
# Coding and Git

Alessandro Bocci

name.surname@unipi.it

Advanced Software Engineering (Lab)

27/09/2024

# What will you do?

- Hands-on with git and GitHub
- Prototype a service in Python



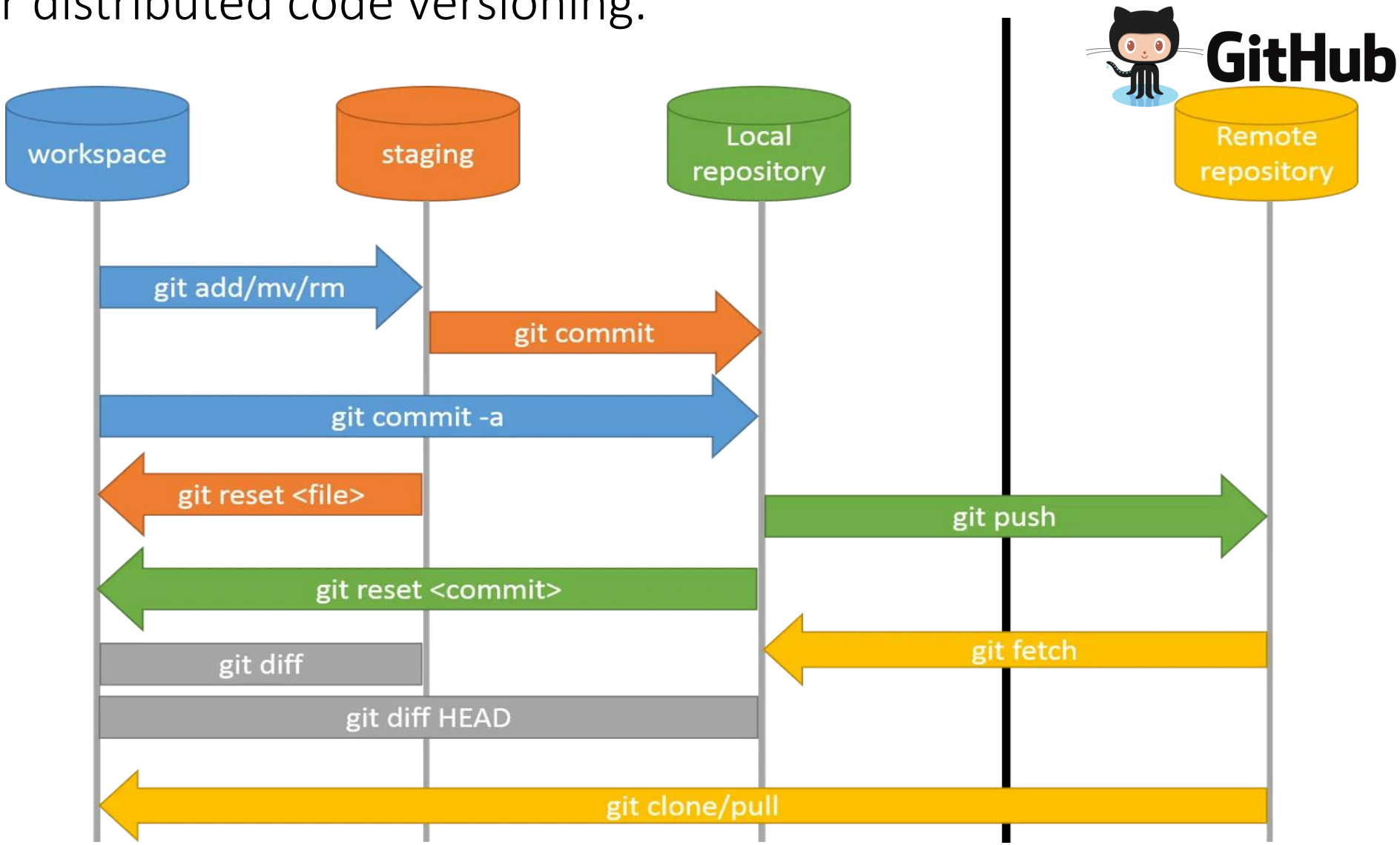
# Checklist

- Download the material folder on the Moodle website.
- Install **Python 3.12** [https://www.python.org/]
- Install an **IDE/editor** of your choice
  (e.g., Visual Studio Code [https://code.visualstudio.com/], PyCharm [https://www.jetbrains.com/pycharm/download/] )
- Install **Git** [https://git-scm.com/downloads]
- Create your **GitHub** account [http://github.com]
- Set up git with Github [https://docs.github.com/en/get-started/getting-started-with-git/set-up-git#setting-up-git]



# What is git?

Software for distributed code versioning.



# How to create a Repo

- Go to github.com and enter with your credentials.
- Repositories are the place where your projects live.
- In the upper-right corner of any page, click **+** and then **New Repository**.
- Type a short, memorable name, e.g. `ase-lab1-24`.
- This repo will be **Public**.
- Initialise it with a **README**.
- .gitignore template: None.
- Click **Create a repository**.

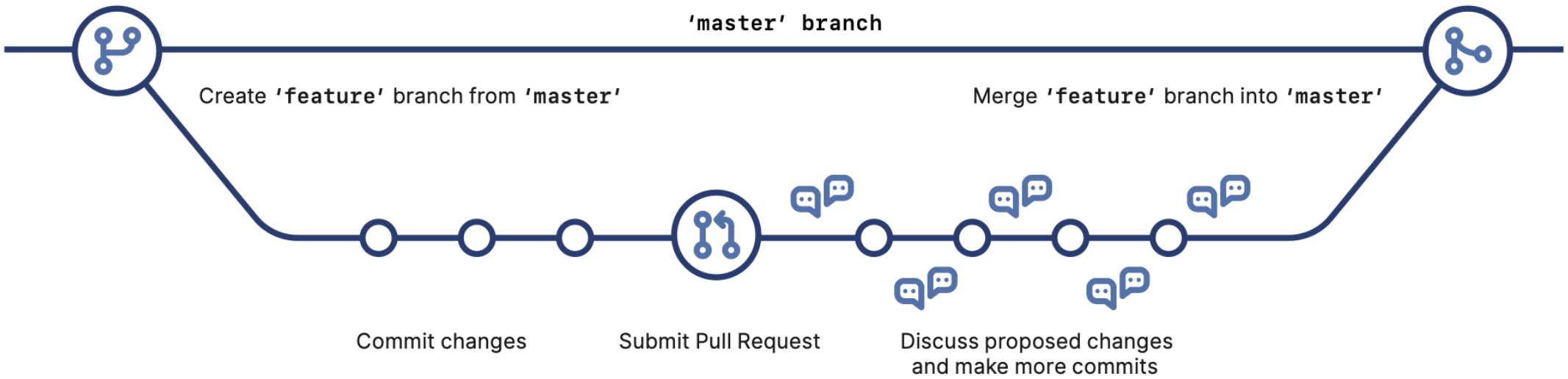
# How to clone your Repo

- Open a terminal.
- Move to the directory where you want to store your work (e.g., `ASE`).
- Use the command `git clone [your repo url]`.
- Move the content of the `app` folder from the `material.zip` archive to the project folder. Then:

```
git add *
git commit -m "first commit"
git push
```

- Check on the repo website.

## GitHub Flow




https://guides.github.com/introduction/flow/

# Manage branches

- Create a new branch

```
git branch [name_of_your_new_branch]
```

- Move to the branch on your local machine

```
git checkout [name_of_your_new_branch]
```

- Push it on GitHub

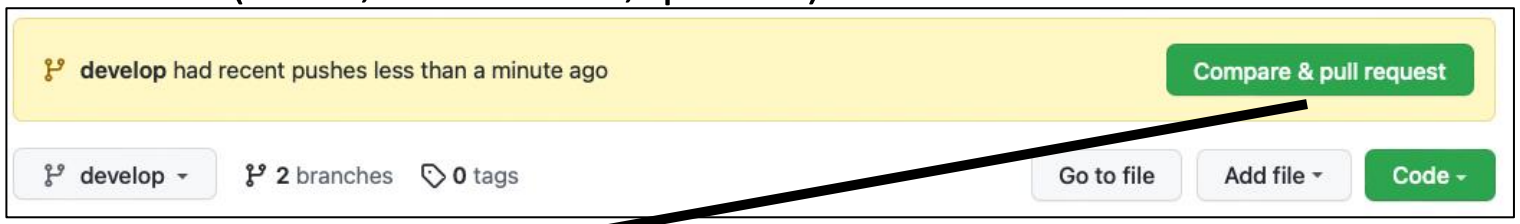
```
git push -u origin [name_of_your_new_branch]
```

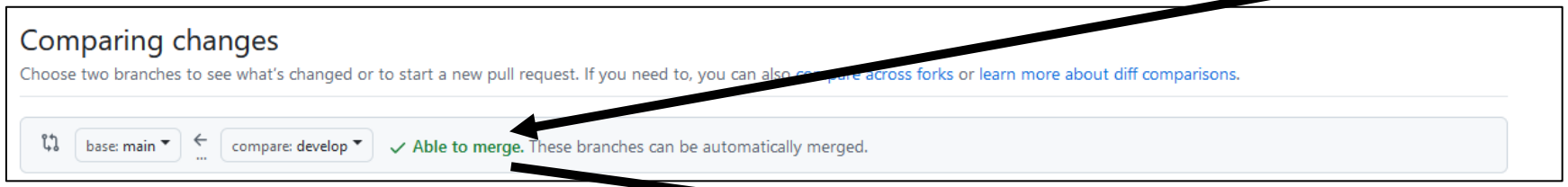
- List all branches with

```
git branch -a
```

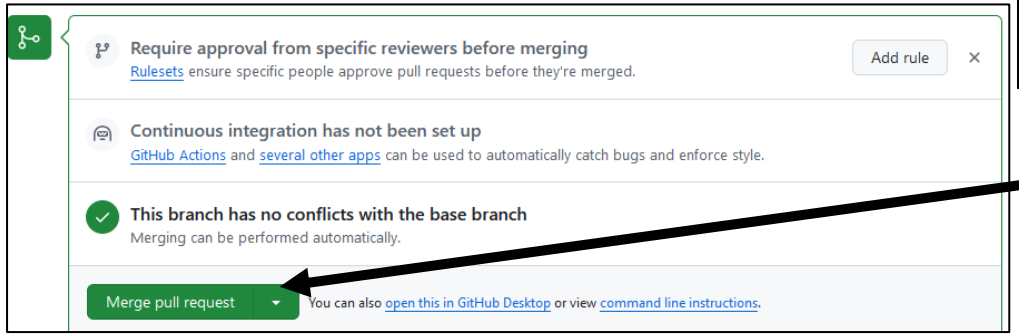
# Merge branches

- Create a new file `touch pippo.txt`
- Update the remote branch (add, commit, push)
- Check the website








Then, confirm, delete the branch and switch to the main branch.

# Merge branches

- Note 1: When you use the web interface the changes are not reflected automatically on your local machine.

  After moving on the main branch, delete locally the other branch.

  ```
  git branch -D [name_of_your_new_branch]
  ```

- Note 2: What we did via the web interface can also be done with git commands in the terminal.

What happens when two branches cannot be automatically merged?

# Before coding with Python – venv

When you install Python packages on your machine, it is advised to use a virtual environment to avoid package conflicts.

In general: one environment per project.

From the terminal:

- Create a venv `python -m venv <path>` This will create a folder in the specified path.
- Activate the venv `source <path>/bin/activate`
- Now everything installed with pip and every python execution will refer to the venv.
- To exit the venv `deactivate` or close the terminal.

Note that the commands may change depending on your system.

# Before coding with Python – venv

When you install Python packages on your machine, it is advised to use a virtual environment to avoid package conflicts.

In general: one environment per project.

From the terminal: Only the first time!

- Create a venv `python -m venv <path>` This will create a folder in the specified path.
- Activate the venv `source <path>/bin/activate`
- Now everything installed with pip and every python execution will refer to the venv.
- To exit the venv `deactivate` or close the terminal.

Note that the commands may change depending on your system.

# How to avoid to push the env

In the root folder of your repo create (or modify if it exists) the file named

**`.gitignore`**

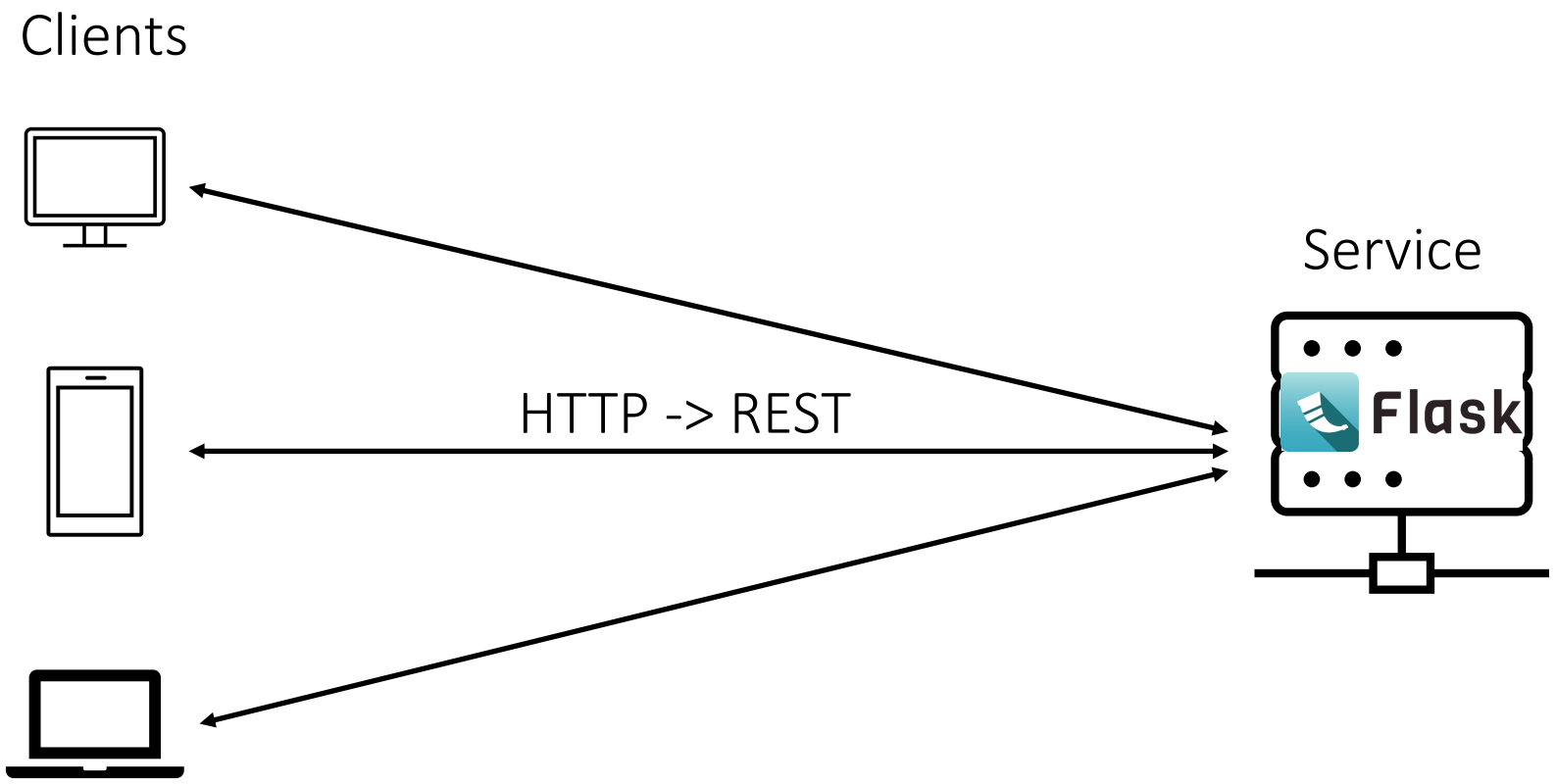
In this file you can specify the files and the folder that will be not considered by git.

Add the name of the folder you specified as <path> when using venv.

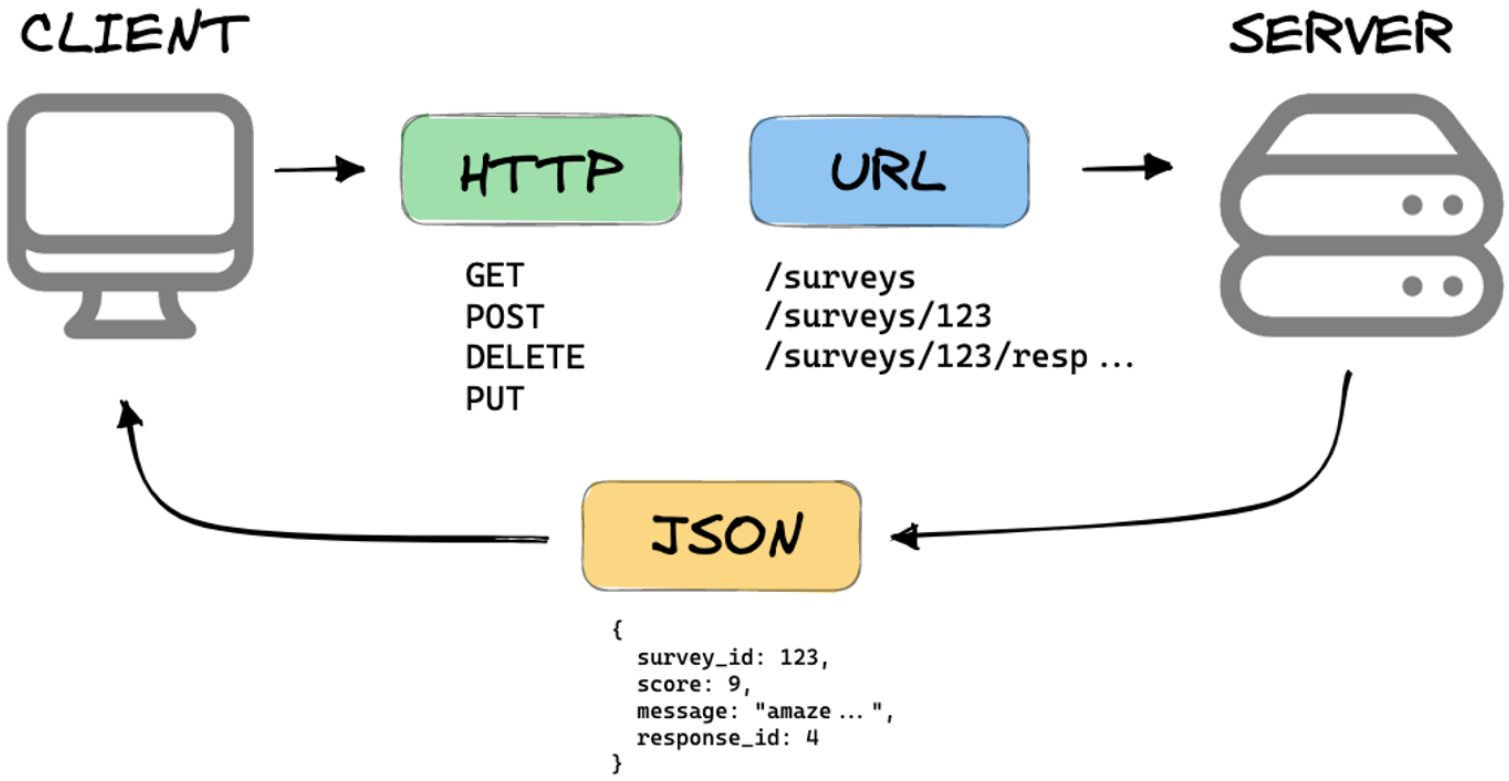
Windows users: activate the option to show hidden files.

# Let’s code a web service! - Flask

Flask is a web application (service) framework written in Python.

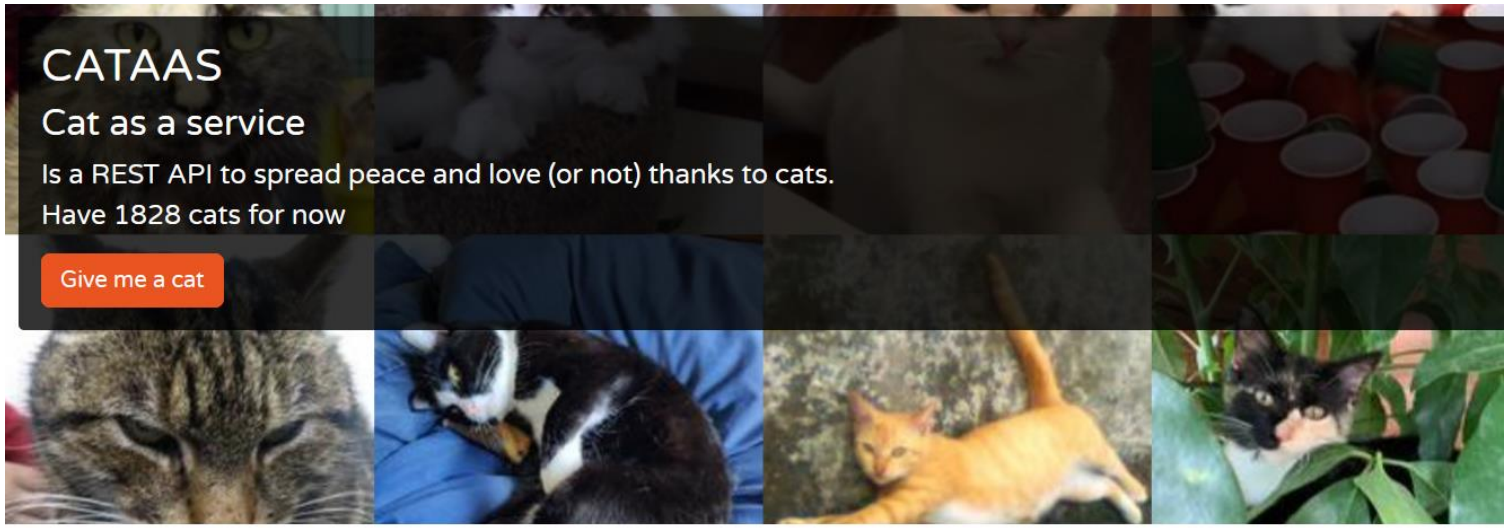


# RESTful API




https://mannhowie.com/rest-api

# RESTful API example - CATaaS




Hey ! Do you like Cataas? Do you want to support the project ? Buy me a beer

Hey cat lovers, new major version of cataas :

- Now JSON will returns if request contain `application/json` header, API doc updated here
- Fix tags search and fix tags combined with "," separator (see documentation)

## Basic

| Url | Description | Example |
|---|---|---|
| **/cat** | Will return a random cat | Random cat |
| **/cat/:tag** | Will return a random cat with a `:tag`, You can combine multiple tags with `:tag` separator | Random orange cute cat |
| **/cat/gif** | Will return a random gif cat \o/ | Random gif cat |
| **/cat/says/:text** | Will return a random cat saying `:text` | Random cat saying hello |
| **/cat/:tag/says/:text** | Will return a random cat with a `:tag` and saying `:text` | Random cute cat saying hello |
| **/cat/says/:text?fontSize=:size&fontColor=:color** | Will return a random cat saying `:text` with text's `:fontSize` and text's `:fontColor` | Custom random cat saying hello |

https://cataas.com/

# app.py

```
5   @app.route('/add')
6   def add() -> Any:
7       a = request.args.get('a', type=float)
8       b = request.args.get('b', type=float)
9       if a and b:
10          return make_response(jsonify(s=a+b), 200) # HTTP 200 OK
11      else:
12          return make_response('Invalid input\n', 400) # HTTP 400 BAD REQUEST
```

Python code to generate an HTTP GET response to

`http://<service_host>/add?a=X&b=Y`

where X and Y are numbers.

# app.py

```
5   @app.route('/add')
6   def add() -> Any:
7       a = request.args.get('a', type=float)
8       b = request.args.get('b', type=float)
9       if a and b:
10          return make_response(jsonify(s=a+b), 200) # HTTP 200 OK
11      else:
12          return make_response('Invalid input\n', 400) # HTTP 400 BAD REQUEST
```

Route: specifies the path (`/add`) and can specify the HTTP method (default = GET). It is a Flask annotation.

A route is and endpoint part of the REST API of our service.

`http://<service_host>/add?a=X&b=Y`

# app.py

```python
@app.route('/add')
def add() -> Any:
    a = request.args.get('a', type=float)
    b = request.args.get('b', type=float)
    if a and b:
        return make_response(jsonify(s=a+b), 200) # HTTP 200 OK
    else:
        return make_response('Invalid input\n', 400) # HTTP 400 BAD REQUEST
```

Python function that is called when arrives a request for a route.

`http://<service_host>/add?a=X&b=Y`

# app.py

```
5   @app.route('/add')
6   def add() -> Any:
7       a = request.args.get('a', type=float)
8       b = request.args.get('b', type=float)
9       if a and b:
10          return make_response(jsonify(s=a+b), 200) # HTTP 200 OK
11      else:
12          return make_response('Invalid input\n', 400) # HTTP 400 BAD REQUEST
```

Obtaining query string parameters **a** and **b**.

`http://<service_host>/add?a=X&b=Y`

# app.py

```
5   @app.route('/add')
6   def add() -> Any:
7       a = request.args.get('a', type=float)
8       b = request.args.get('b', type=float)
9       if a and b:
10          return make_response(jsonify(s=a+b), 200) # HTTP 200 OK
11      else:
12          return make_response('Invalid input\n', 400) # HTTP 400 BAD REQUEST
```

Response to the client, with payload and HTTP code.

`http://<service_host>/add?a=X&b=Y`

# How to run a Flask service

From the terminal (inside an activated venv):

- Install the service requirements `pip install -r requirements`
- Run the service `flask run --host=0.0.0.0 --port=5005`

  Now the service will be waiting on *any* network interface (0.0.0.0) at *port* 5005.

  The terminal will log the events of the service, to close the service and take control of the terminal ctrl+C (Command+C).

  If the 5005 port is used by another service of your system, you can change it.

- Open a browser and visit `http://localhost:5005/add?a=30&b=12`

You should see the result of the execution of the code.

# Now it’s your turn

1. Create the repo.
2. Create a new branch of your repo.
3. Complete the REST API of the `app.py` by coding the commented routes.
4. Merge the branch in the main one.



# BONUS STAGE!

# Bonus stage

- Add new endpoints to the REST API
  - `/upper` which given the string `a` it returns it in a JSON all in uppercase.
  - `/lower` which given the string `a` it returns it in a JSON all in lowercase.
  - `/concat` which given the strings `a` and `b` it returns in a JSON the concatenation of them.
  - `/reduce` which takes the operator `op` (one of add, sub, mul, div, concat) and a `lst` string representing a list and apply the operator to all the elements giving the result. For instance, `/reduce?op=add&lst=[2,1,3,4]` returns a JSON containing {s=10}, meaning 2+1+3+4.
  - `/crash` which terminates the service execution after responding to the client with info about the host and the port of the service.
  - `/last` which returns a string representing the last operation requested with success, in the format op(args)=res, e.g. `add(2.0,3.0)=5.0` or `reduce('add',[2,1,3,4])=10` or `rand(1,3)=2`. It answers with HTTP code 404 if no operation was performed.

    Hint: to do this you have to modify the other endpoints and use a file.

# Lab take away

- ❑ Create a GitHub repository
- ❑ Play with git branches
- ❑ Code and run a Python service

# Project take away

- ❑ The project will be delivered in a GitHub repository.
- ❑ It will be a service with a REST API.
- ❑ Python language it is not mandatory.