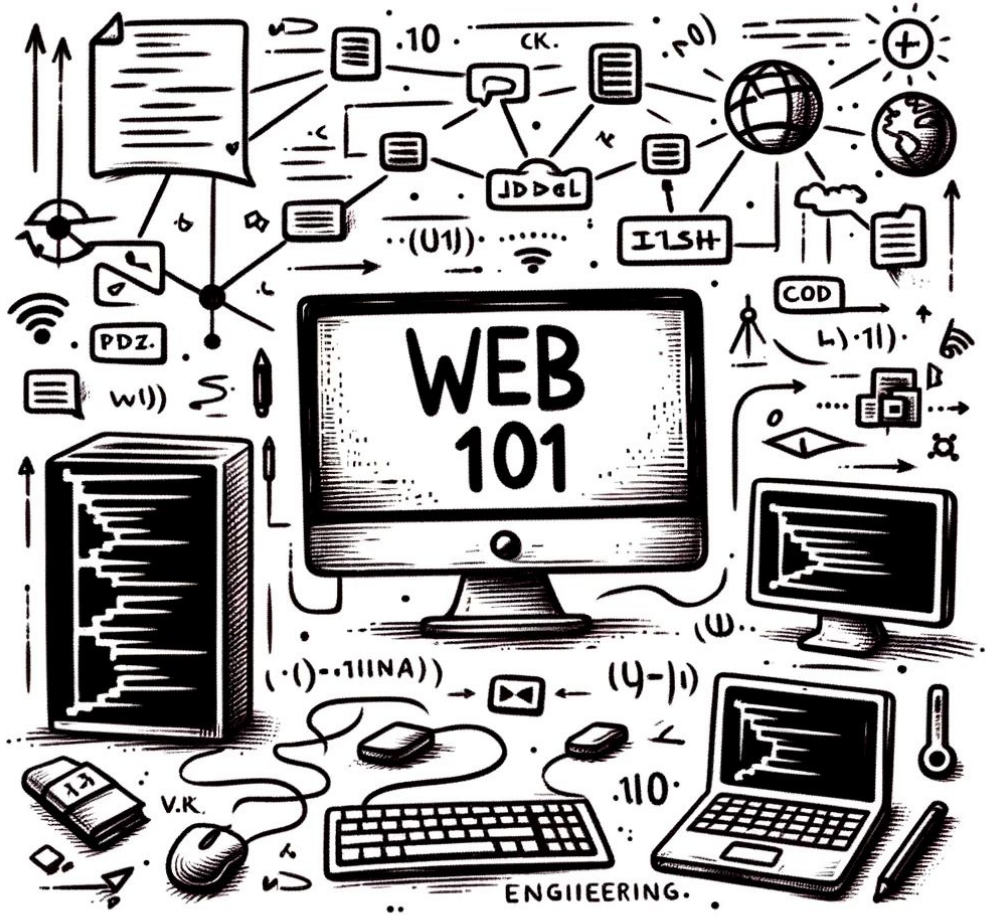

WEB
101
ENGIIEERING.

whoami

Adrien Brunet
Co Founder & CTO @ May
may.app

Promo sortante 2013
brunet.adrien@gmail.com

https://github.com/adrienbrunet
https://github.com/adrien-may

https://www.linkedin.com/in/adrienbrunet

Et vous ?

Tour de table express

Prénom

Vos ambitions après l'école ou le stage ?

Ce que vous aimez dans le code / dev ?

Ce que vous espérez apprendre avec ce cours ?

Prérequis

Environnement python :

python 3.10+
pip & virtualenv

Environnement node :

node 22

Contrôle de version des sources :

git + compte github https://github.com

Autres :

Base de données dbeaver : https://dbeaver.io

Postman : https://www.postman.com/downloads/
ou Insomnia : https://insomnia.rest/

Les questions auxquelles vous saurez répondre…

Comment fonctionne un “site”, une “app”

Comment fonctionne une requête ?

Où sont stockées les données ?

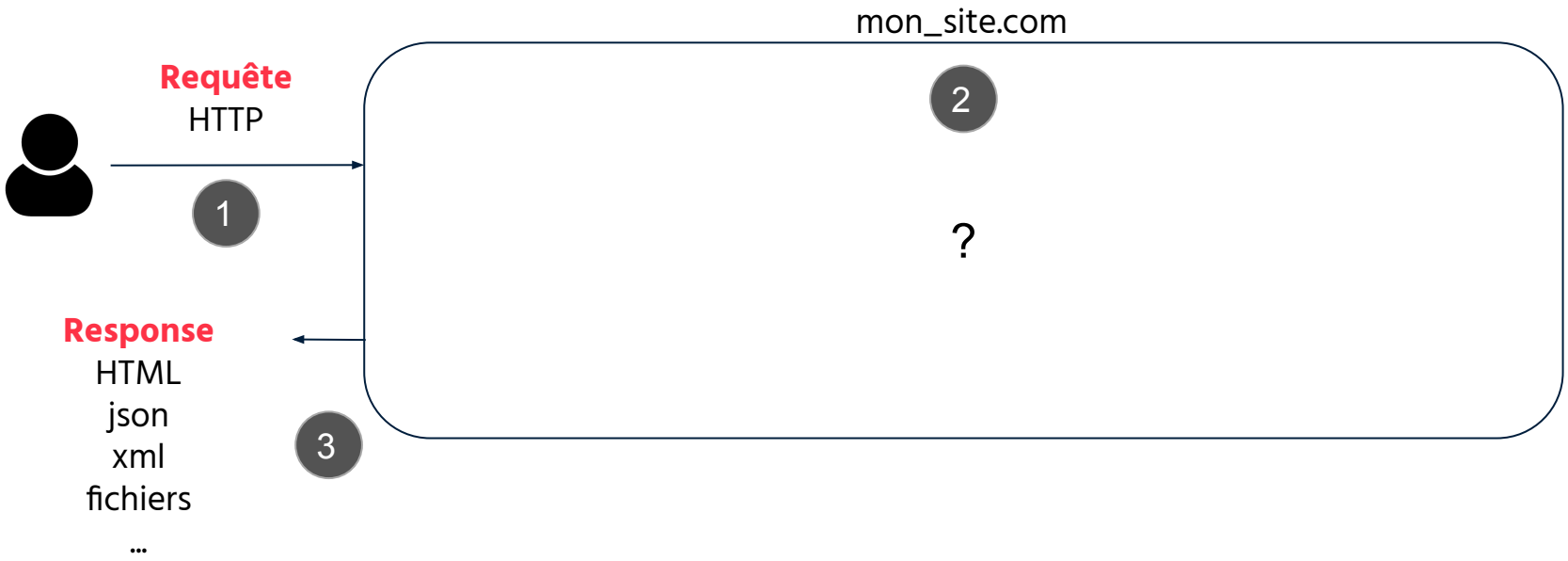
Où s’opère le calcul de rendu de la page ?

Définition

> Un **site web** est un ensemble de pages web et de ressources reliées par des hyperliens, défini et accessible par une adresse web. Un site est développé à l'aide de langages de programmation web, puis hébergé sur un serveur web accessible via le réseau mondial Internet, un intranet local, ou n'importe quel autre réseau, tel que le réseau Tor.
>
> L'ensemble des sites web publics constituent le World Wide Web.

Wikipedia

A web request / http request - bottom up



In case you need it

HTML resources

HTML tags: https://www.w3schools.com/tags/

CSS rules : https://developer.mozilla.org/en-US/docs/Learn/CSS/First_steps/How_CSS_is_structured

Autres ressources :

https://developer.mozilla.org/fr/
https://json.org/example.html
https://www.javascript.com/

HTML

page1.html

```
<p>Ma première page</p>
```

page2.html

```
<styles>
 .title {
   font-weight: 500;
   color: "tomato";
 }
</styles>
<p class='title'>Ma première page avec du style</p>
```

HTML + CSS

my_style.css

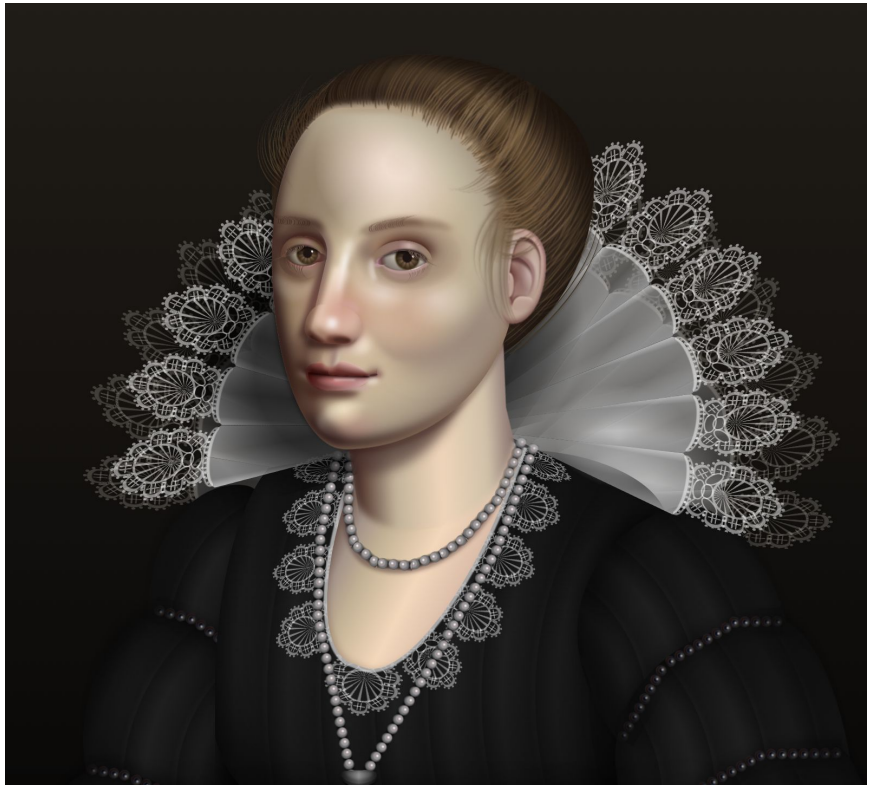
```
h1 {
 font-weight: 500;
 font-size: 48px;
 color: "tomato";
}
```

index.html

```
<html>
 <head>
   <link rel="stylesheet" type="text/css"
href="./my_style.css">
 </head>
 <body>
   <h1>Title</h1>
   <p>My first paragraph</p>
 </body>
</html>
```

CSS à l'extrême

https://diana-adrianne.com/purecss-lace/



Other formats

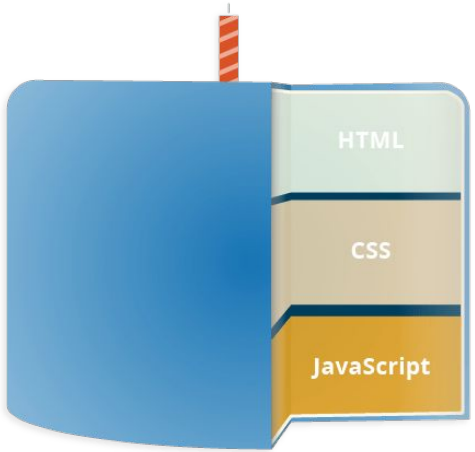
JSON

```
{
 "menu": {
   "id": "file",
   "value": "File",
   "items": [
     {
       "value": "New",
       "onclick": "CreateNewDoc()"
     },
     {
       "value": "Open",
       "onclick": "OpenDoc()"
     }
   ]
 }
}
```

What about js ?

Late bird dans l'écosystème, on ferait difficilement sans aujourd'hui.

> JavaScript is a scripting or programming language that allows you to implement complex features on web pages — every time a web page does more than just sit there and display static information for you to look at — displaying timely content updates, interactive maps, animated 2D/3D graphics, scrolling video jukeboxes, etc. — you can bet that JavaScript is probably involved. It is the third layer of the layer cake of standard web technologies, two of which (HTML and CSS) we have covered in much more detail in other parts of the Learning Area.



First tool - chrome web dev tool

https://developer.chrome.com/docs/devtools/overview/

Pick any website

- inspect its element
- find nice css snippets
- follow javascript code ?

Console

- Interact with the website ?
- perform computations ?
- see what's available

Network

- look at what is being downloaded ?
- see dependencies

Intégration… à vous de frimer (20 min max)

Métier: **web integrator**

Explorons canvas

C’est parti pour un space invaders !

Premier bilan

On a survolé une partie du métier lié au “front-end”.

Comment ces réalisations là “arrivent” jusqu’au navigateur ?

Comment gérer la logique métier ?

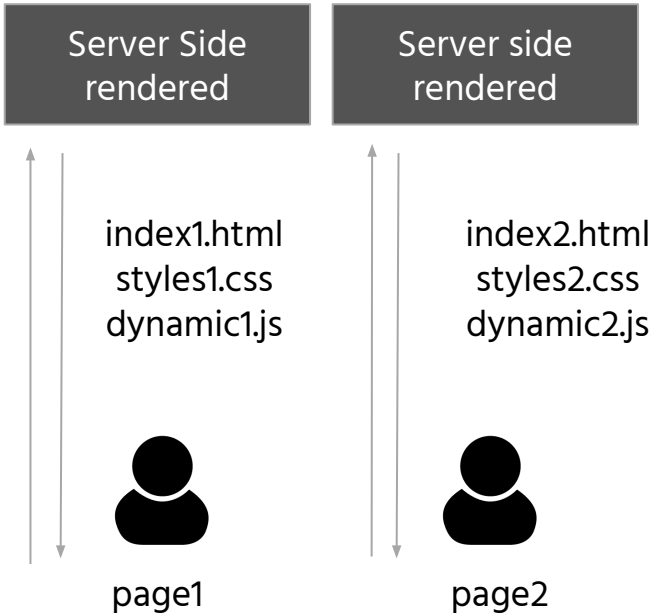
Comment sauvegarder des données, protéger le site avec authentification ?

Comment échanger des données ? Ré-utiliser des fonctionnalités ?

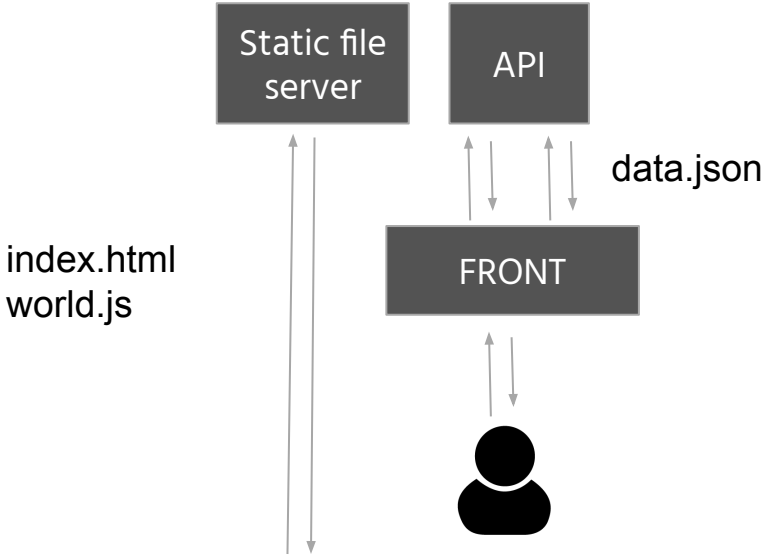
→ **Let’s deep dive to the backend side**

Précisions

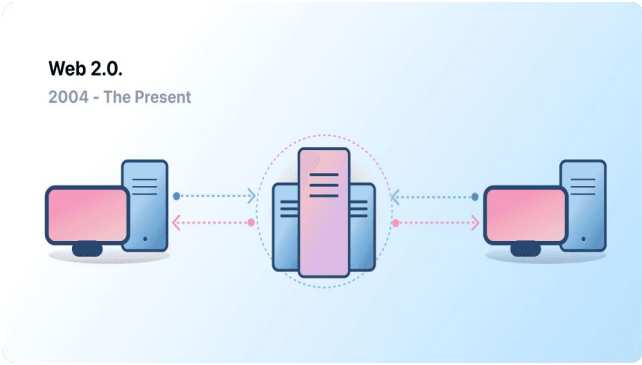
Day 1

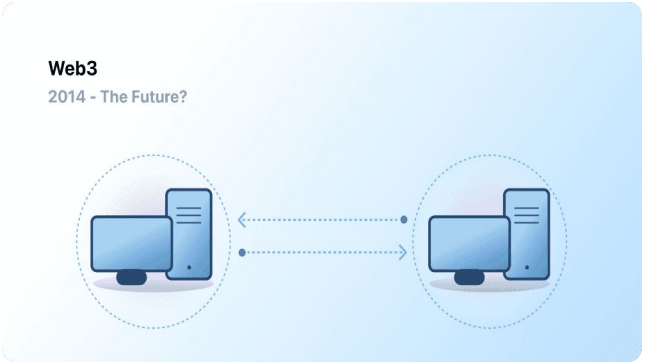


Day 2



Et le web “3” ?





https://ethereum.org/en/web3/

Back vs Front - qui finira par gagner ?

https://www.youtube.com/live/1g5ruM-16_Y?si=xM8LgdjN6Phuz1MK&t=7607

Programme

JOUR 1

Matin : Découverte de Flask

AM : TP - Création d'un site ex nihilo et/ou API

JOUR 2

Matin : Découpage front/back. Site React

AM : Suite React, React-Native pour les intrépides

Fil rouge

Avoir une compréhension des différentes problématiques / métiers impliqués

Comprendre dans sa globalité comment s'articulent les différents domaines impliqués.

En bref

Flask : micro framework web en python

Dépendances :

Werkzeug

Routing, Debugging et Web Server Gateway Interface (WSGI)

Jinja2

Moteur de template

Pas de base de données, authentification, forms etc...

Flask

Install party

virtualenvs

DANS VOTRE DOSSIER DE TRAVAIL, CRÉEZ UN DOSSIER ET ALLEZ DEDANS

```
mkdir my_project
cd my_project
python3 -m venv /path/to/virtual/environment
```

```
ex: python3 -m venv venv
```

```
source /path/to/virtual/environment/bin/activate
```

```
ex: (unix/macOS) source venv/bin/activate
(windows) venv\scripts\activate.bat
```

Flask

pip install --upgrade pip

```
pip install flask
```

```
pip freeze > requirements.txt
```

Ca marche ?

```
virtualenv --version
```

```
python
```

```
>>> import flask
```

Ma première application flask

app.py

```
from flask import Flask

app = Flask(__name__)
```

```
flask shell ?
```

Créez le dossier “tests”
tests/test_app.py (+ don’t forget __init__.py)

```
from flask import Flask

from app import app


def test_app():
    assert app is not None
    assert isinstance(app, Flask)
```

```
pytest ??
```

Install party, le retour

Métier: **Q&A - Test engineer**

https://github.com/adrien-may/ecm2024

https://github.com/adrien-may/ecm2024/tree/first_app

pytest

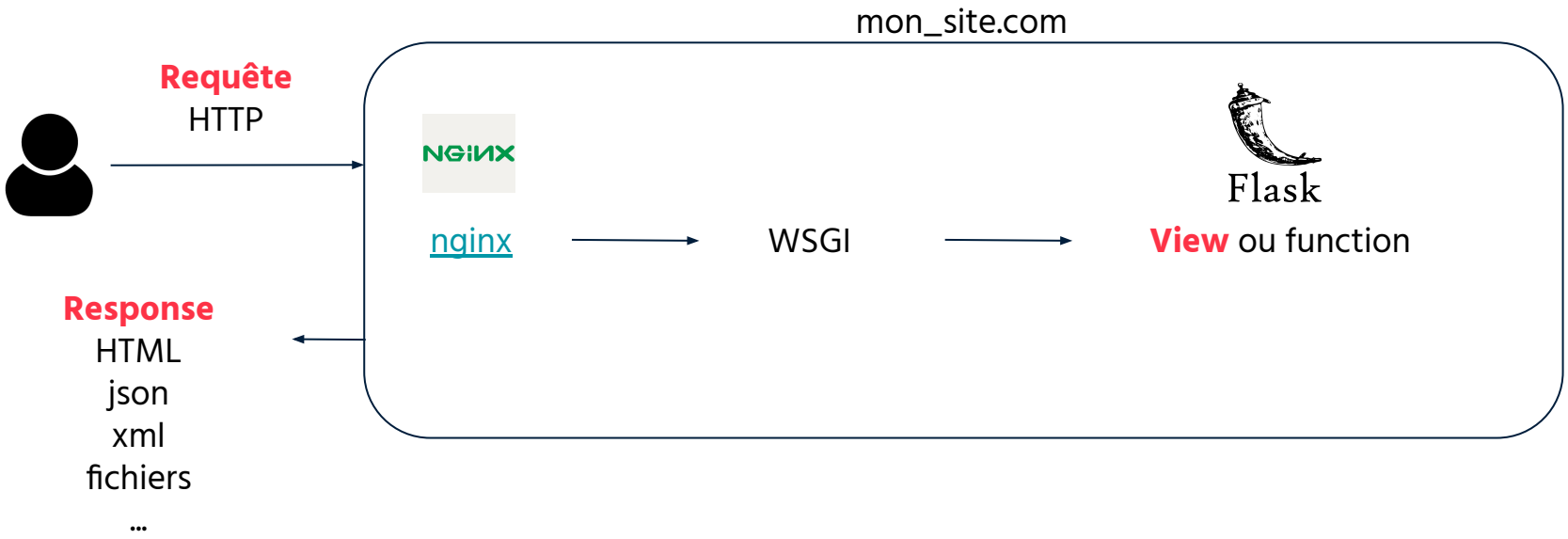
```
> pip install -r requirements.dev.txt

# Install de pytest et plusieurs
dépendances…
```

Ca marche ?

```
> pytest
```

branch: first_app

Requêtes web



Décorateur - kesako

Une fonction qui retourne une fonction

```
def my_decorator(func):
    def wrapper():
        print("Avant, c'est avant.")
        func()
        print("Après, c'est après.")
    return wrapper

@my_decorator
def say_something():
    print("Hello hello.")
```

```
> say_something()
# Avant c'est avant.
# Hello hello
# Après, c'est après.
```

Nos premières views + route

branch: first_view

app.py

```python
from flask import Flask

app = Flask(__name__)

@app.route('/')
def index():
    return '<h1>ECM Bonjour</h1>'


@app.route('/user/<name>')
def user(name):
    return f'<h1>Hello, {name}</h1>'
```

tests/test_app.py

```python
def test_index_route():
    client = app.test_client()
    response = client.get("/")
    assert response.status_code == 200
    assert "ECM" in response.data.decode("utf-8")


def test_index_user():
    client = app.test_client()
    response = client.get("/user/adrien")
    assert response.status_code == 200
    result = response.data.decode("utf-8")
    assert "Hello, adrien" in result
```

Local dev server

```
> flask run
```

```
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: off
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

Ouvrez votre navigateur aux URLs suivantes :

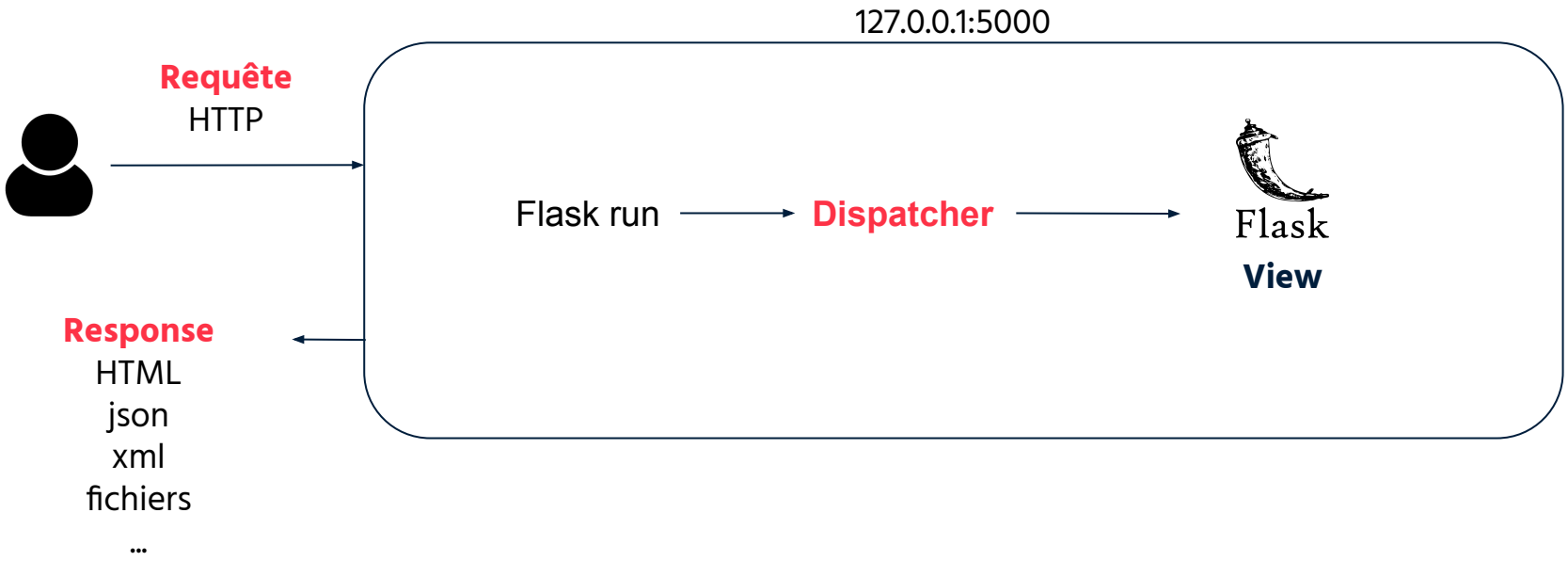
http://127.0.0.1:5000/

http://127.0.0.1:5000/user/friend

Dev mode :

```
export FLASK_APP=myapp
export FLASK_ENV=development
flask run
```

Et le joli graphe d'avant ?



Et le joli graphe d'avant ?

Dispatcher

```
$ flask shell

> from app import app

> app.url_map

Map([<Rule '/static/<filename>' (GET, OPTIONS, HEAD) -> static>,

 <Rule '/' (GET, OPTIONS, HEAD) -> index>,

 <Rule '/user/<name>' (GET, OPTIONS, HEAD) -> user>])
```

Et le cycle request/response ?

Requests

```
from flask import Flask, request

app = Flask(__name__)

@app.route('/')
def index():
   user_agent = request.headers.get('User-Agent')
   return f"<p> Your browser is {user_agent}</p>"
```

Application

```
from app import app
from flask import current_app

current_app.name
# ERROR

app_ctx = app.app_context()
app_ctx.push()
current_app.name
# 'app'
app_ctx.pop()
```

Contextes

Application context

current_app - The application instance for the active application.

g - An object that the application can use for temporary storage during the handling of a request. This variable is reset with each request.

Request context

request - The request object, which encapsulates the contents of a HTTP request sent by the client.

session - The user session, a dictionary that the application can use to store values that are “remembered” between requests.

Request hooks

before_first_request : Register a function to run before the first request is handled.

before_request : Register a function to run before each request.

after_request : Register a function to run after each request, if no unhandled exceptions occurred.

teardown_request : Register a function to run after each request, even if unhandled exceptions occurred

G object

Exemple d'utilisation :

Ouverture d'une connection en base de donnée

Fermeture à la fin de la requête

```
from flask import g

def get_db():
  if 'db' not in g:
      g.db = connect_to_database()

  return g.db

@app.teardown_appcontext
def teardown_db():
  db = g.pop('db', None)

  if db is not None:
      db.close()
```

Request, ok. Response ?

Notre code pour l'instant :

```
return '<h1>My text</h1>'
```

Le protocole HTTP, attend :

Un status code (200, 404, 500, etc...)
Contenu (html? Plain text? json?)
Headers

Flask ajoute pour nous ce qu'il manque.

Exemples :

```
return 'bli bla blo', 200
```

—

```
return 'bli', 200, headers_dict
```

—

```
from flask import make_response


@app.route('/')
def index():
    response = make_response('<h1>Have a
cookie!</h1>')
    response.set_cookie('answer', '42')
    return response
```

Introduction des templates

Template en flask : Jinja

Il existe de très nombreux moteurs de template disponibles, *jinja* reste très populaire en flask et en django.

my_template.html

```
<main>
 <p> Hello {{ user }} </p>
</main>
```

Les variables doivent être définies dans un contexte pour être interprétées par le gabarit (template).

Templates : logique et filtres

Quelques filtres

`safe, capitalize, lower, upper, title, trim, striptags`

Logique

```
{% if %}{% else %}
{% for %}{% endfor %}
```

Allez plus loin

Macro - flask-bootstrap (?)

my_template.html

```
<main>
 <p>Hello {{ user|capitalize }}</p>
 <section>
   <h1>Adults</h1>
   {% for member in family %}
     {% if member.age >= 18 %}
     <p>{{ member.name|upper }}</p>
     {% endif %}
   {% endfor %}
 </section>
</main>
```

Utilisation d'un template

app.py

```python
from flask import Flask, render_template

app = Flask(__name__)

@app.route('/user/<name>')
def user(name):
    return render_template('user.html', name=name)
```

templates/user.html

```
<html>

<head>
   <title>FLASK TEST</title>
</head>

<body>
   <h1>Hello {{name}}</h1>
</body>

</html>
```

Utilisation d'un template (et les tests !)

test.py

```
def test_user_template():
    client = app.test_client()
    response = client.get("/user/adrien")
    template = app.jinja_env.get_template('user.html')
    assert template.render(name="adrien") == response.get_data(as_text=True)
```

Que pensez-vous de ces tests ?

Petits ajouts (½)

Coverage & sugar

Jetez un oeil à requirements.dev.txt.

A new challenger appears: ./setup.cfg (fichier à créer)

```
[tool:pytest]
addopts = --cov-fail-under 100 --cov . --cov-report term:skip-covered
--cov-report term-missing
```

```
> pytest
```

Petits ajouts (2/2)

```
pip install pytest-flask

Mettez à jour votre fichier de  requirements.dev.txt (sic)
```

https://github.com/pytest-dev/pytest-flask

Petit aparté sur pytest :

Injection de dépendance
Fixtures
Conftests
…

Nouveaux tests... refactor

conftest.py

```python
import pytest

from app import create_app

@pytest.fixture
def user_name():
    return "adrien"


@pytest.fixture
def app():
    app = create_app()
    return app
```

app.py

```python
from flask import Flask, render_template

def create_app():
    app = Flask(__name__)

    @app.route('/')
    def index():
        return '<h1>ECM Bonjour</h1>'


    @app.route('/user/<name>')
    def user(name):
        return render_template('user.html', name=name)

    return app
```

Nouveaux tests... (pas si nouveau)

tests/test_app.py

```python
from flask import Flask

def test_app(app):
    assert app is not None
    assert isinstance(app, Flask)


def test_index_route(client):
    response = client.get("/")
    assert response.status_code == 200
    assert "ECM" in response.get_data(as_text=True)
```

(suite)

```python
def test_index_user(client, user_name):
    response = client.get(f"/user/{user_name}")
    assert response.status_code == 200
    assert f"Hello {user_name}" in
response.get_data(as_text=True)


def test_user_template(app, client, user_name):
    response = client.get(f"/user/{user_name}")
    template = app.jinja_env.get_template('user.html')
    assert template.render(name=user_name) ==
response.get_data(as_text=True)
```

Nouveaux tests... (pour de vrai)

conftest.py

```
from flask import template_rendered
...
@pytest.fixture
def captured_templates(app):
    recorded = []

    def record(sender, template, context, **extra):
        recorded.append((template, context))

    template_rendered.connect(record, app)
    try:
        yield recorded
    finally:
        template_rendered.disconnect(record, app)
```

(pip install blinker)

tests/test_app.py

```
def test_user_view_uses_correct_template(
        client, captured_templates, user_name
):
    response = client.get(f"/user/{user_name}")
    assert len(captured_templates) == 1

    template, context = captured_templates[0]

    assert template.name == "user.html"

    assert "name" in context
    assert context["name"] == user_name
```

branch: first_templates

Premier point sur notre code

On a notre première page, c'est super... mais après ?

Quelles sont les limites auxquelles nous allons être confrontés ?
Que peut-on améliorer actuellement ?

Un coup de polish

Quelques frameworks
https://bulma.io/
https://material.io/design/
https://getbootstrap.com/
https://tailwindcss.com/
...

Pour aujourd'hui et avec flask :
https://pythonhosted.org/Flask-Bootstrap/

Métier: **front-end**

✨

API - plus de template ?

```python
@app.route('/professor')
def my_api_route():
    return {
        "name": "Adrien",
        "birthday": "02 January",
        "age": 85,
        "sex": None,
        "friends": ["Amadou", "Mariam"]
    }
```

—

```python
def test_professor_view(app, client):
    response = client.get("/professor")
    assert response.json["name"] == "Adrien"
```

branch: first_api

Métier: **back-end engineer**

POSTMAN

https://www.getpostman.com/

Installez cet outil et essayer immédiatement de pinger votre application flask

Postman est un exemple parmi d'autres d'outils vous permettant de :

Consommer vos APIs / Tester vos routes
Designer/Mocker Paramétrer des défauts
Garder une liste de toutes les routes de votre API
Partager entre collègues/communautés vos configurations

Equivalent: hoppscotch, insomnia, ...

Base de données

Métier: **DBA**

Pas de SQL aujourd'hui → ORM
SQLAlchemy and Flask - voir https://flask-sqlalchemy.palletsprojects.com/en/2.x/

```
pip install flask-sqlalchemy flask-migrate pytest-flask-sqlalchemy
```

app.py

```
import os
from flask import Flask, render_template
from flask_migrate import Migrate
from flask_sqlalchemy import SQLAlchemy
from sqlalchemy.orm import DeclarativeBase

class Base(DeclarativeBase):
   pass

basedir = os.path.abspath(os.path.dirname(__file__))
db = SQLAlchemy(model_class=Base)
```

Base de données

Dans `create_app` :

```
app.config['SQLALCHEMY_DATABASE_URI'] = f"sqlite:///{os.path.join(basedir, 'data.sqlite')}"
app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False

db.init_app(app)

from tasks.models import Task

Migrate(app, db)
```

Mon premier modèle

tasks/models.py (+ __init__.py)

```
import datetime

from app import db

from sqlalchemy import Boolean, DateTime, Integer, String
from sqlalchemy.orm import Mapped, mapped_column

class Task(db.Model):
    __tablename__ = 'Task'

    id: Mapped[int] = mapped_column(Integer, primary_key=True, autoincrement=True)
    title: Mapped[str] = mapped_column(String(255), unique=True, index=True)
    creation_date: Mapped[datetime.datetime] = mapped_column(DateTime, default=datetime.datetime.utcnow)
    done: Mapped[bool] = mapped_column(Boolean, default=False)

    def __repr__(self):
        return self.title
```

Ma première migration

> flask db init
> flask db migrate
> flask db upgrade
…

```
def upgrade():
    # ### commands auto generated by Alembic - please adjust! ###
    op.create_table('Task',
    sa.Column('id', sa.Integer(), nullable=False, autoincrement=True),
    sa.Column('title', sa.String(length=255), nullable=True),
    sa.Column('creation_date', sa.DateTime(), nullable=True),
    sa.Column('done', sa.Boolean(), nullable=True),
    sa.PrimaryKeyConstraint('id')
    )
    op.create_index(op.f('ix_Task_title'), 'Task', ['title'], unique=True)
    # ### end Alembic commands ###

def downgrade():
    op.drop_index(op.f('ix_Task_title'), table_name='Task')
    op.drop_table('Task')
```

Tester un modèle ?

Ma première factory avec factory-boy!

tasks/tests/factories.py

```python
from app import db

import factory
from tests import common
from ..models import Task


class TaskFactory(factory.alchemy.SQLAlchemyModelFactory):
    title = factory.Faker("text")

    class Meta:
        model = Task
        sqlalchemy_session = common.Session
```

tasks/tests/test_models.py

```python
from .factories import TaskFactory


def test_task_repr(session):
    task = TaskFactory()
    session.commit()
    assert repr(task) == task.title
```

tests/common.py

```python
from sqlalchemy.orm import scoped_session, sessionmaker

Session = scoped_session(sessionmaker())
```

Boom, plus rien ne marche !

conftest.py

```python
import pytest
from flask import template_rendered
from app import create_app, db as _db


def user_name(): [...]
def captured_templates(app): [...]

@pytest.fixture(scope='session')
def app(request):
    app = create_app()
    # Establish an application context before running the tests.
    ctx = app.app_context()
    ctx.push()
    def teardown():
        ctx.pop()

    request.addfinalizer(teardown)
    return app
```

```python
@pytest.fixture(scope="session")
def db(app, request):
    def teardown():
        _db.drop_all()

    _db.app = app
    _db.create_all()
    request.addfinalizer(teardown)
    return _db

@pytest.fixture(scope="function")
def session(db, request):
    connection = db.engine.connect()
    transaction = connection.begin()
    session = common.Session(bind=connection)
    db.session = common.Session

    def teardown():
        transaction.rollback()
        connection.close()
        common.Session.remove()
    request.addfinalizer(teardown)
    return session
```


branch: first_db

Vraie route API ?

app.py

```
…
@app.route('/todoz')
def my_api_route():
    tasks = Task.query.all()
    return {
        "results": [
            {
                field: getattr(task, field)
                for field in Task.__table__.columns.keys()
            }
            for task in tasks
        ]
    }
```

branch: wrong_api

There must be a better way!

Si vous tombez sur

```
sqlalchemy.exc.OperationalError: (sqlite3.OperationalError) no such table: Task
[SQL: SELECT "Task".id AS "Task_id", "Task".title AS "Task_title", "Task".creation_date AS "Task_creation_date",
"Task".done AS "Task_done"
FROM "Task"]
(Background on this error at: https://sqlalche.me/e/20/e3q8)
127.0.0.1 - - [12/Nov/2024 20:09:51] "GET /todoz HTTP/1.1" 500 -
```

Ca fait peur mais quand vous faîtes tourner pytest, vous détruisez votre base. Elle n'est donc plus "migrer" correctement.

Vous pouvez supprimer votre migration précédente et relancer les commandes

> flask db migrate

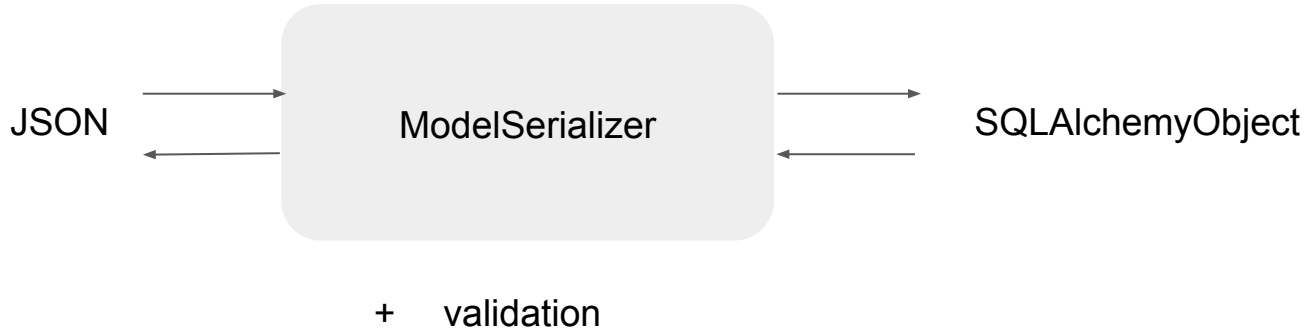
> flask db upgrade

Ce n'est pas du tout idéal…

Serialization

Install party

```
pip install marshmallow marshmallow-sqlalchemyflask-marshmallow
```



Serialization

https://flask-marshmallow.readthedocs.io/en/latest/

tasks/serializers.py

```
from app import db, ma
from .models import Task


class TaskSchema(ma.SQLAlchemyAutoSchema):
   class Meta:
       model = Task
```

app.py (+ add TaskSchema and Marshmallow import)

```
db = …
ma = Marshmallow()
[...]
@app.route('/todoz')
    def my_better_api_route():
        tasks = Task.query.all()
        return {"results": TaskSchema(many=True).dump(tasks)}
```

test_app.py

```
from tasks.tests.factories import TaskFactory
[...]
def test_todoz(app, client, session):
    task = TaskFactory()
    session.commit()

    response = client.get("/todoz")
    assert len(response.json["results"]) > 0
```

branch: correct_api

Un CRUD ?

Dernière installation : `pip install flask-smorest`

App.py

```
from flask_marshmallow import Marshmallow
from flask_smorest import Api, Blueprint, abort

ma = Marshmallow()
…
app.config['API_TITLE'] = 'My ECM API'
app.config["API_VERSION"] = "1"
app.config["OPENAPI_VERSION"] = "3.0.2"
app.config["OPENAPI_URL_PREFIX"] = "/openapi"
app.config["OPENAPI_SWAGGER_UI_PATH"] = "/api"
app.config["OPENAPI_SWAGGER_UI_URL"] =
"https://cdn.jsdelivr.net/npm/swagger-ui-dist/"

…
api = Api(app)
ma.init_app(app)

from tasks.views import task_blueprint
api.register_blueprint(task_blueprint)
```

tasks/views.py

```
from flask.views import MethodView
from flask_smorest import Blueprint, abort
from app import db
from .models import Task
from .serializers import TaskSchema
from sqlalchemy.orm.exc import NoResultFound

task_blueprint = Blueprint(
    "tasks", "tasks", url_prefix="/tasks", description="Operations on tasks"
)

@task_blueprint.route("/")
class Tasks(MethodView):
    @task_blueprint.response(200, TaskSchema(many=True))
    def get(self):
        return db.session.query(Task).all()

    @task_blueprint.arguments(TaskSchema(only=["title"]))
    @task_blueprint.response(201, TaskSchema)
    def post(self, new_data):
        try:
            task = Task(**new_data)
            db.session.add(task)
            db.session.commit()
            return task
        except Exception as e:
            abort(400, message="An error occurred")
```

Un CRUD !

tasks/views.py (suite)

```python
@task_blueprint.route("/<int:task_id>")
class TasksById(MethodView):
    @task_blueprint.response(200, TaskSchema)
    def get(self, task_id):
        try:
            return db.session.get_one(Task, task_id)
        except NoResultFound:
            abort(404, message="Task not found")

    @task_blueprint.arguments(TaskSchema(only=["title", "done"]))
    @task_blueprint.response(200, TaskSchema)
    def put(self, update_data, task_id):
        try:
            task = db.session.get_one(Task, task_id)
        except NoResultFound:
            abort(404, message="Task not found")
...
        try:
            task.title= update_data["title"]
            db.session.add(task)
            db.session.commit()
        except Exception as e:
            abort(400, message="An error occurred")
        return task

@task_blueprint.response(204)
def delete(self, task_id):
    """Delete task"""
    try:
        task = db.session.get_one(Task, task_id)
        db.session.delete(task)
        db.session.commit()
    except NoResultFound:
        abort(404, message="Task not found")
```

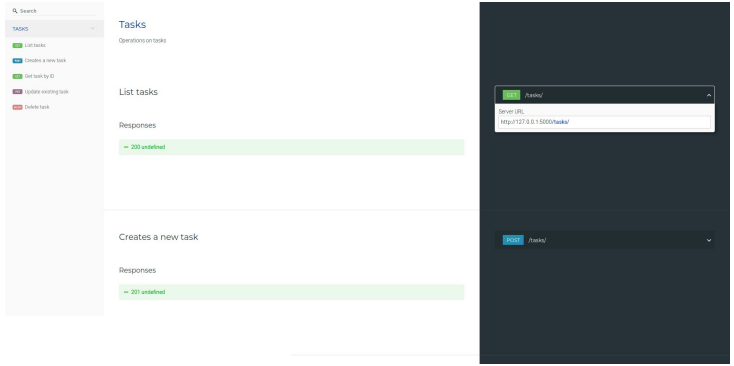
branch: first_crud

(Pour les tests)

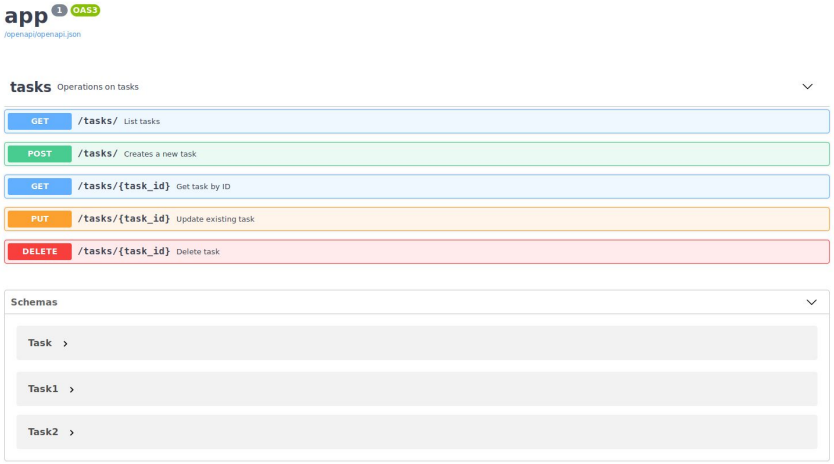
OpenApi en cadeau

```
> flask run
```

http://127.0.0.1:5000/openapi/api



Swagger U
ReDoc



Être prêt pour le front qu'on utilisera demain

Anticiper les problèmes de Cors

En fait c'était pas la dernière installation : `pip install flask-cors`

App.py

```
from flask_cors import CORS
[...]
   app = Flask(__name__)
   CORS(app)
```

Faire évoluer nos routes d'update

branch: front_ready

Si vous avez fini…

(Une seule personne a réussi en 5ans.)

Faire vos services pour un site “basique” :

SERVICES PUBLIQUES :
Un service d’authentification (email + password) via jwt (voir `flask-jwt + flask-bcrypt ou argon2`)

SERVICES PRIVÉS
Un service pour récupérer les informations utilisateurs (username, date de création, mise à jour password)
Un service de gestion de todolist (CRUD) (fait ensemble)

En avance ?

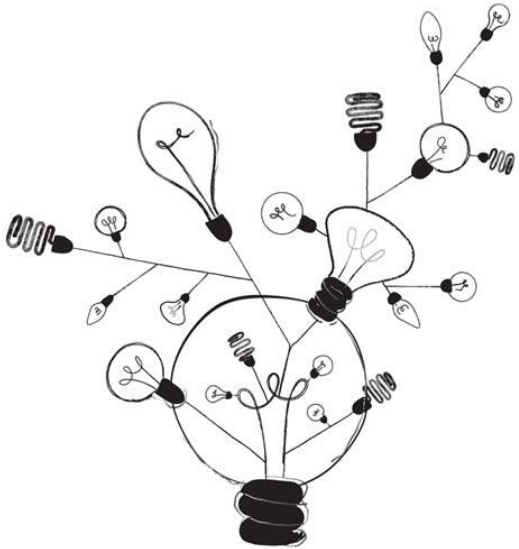
De nombreux sujets peuvent être explorés à partir de ces bases.

Exemples :

Améliorer les tests et la documentation
Limiter le nombre de queries
Servir des fichiers statiques
Rendre le total plus esthétique
Séparer le code en modules

Vous pouvez aussi utiliser vos propres modèles de données / structures si cela peut vous aider pour un de vos projets 3A.

Fin journée 1



Côté front

Prérequis:

node >22 & npm
HTML & CSS
JavaScript simple.
Compréhension sommaire du DOM
Connaissance de la syntaxe ES6

Objectifs:

Découverte de react en milieu naturel
Première app et state local
Un tic-tac-toe [optionnel]
Create-react-app et première application
Communication avec notre back-end en flask

React

Comme pour flask, le but n'est pas de vous montrer la dernière techno à la mode mais une qui est "éprouvé", largement utilisé et qui vous permet d'être évolutif.

- Une lib JavaScript https://github.com/facebook/react presque 200k stars
- Pas un framework (contrairement à Angular)
- Projet open-source, créé et maintenu entre autre par Facebook
- Utilisé pour faire des interfaces utilisateurs
- La "**view"** d'une application en MVC (Model View Controller)

React - into the wild

index.html

```
<!DOCTYPE html>
<html>
 <head>
   <meta charset="utf-8" />
   <title>Hello React!</ title>

   <script src="https://unpkg.com/react@^16/umd/react.production.min.js"  ></script>
   <script src="https://unpkg.com/react-dom@16.13.0/umd/react-dom.production.min.js"  ></script>
   <script src="https://unpkg.com/babel-standalone@6.26.0/babel.js"  ></script>
 </head>

 <body>
   <div id="root"></div>
   <script type="text/babel">
     // React code will go here
   </script>
 </body>
</html>
```

React - into the wild

Mon premier code react

```
class App extends React.Component {
  render() {
    return <h1>Hello world!</h1>
  }
}

ReactDOM.render(<App />, document.getElementById('root'))
```

Est-ce satisfaisant ?

create react app

```
npm install -g  npx
```

https://github.com/facebook/create-react-app

```
npx create-react-app react-tutorial
```

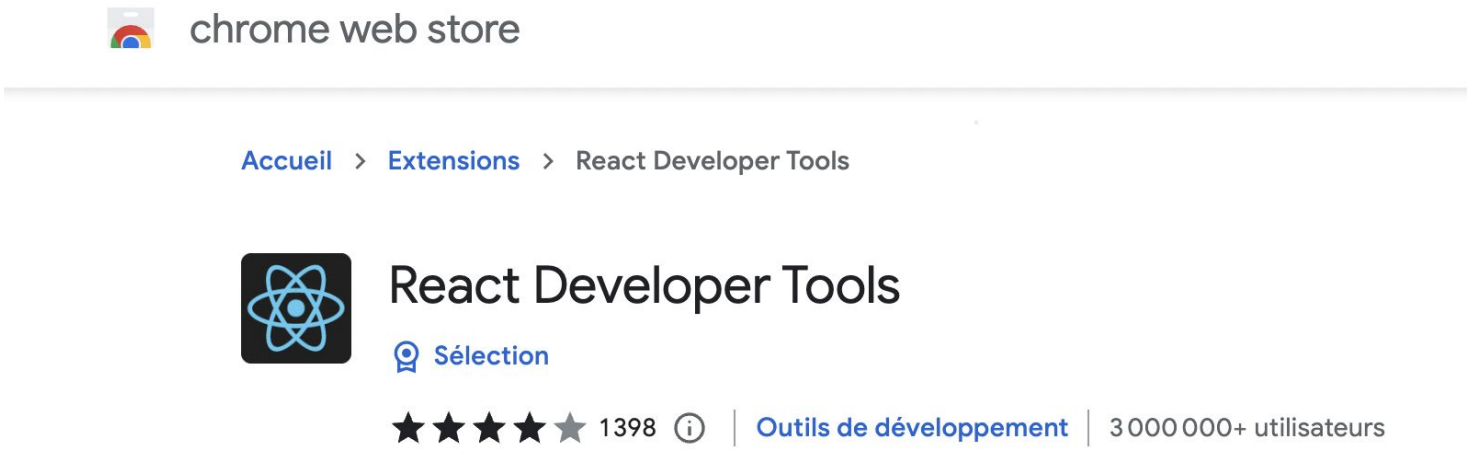
(installez npx si besoin)

```
cd react-tutorial
npm start
```

Prenez 5 min pour modifier le code et le voir en action

react developer tools

Aidez vous et installez tout de suite ce plugin chrome :
https://chrome.google.com/webstore/detail/react-developer-tools/fmkadmapgofadopljbjfkapdkoienihi



Cette équation est fausse

JSX : JS + XML = HTML ?

```
const heading = <h1 className="site-heading">Hello, React</h1>
```

⇒ Cela revient à écrire

```
const heading = React.createElement('h1', {className: 'site-heading'}, 'Hello, React')
```

- **className**
- Properties & methods en JSX → camelCase - ex: onclick → onClick.
- Écriture stricte du XML (closing tag…)

Templating

```
const name = 'Tania'
const heading = <h1>Hello, {name}</h1>
```

Les composants et un premier hook

App.js

```
import { useState } from 'react'

const MyApp = () => {
 const [counter, setCounter] = useState(0)

 return (
   <>
     <p>Count: {counter}</p>
     <button onClick={() => setCounter(counter + 1)}>Increment</button>
   </>
 )
}
```

Props

App.js

```
const MyApp = ({ defaultCounter = 0 }) => {
 const [counter, setCounter] = useState(defaultCounter);

 return (
   <p>Count: {counter}</p>
   <button onClick={() => setCounter(counter + 1)}>Increment</button>
 )
}
```

Let’s play ! et… discutons du state

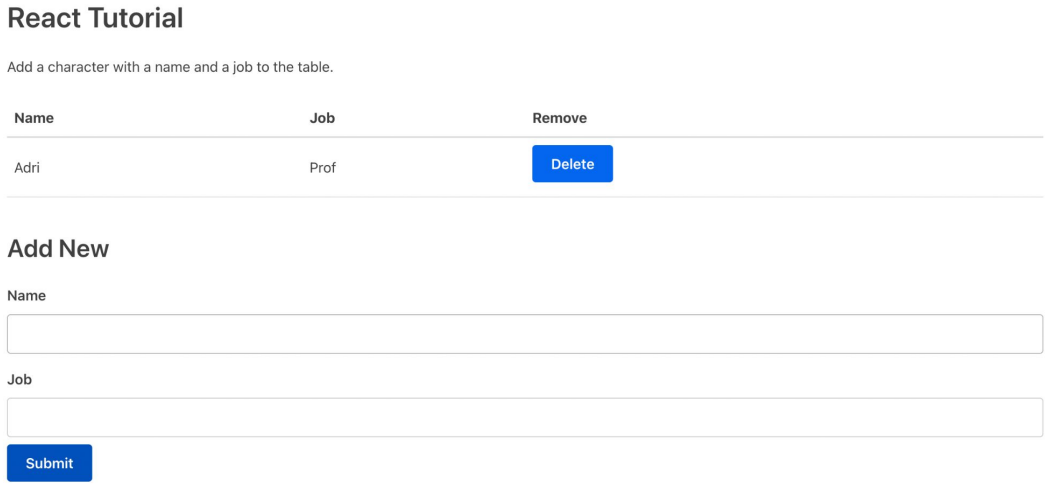
https://react.dev/learn/tutorial-tic-tac-toe

https://redux.js.org/
react context

Une première app

https://github.com/taniarascia/react-tutorial

Nous allons développer cette interface

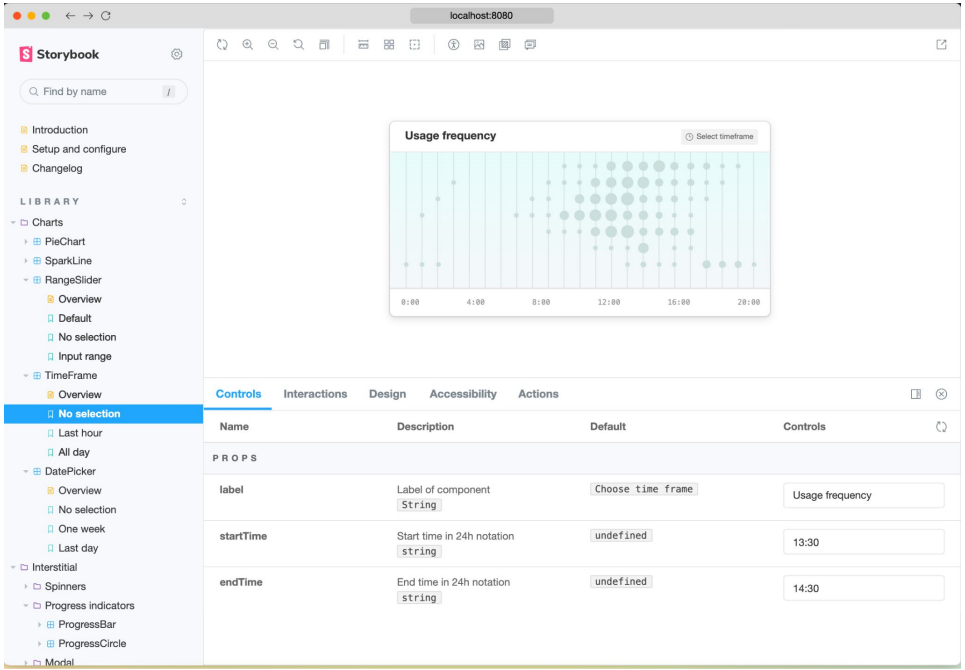


https://taniarascia.github.io/react-tutorial/
TD : https://www.taniarascia.com/getting-started-with-react/#class-components

Storybook

https://storybook.js.org/

```
npx storybook init
npm run storybook
```



https://storybook.js.org/docs/react/get-started/install

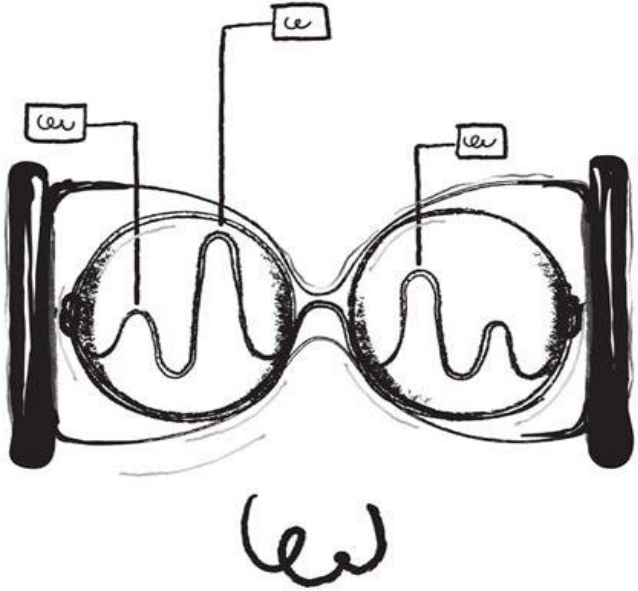
Let's code together

TD : faire une todo list à partir du backend J1

Nous avons tous les éléments pour construire une interface pour notre backend flask !

Rappel: le back est sur https://github.com/adrien-may/ecm2024 branch: front_ready

https://github.com/adrien-may/react-todo



On triche avec github pages

Mise en production ?

```
npm install --save-dev gh-pages
```

```
"scripts": {
  // ...
  "predeploy": "npm run build",
  "deploy": "gh-pages -d build"
}
```

```
npm run deploy
```

https://pages.github.com/

Métiers

Architecte
Front-end engineer
Back-end engineer
Q&A engineer - testing engineer
Devops engineer
DBA

Full stack ?

Et autour gravite bien d'autres métiers : security engineer, SRE, PO, sales...

Vos nouvelles compétences

WEB - Vous avez de meilleurs notions de comment les sites/app fonctionnent

BACKEND - Vous êtes capable de modéliser vos données et créer les services qui les manipulent

FRONTEND - Vous avez la capacité de construire une interface pour vos utilisateurs

API - Vous êtes en mesure d'écrire et consommer une API

Testing - Vous avez des bases en test et TDD, mock, fixtures

Aller plus loin

Livres (Orchestration, Micro services, Docker, Kubernetes, Data/IA services...)

Performances

Déploiement (production, essayer aws, azure, gcp ?)

Monitoring (logs, alertes, etc...)

Qualité (monitoring, tests E2E, auto scaling)

Product

Des gens dans le parcours entrepreunariat ? Des questions ?!

Merci

Des questions ? Du feedback ?

PS: On recrute, adrien@may-sante.com (et pas qu'en web : mobile, infra, IA LLM....)