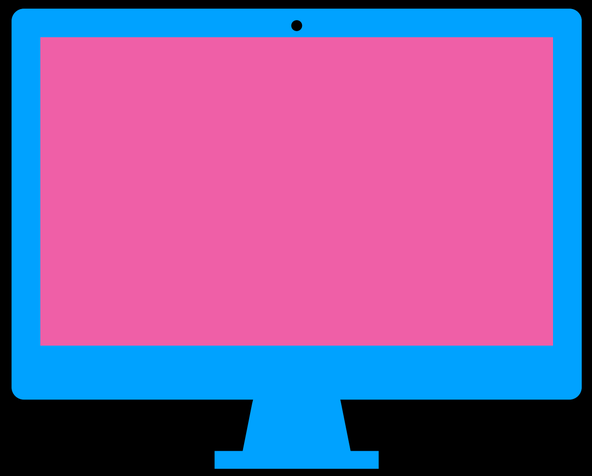
PART I: REST API

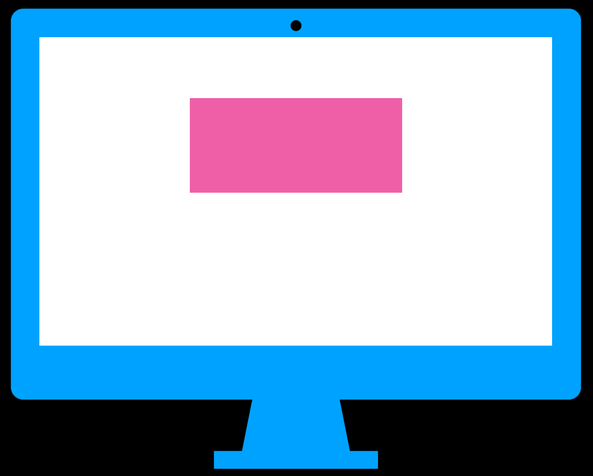
Single Page Application

- Web applications that works inside a browser and enables asynchronous loading of elements (components)
- Exchange of data or updating in this type of application is being done without refreshing the page
- Like Post example:

Traditional : full page reload



SPA: updated particular post



Key concepts

- API - stands for „**A**pplication **P**rogramming **I**nterface” and enables communication between applications.
- REST - stands for „**RE**presentational **S**tate **T**ransfer” and determines how the API should look like with a set of rules and standards. In REST:
  - We use uniform interface which basically means standardized communication between the server and the client. This rule allows using various devices/ applications to communicate with the server using a single interface.
  - We access resources (pieces of data) with the use of endpoints
  - Communication between the server and client is based on HTTP protocol using methods: GET, POST, PUT, PATCH, DELETE and the client and the server are separate apps that can be developed independently
  - We rely on stateless - each transaction is performed as if it was for the first time (no history). The server doesn’t store any state about the client session on the server side
  - Representation of the state is in JSON or XML format

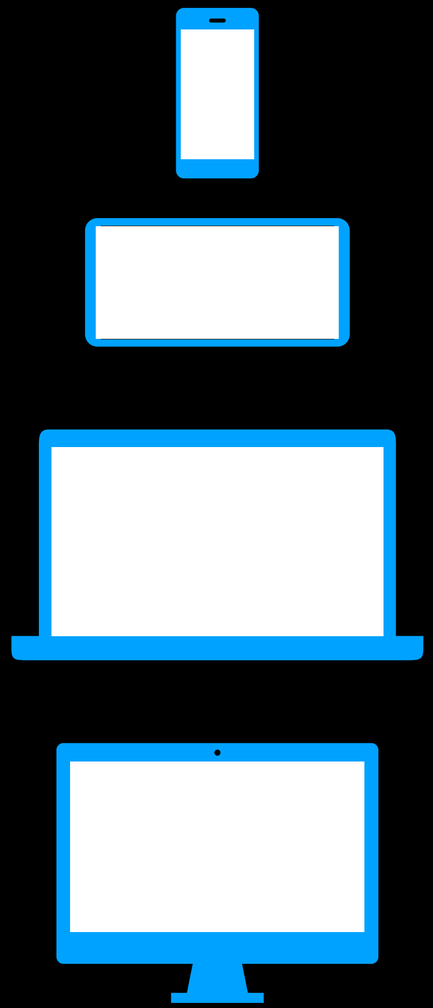
Endpoints

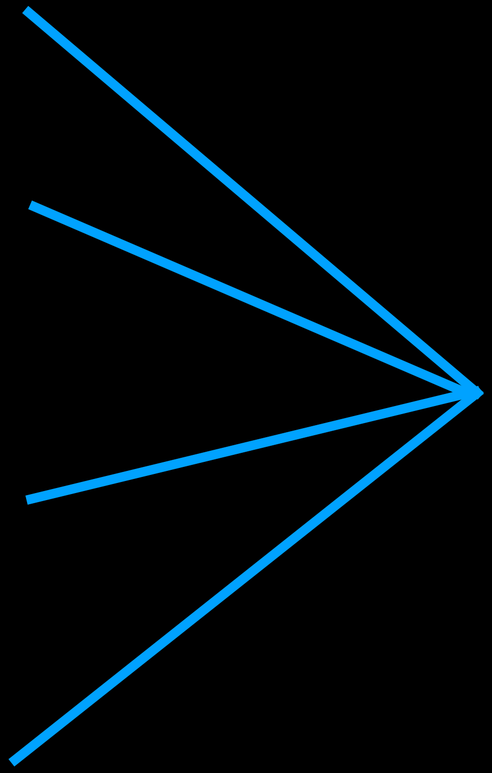
GET	https://whatimagesite.com/api/images
GET	https://whatimagesite.com/api/images/1
POST	https://whatimagesite.com/api/images
PUT	https://whatimagesite.com/api/images/1
DELETE	https://whatimagesite.com/api/images/1

Scheme

Button click

On the frontend, associated with a endpoint and HTTP method





JSON

API

Database

Status code

- 200 – OK
- 201 – CREATE
- 204 – NO CONTENT
- 304 – NOT MODIFIED
- 400 – BAD REQUEST
- 401 – UNAUTHORIZED
- 403 – FORBIDDEN
- 404 – NOT FOUND
- 500 – INTERNAL SERVER ERROR

JSON format

JSON and dictionaries look for similar to each other, however JSON is a data format (string) while a dictionary in python is a data structure.

```python
import json

meals = ['pizza', 'pasta', 'soup']
price = [15, 12, 7]

menu = dict(zip(meals, price))
print(menu)
print(type(menu))
# output:
# {'pizza': 15, 'pasta': 12, 'soup': 7}
# <class 'dict'>

menu2 = json.dumps(menu)
print(menu2)
print(type(menu2))
# output:
# '{"pizza": 15, "pasta": 12, "soup": 7}'
# <class 'str'>
```

```python
def json_check(data):
    try:
        json.loads(data)
    except:
        return False
    return True

print(json_check(menu))
# output:
# False

print(json_check(menu2))
# output:
# True
```

Serialization - In python this is a process of translating data structure into JSON format

Summary

- An API enables a standardized way of communication with other applications, while REST determines how the API should look like

- In our project the frontend (React) will talk to the backend (Django). Thanks to the API that we are going to create, React will be able to communicate with the database, without reloading the entire page sending and receiving data in JSON

PART II: DRF

Basic information

- First step is to install Django Rest Framework
- Next we need to create an api directory inside of our application folder named “api” which will consists of 3 files:
  - serializers.py
  - views.py
  - urls.py

serializers

- From DRF documentation:
  - “Serializers allow complex data such as querysets and model instances to be converted to native Python datatypes that can then be easily rendered into JSON, XML or other content types”
- In our project we will use ModelSerializer to serialize objects based on a Model. In order to do that we need to create a serializer class and inside of it with the use of class Meta - indicate the model and the fields

views

- In our project in the views.py file we are going to create a ModelViewSet to handle our views
- ViewSets together with Routers (urls slide) enable speeding up the implementation of our API because we don't have to create separate views since inside the viewset class we have available methods such as : list, create, retrieve, update, partial_update, destroy
- We will be using a ModelViewset which we’ll create by providing a queryset (related to the model) and serializer class
- In the future you can extend this part with i.e. authentication classes or permission classes

views

Generic Views

```
class ImageList(ListCreateAPIView):
    queryset = Image.objects.all()
    serializer_class = ImageSerializer

class ImageDetail(RetrieveUpdateDestroyAPIView):
    queryset = Image.objects.all()
    serializer_class = ImageSerializer
```

Separate views for listing the objects and getting the detail, updating and deleting

Viewsets (ModelViewset)

```
class ImageViewSet(viewsets.ModelViewSet):
    queryset = Image.objects.all()
    serializer_class = ImageSerializer
```

In this case we can access all the CRUD actions using one single view.

urls

- The urls.py file of the images application and the main project urls.py will be connected
- In the urls.py file of images app, we will create a router that will connect our ModelViewSet into some standarized structure of URLs and we won’t need to take care of this manually.
- This is another tool that works with ViewSets to speed up the implementation of our API