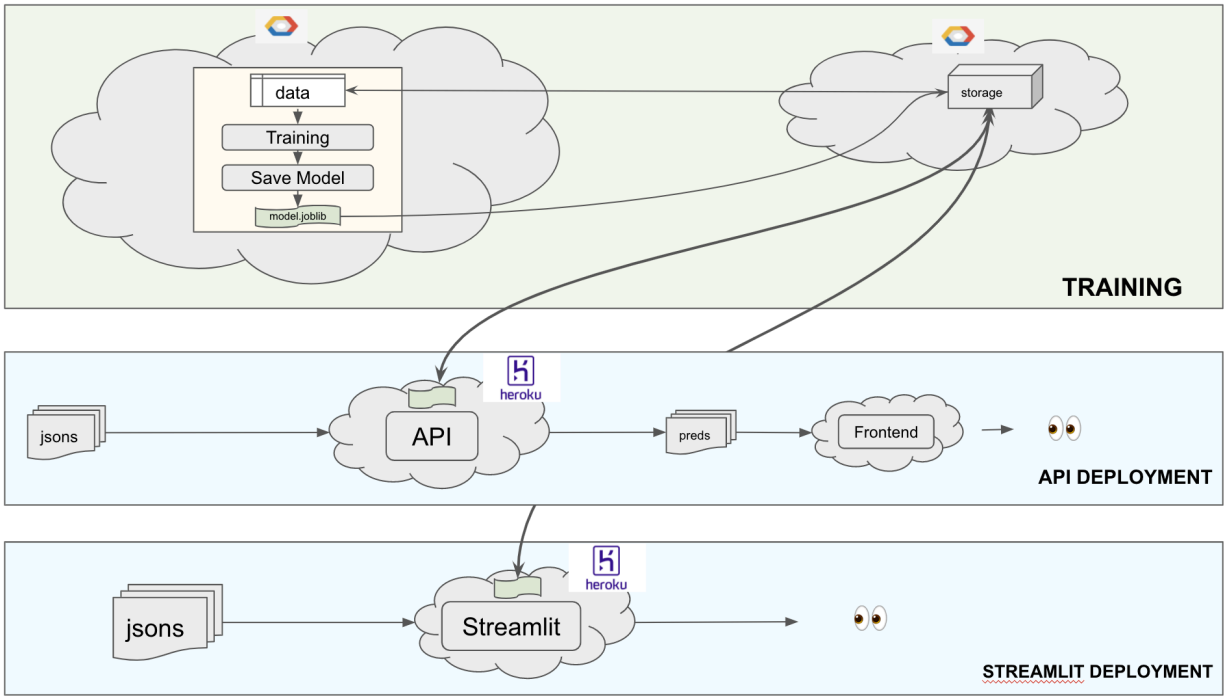
User Interface

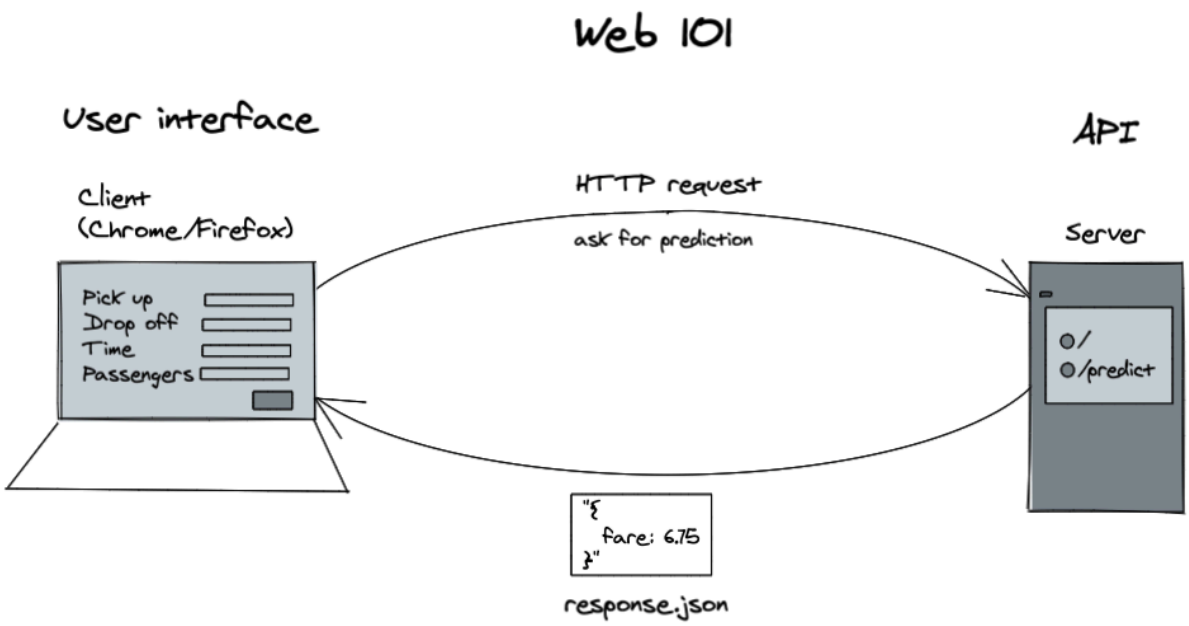
Reminders from Predict in Production

Objective

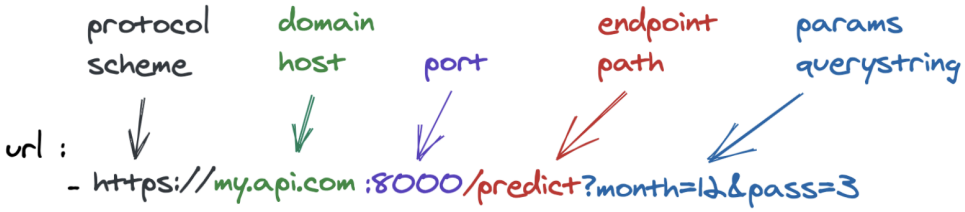


Web Behind the Scenes

Requesting a Resource Using HTTP



HTTP Request



verb :

- GET : read (params in the url)
- POST : create (params in the body)
- PUT, PATCH : update (params in the body)
- DELETE : delete

headers

(body)

HTTP Response

code :

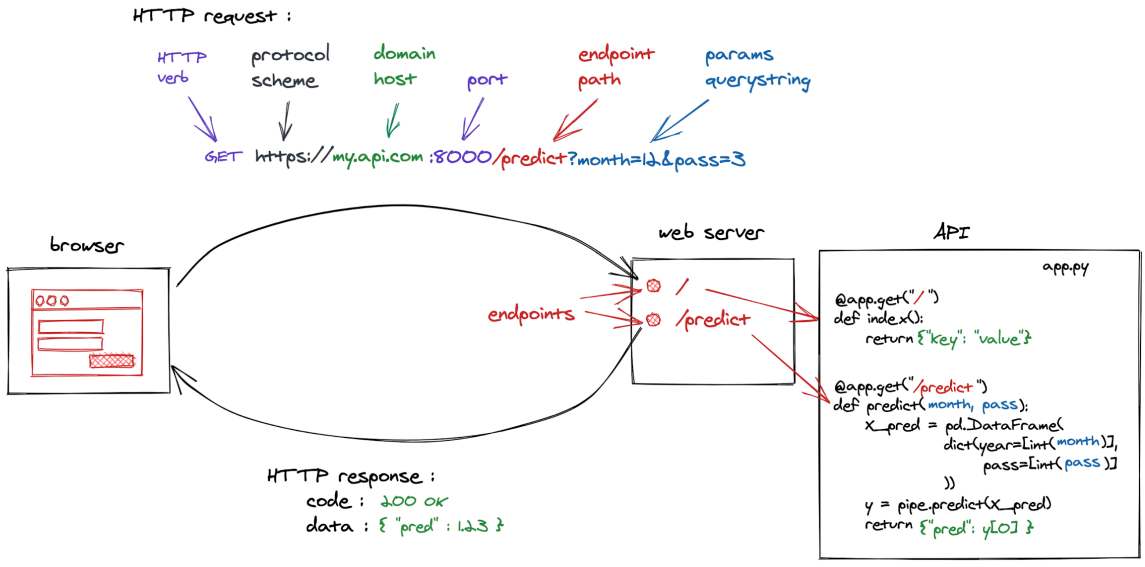
- 2xx : everything is ok
- 3xx : redirection
- 4xx : user made a mistake
- 5xx : developer made a mistake

data :

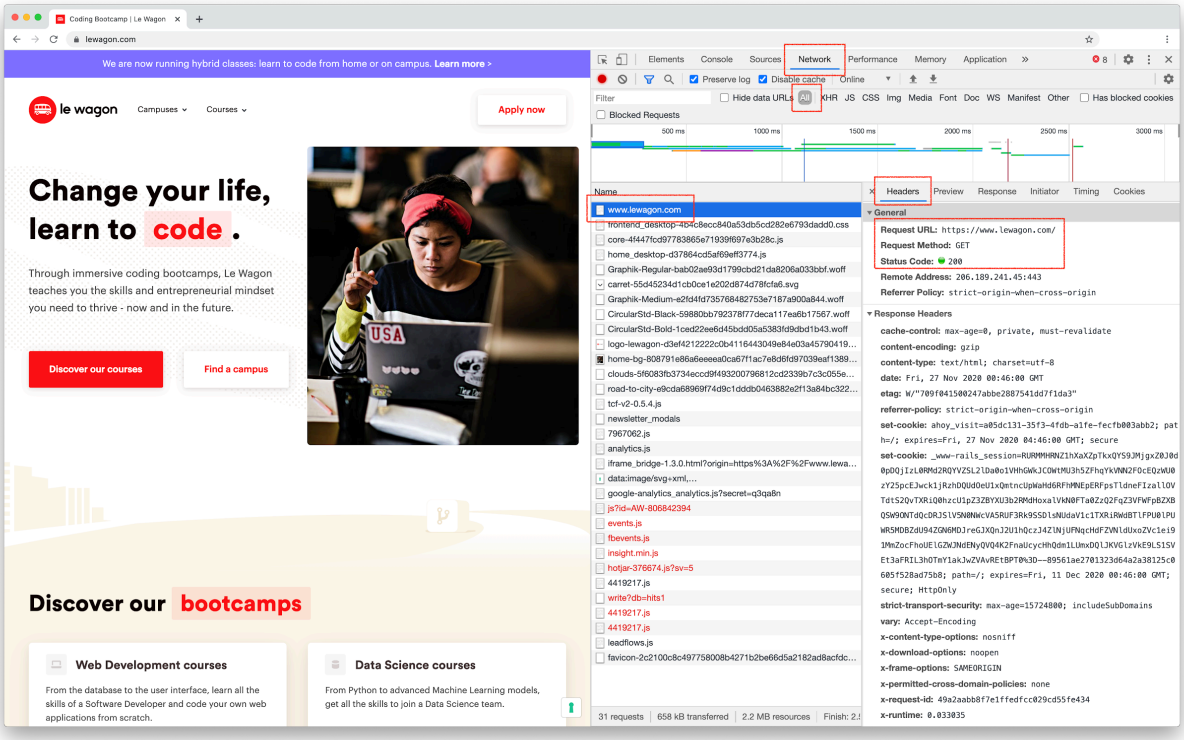
- json or html, css, js, images, etc

headers

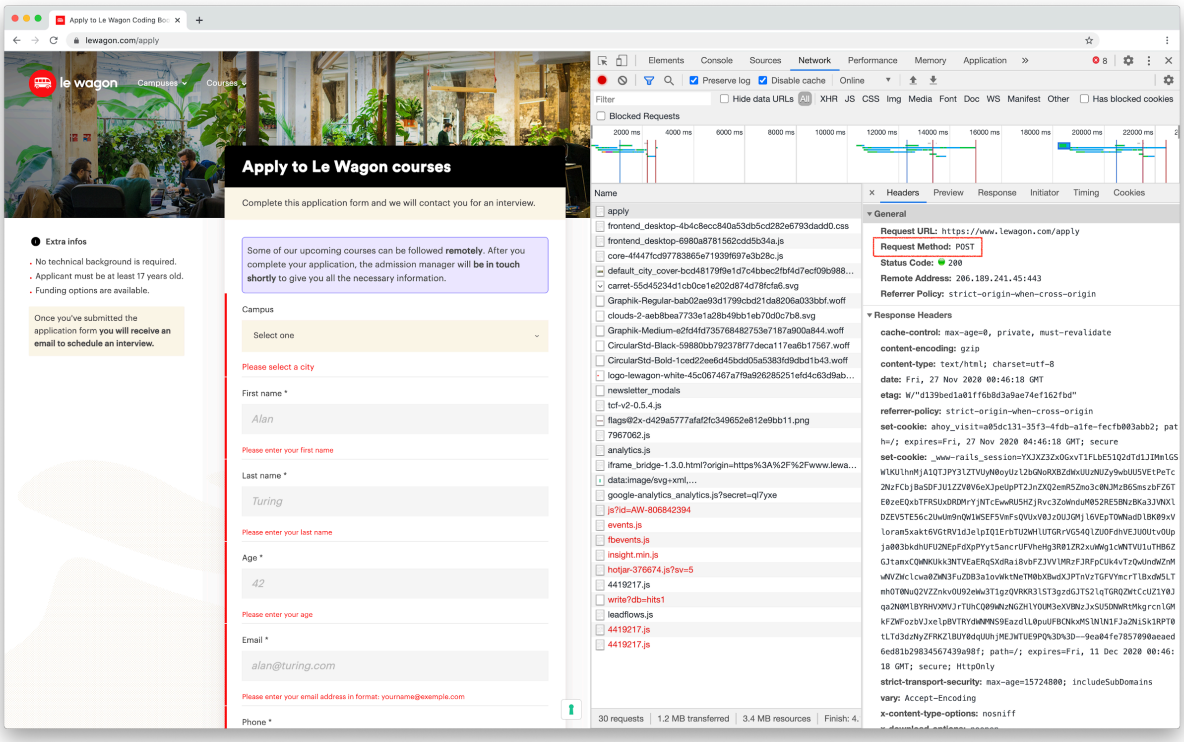
HTTP



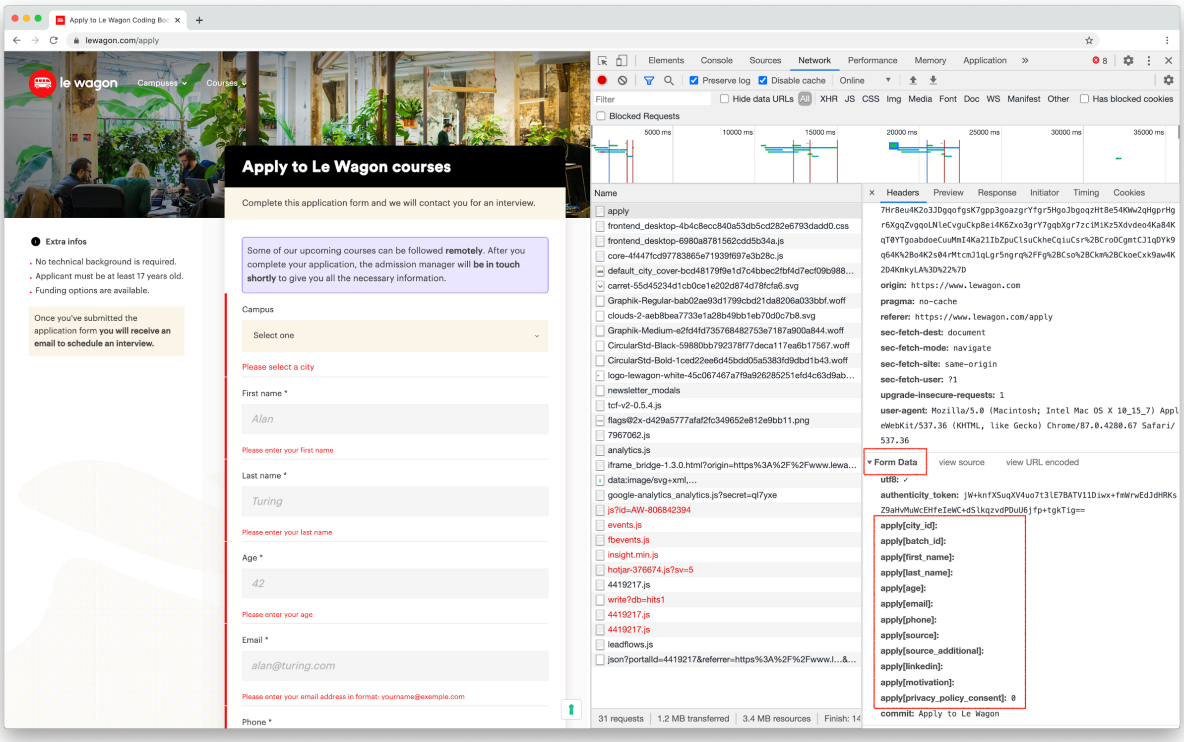
HTTP Headers


Coding Bootcamp | Le Wagon
lewagon.com
We are now running hybrid classes: learn to code from home or on campus. Learn more >
le wagon
Campuses
Courses
Apply now
Change your life, learn to code.
Through immersive coding bootcamps, Le Wagon teaches you the skills and entrepreneurial mindset you need to thrive - now and in the future.
Discover our courses
Find a campus
Discover our bootcamps
Web Development courses
From the database to the user interface, learn all the skills of a Software Developer and code your own web applications from scratch.
Data Science courses
From Python to advanced Machine Learning models, get all the skills to join a Data Science team.
Elements
Console
Sources
Network
Performance
Memory
Application
Preserve log
Disable cache
Online
Filter
Hide data URLs
All
XHR
JS
CSS
Img
Media
Font
Doc
WS
Manifest
Other
Has blocked cookies
Blocked Requests
Name
Headers
Preview
Response
Initiator
Timing
Cookies
General
Request URL: https://www.lewagon.com/
Request Method: GET
Status Code: 200
Remote Address: 206.189.241.45:443
Referrer Policy: strict-origin-when-cross-origin
Response Headers
cache-control: max-age=0, private, must-revalidate
content-encoding: gzip
content-type: text/html; charset=utf-8
date: Fri, 27 Nov 2020 00:46:00 GMT
etag: W/"709f04150247abbe2887541dd7f1da3"
referrer-policy: strict-origin-when-cross-origin
x-content-type-options: nosniff
x-download-options: noopen
x-frame-options: SAMEORIGIN
x-permitted-cross-domain-policies: none
x-request-id: 49a2aabb8f7e1ffedfcc029cd55fe434
x-runtime: 0.033035
31 requests
658 kB transferred
2.2 MB resources

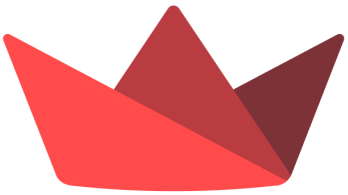
HTTP POST


lewagon.com/apply
le wagon
Campuses
Courses
Apply to Le Wagon courses
Complete this application form and we will contact you for an interview.
Extra infos
No technical background is required.
Applicant must be at least 17 years old.
Funding options are available.
Once you've submitted the application form you will receive an email to schedule an interview.
Some of our upcoming courses can be followed remotely. After you complete your application, the admission manager will be in touch shortly to give you all the necessary information.
Campus
Select one
Please select a city
First name *
Alan
Please enter your first name
Last name *
Turing
Please enter your last name
Age *
42
Please enter your age
Email *
alan@turing.com
Please enter your email address in format: yourname@example.com
Phone *
Request URL: https://www.lewagon.com/apply
Request Method: POST
Status Code: 200
Remote Address: 206.189.241.45:443
Referrer Policy: strict-origin-when-cross-origin
30 requests
1.2 MB transferred
3.4 MB resources

HTTP Body



Streamlit



Streamlit

Streamlit is an application framework geared towards Data Science

Its goal is to enable data scientists to release applications without requiring the assistance of a development team

`Streamlit` allows us to build an app that makes predictions using a trained model, with very few lines of code to handle the user interface and controls, and almost no design efforts

Which Framework for your App?

	Jupyter	Streamlit	Flask	Django
environment	❌ notebook	✅ web site	✅ web site	✅ web site
user input	❌ dataframe	✅ controls	✅ controls	✅ controls
design	❌ notebook	✅ streamlit	✅ HTML/CSS	✅ HTML/CSS
complexity	✅ n/a	✅ low	❌ medium	❌ high

How Does it Work?

Streamlit interprets the code of the app from top to bottom, with every element in the main script file being displayed one after the other.

The elements may be simple variables (strings containing text or markdown), objects (DataFrames are represented as tables), more complex user controls (actions or inputs), charts, maps, or third-party graphs (Matplotlib, Plotly, etc.)

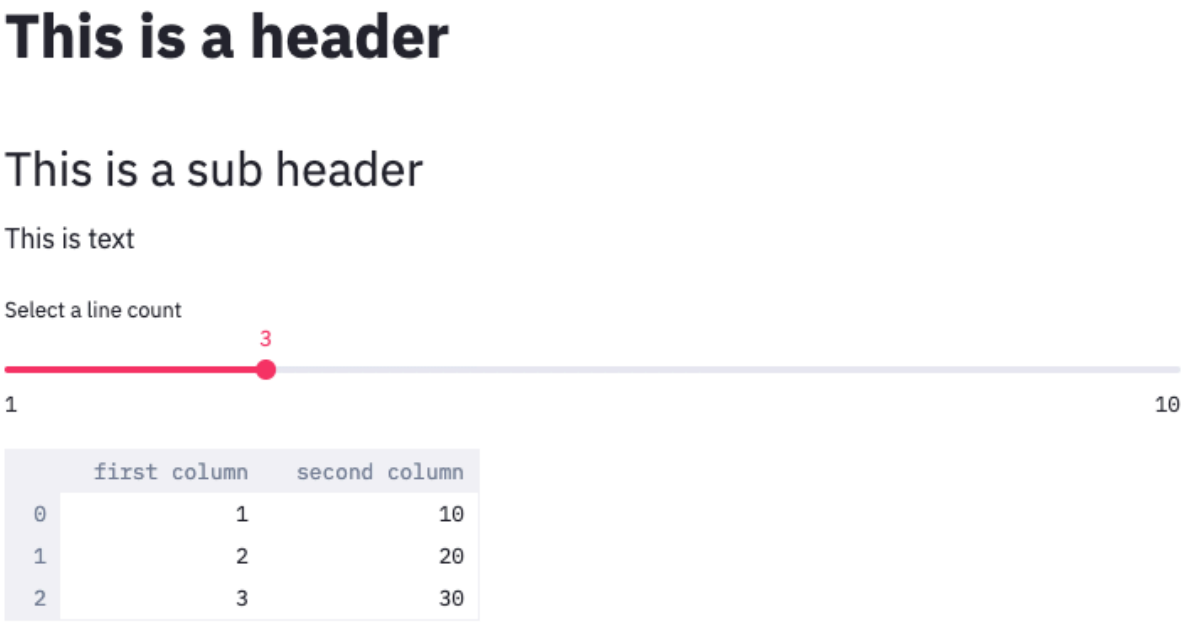
Streamlit draws the output live in the browser, as a notebook would!

A Simple App

```
import streamlit as st

import numpy as np
import pandas as pd
```

```
st.markdown("""# This is a header
## This is a sub header
This is text""")

df = pd.DataFrame({
    'first column': list(range(1, 11)),
    'second column': np.arange(10, 101, 10)
})

# this slider allows the user to select a number of lines
# to display in the dataframe
# the selected value is returned by st.slider
line_count = st.slider('Select a line count', 1, 10, 3)

# and used to select the displayed lines
head_df = df.head(line_count)

head_df
```

Result



	first column	second column
0	1	10
1	2	20
2	3	30

How to Run the App

streamlit run app.py

When this command is run, **Streamlit** starts a web server and opens a new tab in the web browser, showing the app. This allows you to assess the content of the page while you code.

Streamlit automatically reloads the page as soon as the code of the main script file changes and is saved, which is very handy 😉

Streamlit Elements Demo

The Streamlit quick reference, developed by Le Wagon using **Streamlit**, provides a sample demonstration of the main elements usable in a **Streamlit** app.

Have a look at the possibilities provided by **Streamlit** 🎉

For more information, refer to the official Streamlit doc

Going Further

⚠️ **Streamlit** reinterprets the full content of the main script file **every time the user interacts with any control on the page**. This may render the page unresponsive, but can be prevented (read the **Cache** section in the Streamlit quick reference)

Deploy Streamlit on Streamlit Cloud

What is Streamlit Cloud under the hood?

> Under the hood, Streamlit Cloud handles all of the containerization, authentication, scaling, security and everything else, so that all you need to worry about is creating the app.

> Maintaining Streamlit apps is easy, as containers automatically get the latest security patches, and are actively monitored for container health.

All you need to do is link a GitHub repository to Streamlit Cloud!

What do you need in your project to deploy it on Streamlit Cloud?

💻 Let's code a simple app together!
tree

```
.
├── README.md
├── raw_data # WARNING: recommended only for very simple use cases!
│   ├── some_data.csv
│   └── img.png
├── app
│   ├── app.py
│   └── pages
│       ├── page_01.py
│       └── page_02.py
└── requirements.txt
```

⚠️ Only store data directly on your app if you **absolutely have to**, and **exercise caution** when doing so (sensitive data, heavy files, etc.).

Otherwise, the best practice is to store the data you need in a secure location (e.g. Google Storage): your app will be able to access it via `SECRETS`. We'll cover that later on.

app.py

```
import streamlit as st

st.write('Welcome to my app')
```

pages/page_02.py

Streamlit automatically handles pages 🥳
Place your `.py` files in a `pages` folder and that's it!

```
import streamlit as st

st.write('This is page 2')
```

requirements.txt

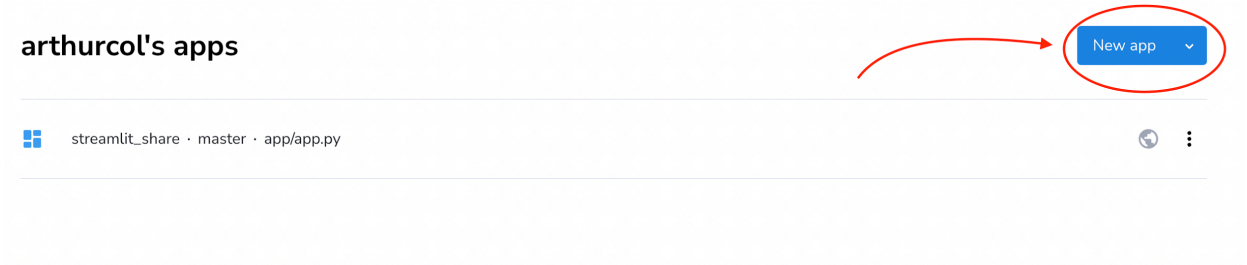
This file is needed for Streamlit to handle the package dependencies of your application. More details in the Streamlit documentation.

Example:
streamlit
pandas
requests

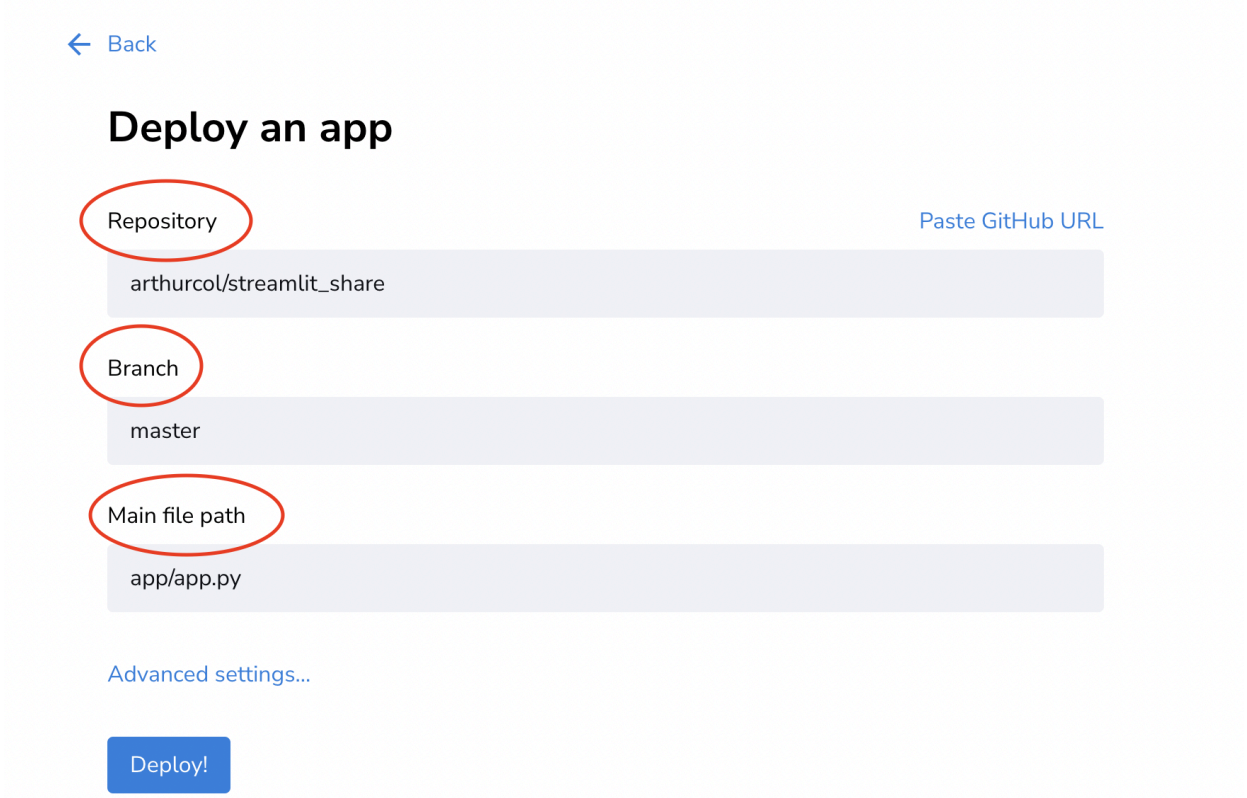
💡 If you need to install a package not available on PyPI, you can also use:
git+https://github.com/USERNAME/YOUR_PACKAGE.git

Deploying the App on Streamlit Cloud

Easy as pie!



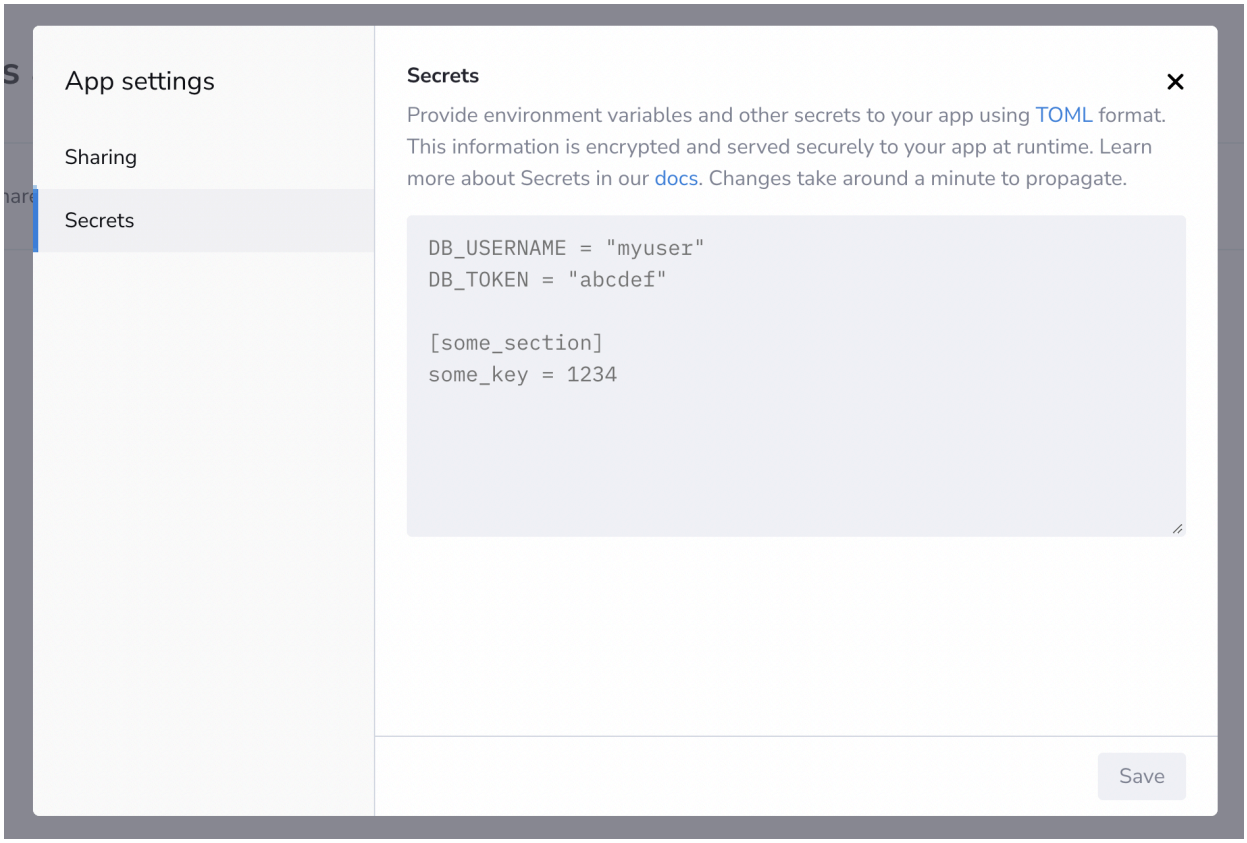
Link Streamlit Cloud to your GitHub account, select your repo and you are good to go 🚀



Make use of `SECRETS` 🤫

🔑 If you need to connect to a non-public database, you can use the `SECRETS` settings of your app to safely store your credentials.

More details for the specific use case of Google Cloud Storage are in the documentation.



💻 How to access your `SECRETS` from your app

Imagine that your app sends requests to an API for which you need an authentication key.

First, you need to add your credentials in the `SECRETS` text field, following the TOML format:

```
spell="alohomora"

[some_magic_api] # you can create sections
key="secretkey"
```

To access your secrets from your `app.py`:

```
import streamlit as st

spell = st.secrets['spell']
key = st.secrets.some_magic_api.key
```

How to test it locally? 🤔

You can add a `secrets.toml` file in the `.streamlit` folder at the root of your project.

🚨 **Do not forget to add this file to your `.gitignore`!**

Creating a Project Repo

Project Repository

⚠️ Do **not** create a `git` repository inside of another `git` repository (for example `data-challenges`)

```
# Go to the location of the data-challenges repo
cd ~/code/USER_NAME/

# Create a separate directory for your project
mkdir appname

# Sit inside of your app
cd appname

# Initialize a new git repo
git init

# Create GitHub repo
gh repo create

# Go to GitHub repo and change the visibility to private (in the settings of the repo)
gh browse
```

You now have a local `git` repo linked to a GitHub repo 🎉

Glossary

Your repo will have files such as:

.git Directory

Main directory used by `.git` to track changes in a folder

`.git` is what makes a folder a repository, so if you erase this directory, you will lose your entire commit history, unless it has been pushed to a remote repository (origin, for example)

.gitignore

Specific to `git`, this file lists the files to exclude from version control

For example, you may want to ignore the `__pycache__` folders, which can be done from `.gitignore`

README.md

Written in Markdown, this **optional** file is normally used to provide information about the app

This will be displayed on the main page of your repository on GitHub

requirements.txt

Contains the Python **packages** required to run Streamlit and your app

If you set up a virtual env for your app, the packages can be obtained via `pip freeze`

app.py

The main script of the app.

The name can be anything, this is the file you indicate as being the main file when deploying on Streamlit Cloud.

__pycache__

Cache directory generated by Python when it runs the code

Erasing this directory has no impact, it may be added to `.gitignore`

.ipynb_checkpoints

Temporary **Jupyter notebook** directory generated whenever a notebook is created

It stores automatic notebook backups, erasing this directory has no impact, so it may be added to `.gitignore`

Good Practice

⚠️ Depending on your model and its dependencies, you may want to create separate packages for your model and for the Streamlit app. This is the case most of the time.

In that case, you will have a "backend" package that you will most likely deploy as a containerized app (e.g. using Google Run) and a very light "frontend" package that you can deploy on Streamlit Cloud. The last will request the first as an API to make predictions.

💡 Beware of your environments
For example, you may have used packages to clean the data or train the model that are **no longer required** for prediction

Surely, Streamlit is not a requirement of your model and does not need to be deployed for training

➡️ Virtual envs can be of great help here

Your turn! 🚀