# Installation / Setting up the project

## --- Creating and starting a virtual environment, OPTIONAL but recommended ---

Virtual environments make local dependency management across different projects easy. By default, Python dependencies are installed on the entire system. When you activate a virtual environment, dependencies are installed into the virtual environment’s directory.

**Create venv**

python -m venv .venv

**Activate venv**

source .venv/bin/activate

**Deactivate venv**

deactivate

## --- Getting the project to run On linux/WSL (Windows subsystem for linux) —

#### Issues/errors faced

This section contains some errors that we faced when we tried to install dependencies with either **pip install .[full]** and **pip install .[dev]** or when we tried to run the program with the given commands in the README.md.

**The version specified is not a valid version according to PEP 440. This may not work as expected with newer versions of setuptools, pip, and PyPI.**

**Change the "version" in setup.py from version=git\_version, to version="1.2.3"**

* This is just a placeholder, as the git\_version doesn’t work. Not done in an actual project

.

**If you have some error with wheel**

pip install wheel

**FileNotFoundError: [Errno 2] No such file or directory: 'curl-config'**

sudo apt install libcurl4-openssl-dev libssl-dev

**fatal error: Python.h: No such file or directory**

sudo apt-get install python3.11-dev

**When running the command "PYTHONPATH=.. uvicorn app:app", you get the error "from xgboost import XGBClassifier ModuleNotFoundError: No module named 'xgboost'"**

NOTE: The following imports can maybe be ignored by running "pip install .[dev]".

pip install xgboost pandas Pillow dhash transformers opencv-python pydantic-settings

**None of PyTorch, TensorFlow >= 2.0, or Flax have been found. Models won't be available and only tokenizers, configuration and file/data utilities can be used.**

* pip install tensorflow # NOTE: This might not be the correct dependency, it gets rid of the warning so we left this here.

**Raise PydanticImportError(pydantic.errors.PydanticImportError: `BaseSettings` has been moved to the `pydantic-settings` package. See https://docs.pydantic.dev/2.4/migration/#basesettings-has-moved-to-pydantic-settings for more details.**

In the file config.py, change the import

"from pydantic import BaseSettings" -> "from pydantic.v1 import BaseSettings"

## Running the scraper

With curl you get “reasonable” amount of output to maybe manually examine:

In the repository root:

PYTHONPATH=. python web\_classifier/cli/scraper.py --scraper curl -i ./web\_classifier/tests/fixtures/test\_urls.txt -o /tmp/scraper\_output.jsonl

With playwright you get lots more output:

PYTHONPATH=. python web\_classifier/cli/scraper.py --scraper playwright -i ./w

eb\_classifier/tests/fixtures/test\_urls.txt -o /tmp/scraper\_output.jsonl

Results will be saved to a file in /tmp/ (the parameter after -o), this can be changed to any file and location.

## Running the extractor

The extractor is also run from the repository root:

PYTHONPATH=. python web\_classifier/cli/extractor.py -i /tmp/scraper\_output.jsonl -o /tmp/extracter\_output.jsonl

This expects the scraped data to be in the file pointed by the argument after -i (here the same as above). The extractor output is again saved by default to a file in /tmp, but this can also be changed with the parameter after -o

## Running the tests

cd web\_classifier

PYTHONPATH=.. uvicorn app:app

We couldn’t get this to work. We had a hunch that the problem is that the .env file needs to contain some information on the models that we don’t have. This causes a problem in the PhishingClassifier class, which is expecting to get names for the classification models.

## Basic functionality of the program

**app.py** starts a uvicorn server (basic HTTP server), that listens to addresses / and /classify/url. / is the homepage, /classify/url is an API that receives an url as the request payload, which is then scraped. The features of the scraped data are extracted and the extracted features are inputted into the ML model that classifies the website.

When the different components are run from the command line, the intermediate result is saved onto the filesystem as a .jsonl file. JSONL (JSON lines) is a data format, where each line in the file corresponds to a single JSON object.

### Scrapers

The scrapers/**base.py** directory contains an abstract Scraper class, that is inherited by BrowserScraper. The CurlScraper inherits from the Scraper class. The PlaywrightScraper and the SeleniumScraper inherit from the BrowserScraper class. The Scraper class defines that the scrapers should have features such as “scrape” (scrapes the website) and “addargs” (adds arguments to the scraper).

The code for a CurlScraper is in the file scrapers/**curl\_scraper.py**. It has features to add arguments to the curl requests as well as creating and sending the request and returning the output in the scrape method.

The code for a PlayWright scraper is in the file scrapers/**playwright\_scraper.py**. The scraper has options for the scraper, such as proxy, using GPU for scraping, using a mobile scraper, adding HTTP headers or a user agent and running in headless mode (Meaning that a new window isn’t opened when the browser is started). It implements the Scraper class.

There also exists a Selenium scraper but it seems that it is not used.

### CLI

Scrapers are started and run from the ./web\_classifier/cli/**scraper.py** that takes the arguments passed with the command and after parsing, passes them to either the curl or the playwright scraper. scraper.py is also responsible for handling running the scraping in parallel if so specified with the arguments.

(To adjust the number of workers/threads, use --num\_conn for curl and --num\_pages for playwright. For additional details on the parameters to pass for each scraper, see CurlScraper and PlaywrightScraper) from the README.md.

### Extractor

The extractor directory contains extractors. The two extractors aren’t related to each other in any way (no common class etc).

The ImageExtractor in **image\_extractor.py** is used for extracting the image data. It isn’t used anywhere

**html\_extractor.py** contains two important classes. The HTMLWrapper is an internal class that contains a HTML document and has methods for parsing and returning different parts of it, such as the JavaScript, the urls, the titles, the metadata, the css, the images, etc. The HTMLExtractor is a public class that uses the HTMLWrapper and outputs data in an uniform manner as HTMLExtractorOutput objects.

### Classifier

The classifications/**classifier.py** file contains an abstract Classifier class, that is inherited by the different classifiers. The classifiers that are implemented are the adult classifier (**adult.py**), phishing classifier (**phishing.py**), and text classifier (**text.py**). The Classifier class implements the predict method, which is used for predicting the class of a website.

### Utils

There are many useful tools here. It would be a good idea to read through the function names so that we don’t reinvent the wheel.

### More examples of running the code

Run playwright scraper with two tabs and in no-headless mode with 20sec wait before rendering the pages:

PYTHONPATH=. python web\_classifier/cli/scraper.py --scraper playwright --no-headless --num\_pages 20000 --implicit\_wait 20000 -i ./tests/fixtures/test\_urls.txt -o /tmp/scraper\_output.jsonl