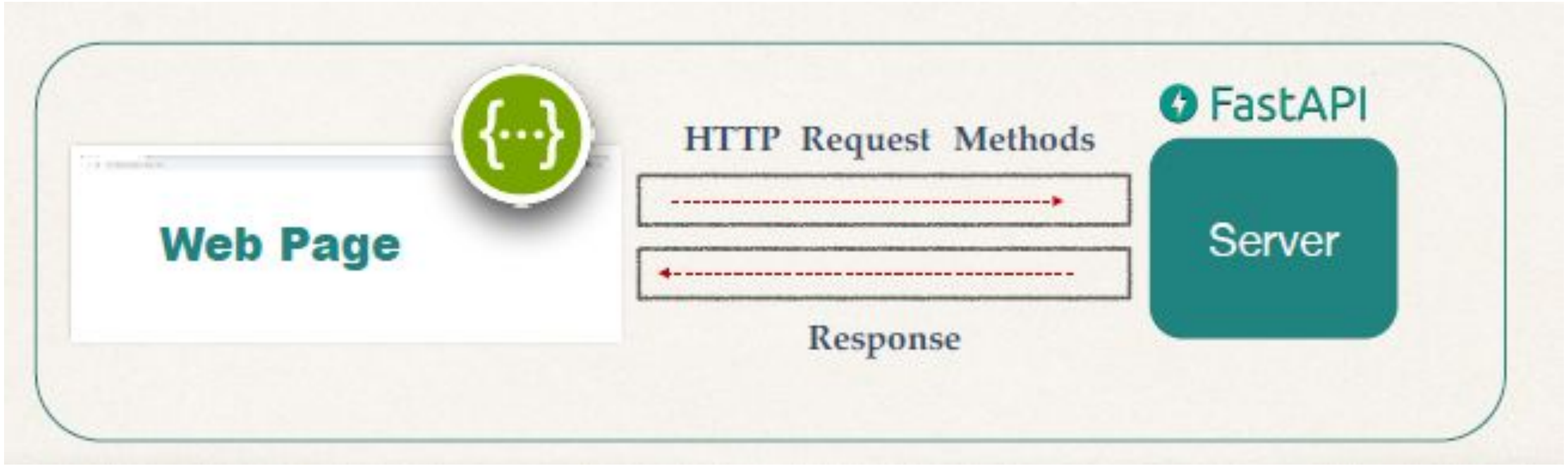


FastAPI

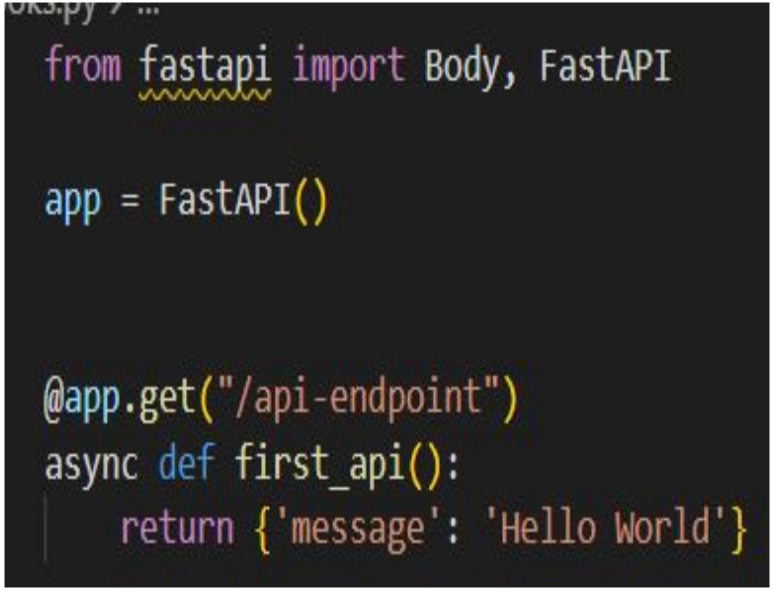
FastAPI

- FastAPI is a python based web-framework for building modern RESTful APIs.
  - Official documentation: https://fastapi.tiangolo.com/
- FastAPI enables developers to handle the business logic for the application.
- Companies such as: Uber, Netflix, and Microsoft use FastAPI.
- Why do I need a web framework?
  - It is possible to write everything yourself, however, it would be reinventing the wheel.
  - Web-frameworks allow for simplified methods to rapid development.

Where does FastAPI fit within the client-server architecture



Creating a FastAPI Application



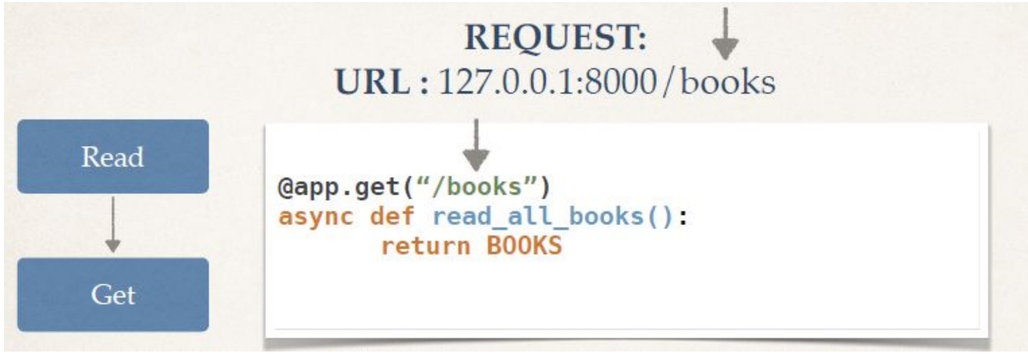
```
oks.py > ...
from fastapi import Body, FastAPI

app = FastAPI()


@app.get("/api-endpoint")
async def first_api():
    return {'message': 'Hello World'}
```

- **API Endpoints**
  - 127.0.0.1:8000/api-endpoint
- **Response:**

  {
    "message": "Hello World!"
  }
- **Running FastAPI application:**
  - uvicorn <*name_of_python_file*>:app -reload
- **URL**
  - 127.0.0.1:8000 (or localhost:8000)

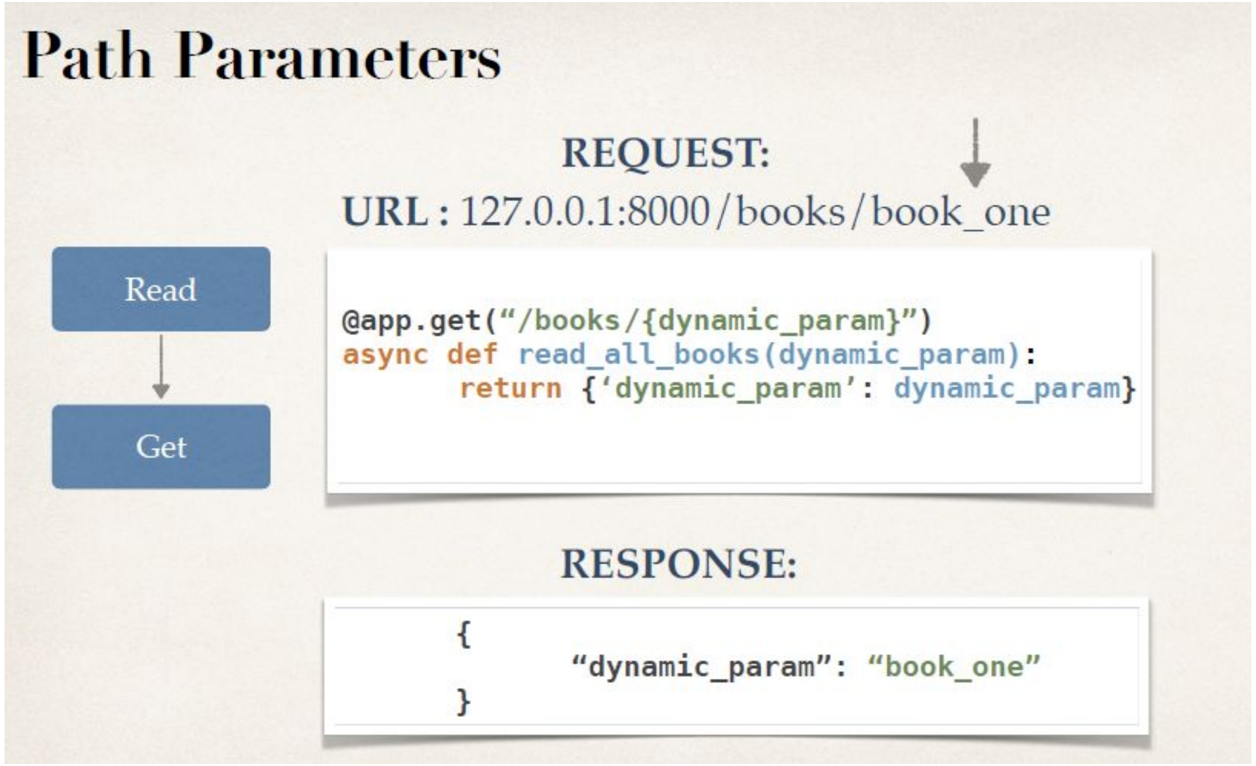
Path Parameters

- Path parameters are essentially request parameters that are attached to the URL.
- Path parameters are used to find information based on location



```
@app.get("/books")
async def read_all_books():
      return BOOKS
```

Path Parameters



REQUEST:

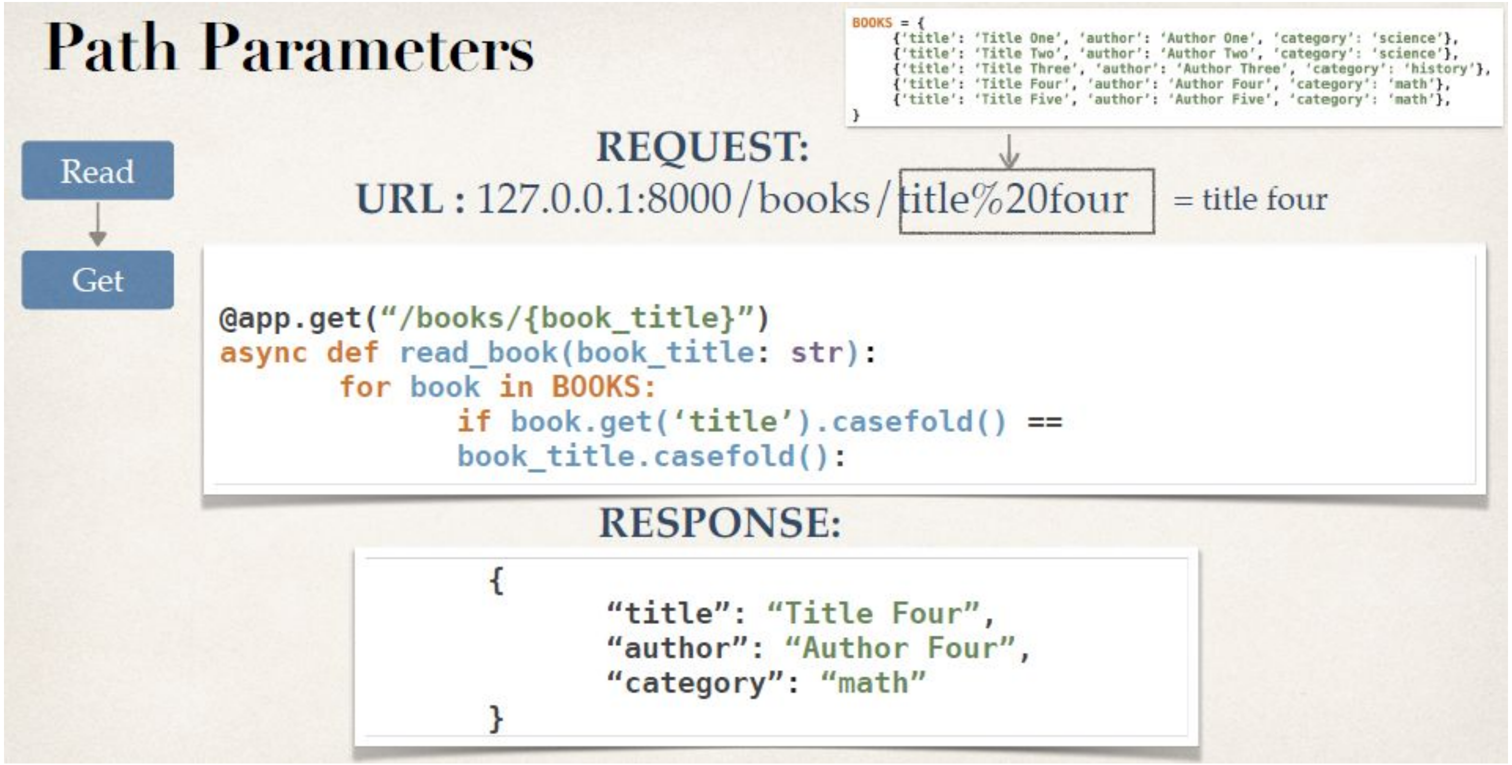
URL : 127.0.0.1:8000/books/book_one

```
@app.get("/books/{dynamic_param}")
async def read_all_books(dynamic_param):
        return {'dynamic_param': dynamic_param}
```

RESPONSE:

```
      {
              "dynamic_param": "book_one"
      }
```

Path Parameters



Path Parameters

```
BOOKS = {
    {'title': 'Title One', 'author': 'Author One', 'category': 'science'},
    {'title': 'Title Two', 'author': 'Author Two', 'category': 'science'},
    {'title': 'Title Three', 'author': 'Author Three', 'category': 'history'},
    {'title': 'Title Four', 'author': 'Author Four', 'category': 'math'},
    {'title': 'Title Five', 'author': 'Author Five', 'category': 'math'},
}
```

REQUEST:

URL : 127.0.0.1:8000 / books / $\boxed{\text{title\%20four}}$ = title four

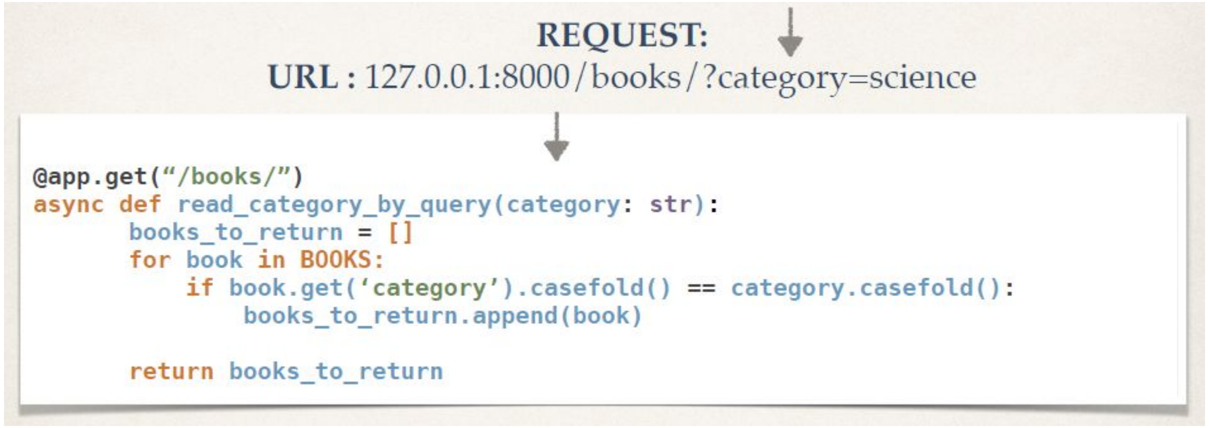
```
@app.get("/books/{book_title}")
async def read_book(book_title: str):
        for book in BOOKS:
                if book.get('title').casefold() ==
                book_title.casefold():
```

RESPONSE:

```
        {
                "title": "Title Four",
                "author": "Author Four",
                "category": "math"
        }
```

Query Parameters

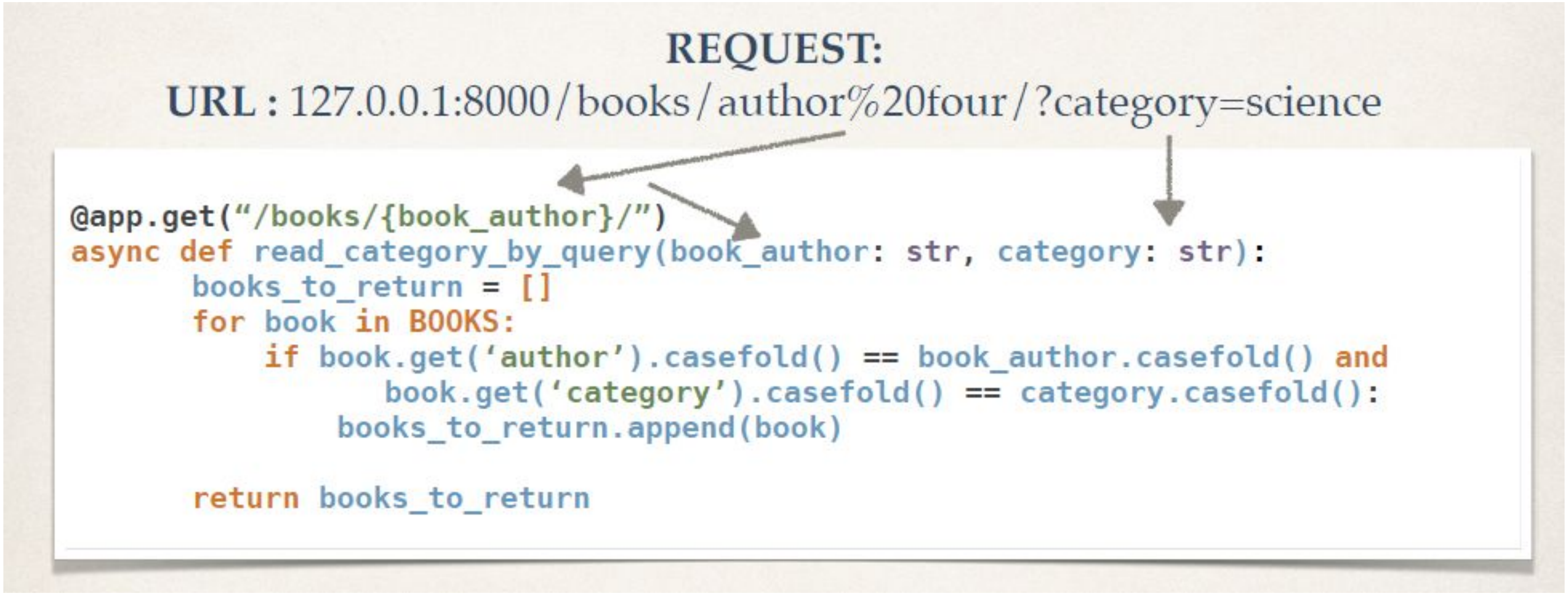
- Query parameters are essentially request parameters that are attached after “**?**”.
- Query parameters have ***name=value*** pairs.
- Example:
    - 127.0.0.1:8000/books/?category=math



REQUEST:
URL : 127.0.0.1:8000/books/?category=science

```
@app.get(“/books/”)
async def read_category_by_query(category: str):
        books_to_return = []
        for book in BOOKS:
            if book.get(‘category’).casefold() == category.casefold():
                books_to_return.append(book)

        return books_to_return
```

Query Parameters


REQUEST:

URL : 127.0.0.1:8000/books/author%20four/?category=science

```
@app.get("/books/{book_author}/")
async def read_category_by_query(book_author: str, category: str):
        books_to_return = []
        for book in BOOKS:
            if book.get('author').casefold() == book_author.casefold() and
                    book.get('category').casefold() == category.casefold():
                books_to_return.append(book)

        return books_to_return
```

Over to you

- The aim of this tutorial is to get first hand experience in launching a FastAPI application locally.
- To create a local FastAPI application:
  - Open a terminal in the directory you will be working from & create virtual python environment in the same directory that your application will be.
    - Check this article if you do not how to setup a python virtual environment.
  - Activate the newly created python virtual environment, and install the following libraries using pip:
    - ***pip install "fastapi[all]"***
  - Create a new Python script.
  - Follow the steps from this tutorial to launch a FastAP application locally.

Over to you

- The **aim** of this tutorial is to get first hand experience in creating various HTTP requests that the client access.
- First, create a python list of fake data. The data can be anything that you want, however it is in the similar structure of the BOOKS data we have been looking at in the tutorial.
    - If you are stuck, there is a dummy_data.txt on Vula that has examples of dummy data that you can use.
- Create various endpoints. Below are some examples:
    - GET all data
    - GET data if a particular field(s) match
    - POST (create) new data
    - PUT (Update) an existing data entry
    - DELETE an existing data.
- By the end of the tutorial, you should be able to perform a short demo for the class.

Handling Client Requests

Data Validation, Exception Handling & Status Codes

- In backend development, it is important to have mechanisms in place that handles client requests.
- Data Validation
  - In order to process a client's request; the data must be valid, and there are safeguards that must be put in place to handle situations where the data sent to the server is invalid.
  - In FastAPI, we can use Pydantics python library. Pydantics is used for data modelling, data parsing, and and has efficient error handling.
- Exception Handling
  - Sometimes, a client's request may be invalid, or the resource the client is request is not available. In server-side development, we need to make sure that these cases are handled appropriately.
- Status codes.
  - An HTTP status code is used to help the client understand what happened on the server-side application when a request is made.

Practical

- The **aim** of this practical is to create your own web server using FastAPI.
- The web server will allow users to track their TODO/Task list. For each TODO (i.e. task) item, it should have the following properties:
    - Id: String value
    - Task Title: String value
    - Completed: Boolean
- Create various endpoints. Below are some examples:
    - GET all Tasks
    - GET Tasks if a particular field(s) match
    - POST (create) new Task
    - PUT (Update) an existing Task
    - DELETE an existing Task.
- For each endpoint make sure you include: data validation, HTTP Exception, and status code.