# Virtual environments in Python

Sergio Paniego Blanco
@sergiopaniego
sergiopaniegoblanco@gmail.com
https://sergiopaniego.github.io/

# REST APIs

- Basic concepts of APIs.
- Introduction to RESTful APIs.
- HTTP methods: GET, POST, PUT, DELETE
- Requests handling and HTTP answers in Python
- Creation of a basic API REST using Django

# Basic concepts of APIs

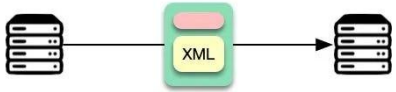
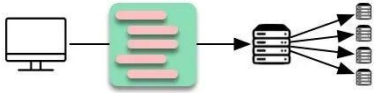
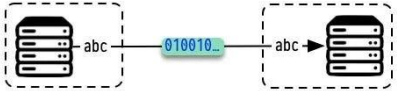
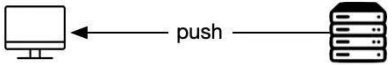
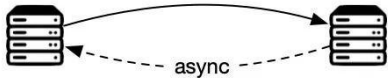
- API (Application Programming Interface)
  - Set of rules and protocols that allows different software applications to communicate with each other.
  - Defines the methods and data formats that applications can use to request and exchange information.
  - Features:
    - Interface for communication
    - Standardized protocols (HTTP/HTTPS…)
    - Requests and responses.

# Basic concepts of APIs

- Role of APIs in software development:
  - Modularity
  - Integration of services
  - Easier collaboration
  - Scalability
  - Cross-platform functionality
- How APIs allow different software applications to communicate:
  - Encapsulation of functionality
  - Data exchange
  - Asynchronous communication
- Types of APIs: web APIs, library APIs, operating system APIs…

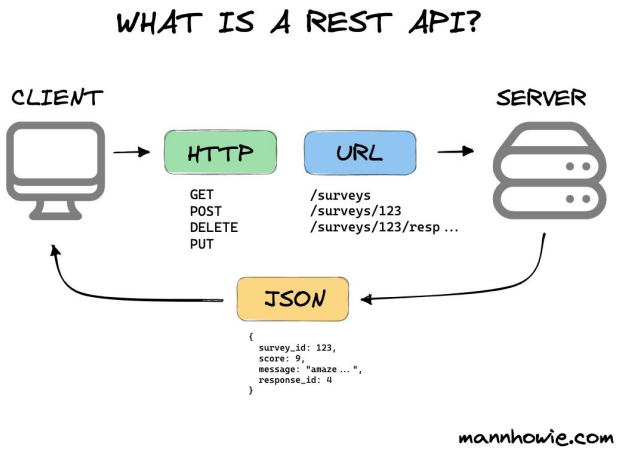
# Basic concepts of APIs

Top 6 Most Popular API Architecture Styles ByteByteGo.com

| Style | Illustration | Use Cases |
|---|---|---|
| SOAP |   | XML-based<br>for enterprise applications |
| RESTful |   | Resource-based<br>for web servers |
| GraphQL |  | Query language<br>reduce network load |
| gRPC |   | High performance<br>for microservices |
| WebSocket |   | Bi-directional<br>for low-latency data exchange |
| Webhook |   | Asynchronous<br>for event-driven application |

# Introduction to RESTful APIs

- REST (Representational State Transfer): architectural style for designing networked apps.
  - Uses standard HTTP methods
  - Used in web development to enable communication between client and server.



# Introduction to RESTful APIs

- Principles:
  - Statelessness: each request must contain all the information needed to understand and process the request. The server doesn't store client context
  - Client-server architecture.
  - Resource-based: resources are the objects or data the API exposes. Each one has a unique URI. Clients interact using HTTP methods (`GET`, `POST`, `PUT`, `DELETE`) to perform CRUD (`Create`, `Read`, `Update`, `Delete`) operations.
  - The resources can have multiple representations (`JSON`, `XML`, `HTML...`)
  - Uniform interface: simplifies architecture.

# HTTP methods

- Methods:
  - GET: Retrieve a resource or a collection of resources.
  - POST: Create a new resource.
  - PUT: Update an existing resource or create one if it doesn't exist.
  - DELETE: Remove a resource.
  Resource URIs:
- Unique URIs: for example `/users` or `/users/123`

**/books**

| | | |
|---|---|---|
| GET | /books | Lists all the books in the database |
| DELETE | /books/{bookId} | Deletes a book based on their id |
| POST | /books | Creates a Book |
| PUT | /books/{bookId} | Method to update a book |
| GET | /books/{bookId} | Retrieves a book based on their id |

# HTTP status codes

- The server uses standard HTTP status codes to indicate the outcome of a request. Common status codes include:
  - 200 OK: The request was successful.
  - 201 Created: A new resource was created successfully.
  - 404 Not Found: The requested resource was not found.
  - 500 Internal Server Error: An unexpected error occurred on the server.

**HTTP Status Codes** javaconceptoftheday.com

| 1xx : Informational Purpose | | 4xx : Client Errors | | 5xx : Server Errors | |
|---|---|---|---|---|---|
| 100 | Continue | 400 | Bad Request | 500 | Internal Server Error |
| 101 | Switching Protocols | 401 | Unauthorized | 501 | Not Implemented |
| 102 | Processing | 402 | Payment Required | 502 | Bad Gateway |
| 103 | Early Hints | 403 | Forbidden | 503 | Service Unavailable |
| **2xx : Success** | | 404 | Not Found | 504 | Gateway Timeout |
| 200 | Ok | 405 | Method Not Allowed | 505 | HTTP Version Not Supported |
| 201 | Created | 406 | Not Acceptable | 507 | Insufficient Storage |
| 202 | Accepted | 407 | Proxy Authentication Is Required | 508 | Loop Detected |
| 203 | Non-Authoritative Information | 408 | Request Time Out | 510 | Not Extended |
| 204 | No Content | 409 | Conflict | 511 | Network Authentication Required |
| 205 | Reset Content | 410 | Gone | | |
| 206 | Partial Content | 411 | Length Required | | |
| 207 | Multi Status | 412 | Precondition Failed | | |
| 208 | Already Reported | 413 | Payload Too Large | | |
| 226 | IM Used | 414 | URI Too Long | | |
| **3xx : Redirection** | | 415 | Unsupported Media Type | | |
| 300 | Multiple Choices | 416 | Range Not Satisfiable | | |
| 301 | Moved Permanently | 417 | Expectation Failed | | |
| 302 | Found | 421 | Misdirect Request | | |
| 303 | See Other | 422 | Unprocessable Entity | | |
| 304 | Not Modified | 423 | Locked | | |
| 305 | Use Proxy | 424 | Failed Dependency | | |
| 306 | No Longer Used | 425 | Too Early | | |
| 307 | Temporary Redirect | 426 | Upgrade Required | | |
| 308 | Moved Permanently | 428 | Precondition Required | | |
| | | 429 | Too Many Requests | | |
| | | 431 | Request Header Fields Too Large | | |
| | | 451 | Unavailable For Legal Reasons | | |

# Example of REST in Action

- RESTful API for Managing Students in a School
  - Retrieve All Students
    - Request: `GET /students`
    - Response: Returns a JSON array with details of all students.
  - Retrieve a Specific Student
    - Request: `GET /students/1`
    - Retrieve a Specific Student Request: `GET /students/1`
  - Add a New Student
    - Request: `POST /students`
    - Body: JSON with student data
    - Response: 201 created

# Example of REST in Action

- RESTful API for Managing Students in a School
  - Update an Existing Student
    - Request: `PUT /students/1`
    - Body: updated student data
    - Response: Returns a status of 200 OK with the updated details of the student.
  - Delete a Student:
    - Request: `DELETE /students/1`
    - Response: Returns a status of 204 No Content, indicating the student with ID 1 has been successfully deleted.

# Requests in Python

- Using requests library https://requests.readthedocs.io/en/latest/

```
pip install requests
```

# Requests in Python

# ACTIVITY

- Use https://catfact.ninja/ `API` and `requests` to:
  - Retrieving a random cat fact and a list of facts (limit to 3).
  - Simulate an error by accessing a non-existent endpoint, and handle different `HTTP` status codes.
  - Practice working with the response in both `JSON` and raw text formats.

# Creation of basic API REST using Django

- Setup and requirements

```
pip install django djangorestframework
```

- Start a new Django project

```
django-admin startproject library_project
```

- Create a new Django app

```
cd library_project
python manage.py startapp books
```

# Creation of basic API REST using Django

- Define a new model in `models.py`
- Migrate the database

```
python manage.py makemigrations
python manage.py migrate
```

- Create a serializer for the Book model to convert model instances to `JSON` format (`books/serializers.py`)
- Define `API` views (`books/views.py`)
- Setup `URL` configuration (`books/urls.py` and `library_project/urls.py`)
- Update `INSTALLED_APSS` with `rest_framework` and books (`library_project/settings.py`)

# ACTIVITY

- Add a Summary Field to the Book API
- Update the existing Book API by adding a new field called summary to the Book model.
- This field should be a CharField with a maximum length of 255 characters.
- After updating the model, modify the serializer to include the new field, and test the changes by creating a new book with a summary using `Postman` or `requests`