Flask Rest API

Antonio Pisanello

PSE_2024_0203

Nomades 2024

Rest Introduction :

Protocol HTTP :

Request

- Methods (Get, post, Put, delete, patch)
- URL → endpoint (paramètres d'URL)

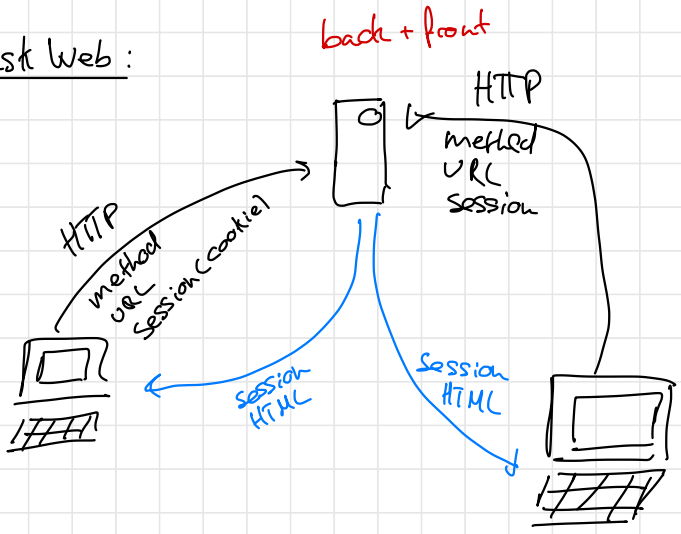
/users/:id
variable

- body
- headers
- paramètres

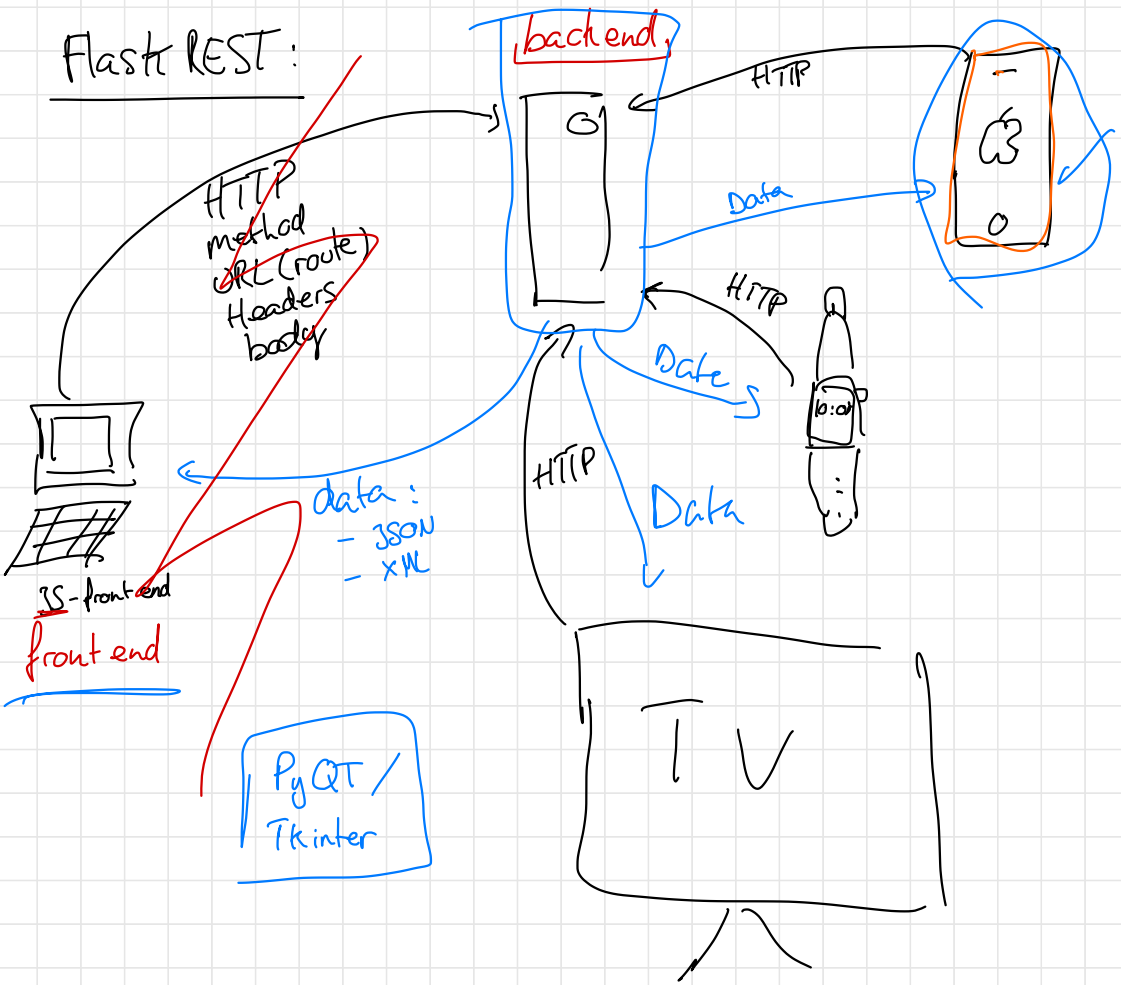
Response :

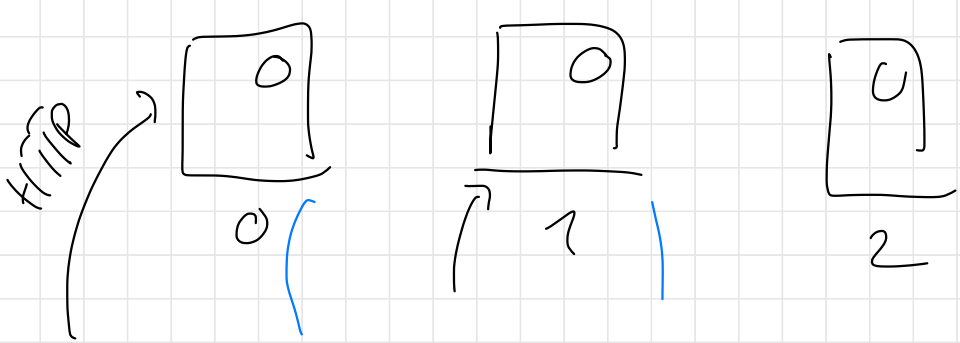
- Status code (200, 201, 404, 418)

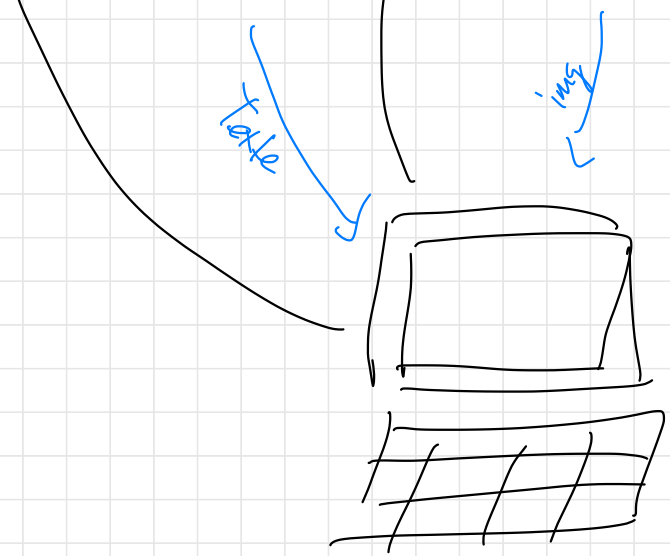
Flask Web :



Flask REST:


backend
HTTP
Data
HTTP
Data
HTTP
Data
HTTP
method
URL (route)
Headers
body
data:
- JSON
- XML
JS - front end
front end
PyQT /
Tkinter
TV


HTTP
0
0
1
0
2



HTTP Methods : (Request)

Get : utilisé pour récuperer les données (ok [200])

post : utilisé pour inserer des données (create [201])

PUT : utilisé pour update tout un objet (ok [200])

Patch : utilisé pour update juste une partie de l'objet (200)

delete : utilisé pour supprimer l'objet (204)

HTTP Status code : (Response)

1xx : Information
2xx : Success
3xx : redirect
4xx : client errors (you fucked up)
5xx : server errors (I fucked up)

Routes (endpoints) : → myself (localhost)

http://127.0.0.1:5000/users

protocole | URL | Port | endpoint

GET http://127.0.0.1:5000/users → Retourne list[User]

POST {"fn": "Alex", "ln": "..."} → Insère un nouvel utilisateur

Get http://127.0.0.1:5000/users/7 → Retourne User

7 : parametre

POST → Insere un nouvel utilisateur en specifiant l'id

PUT → Modifier l'utilisateur (complet)

PATCH → Modifie une info du user

DELETE → Supprimer l'utilisateur

Bonne Pratique

Controlleur : Gestion des routes

Service : Logique de l'application

repository : gestion des données (base de données)

DTO (data transfert object) Marshmallow

gérer les objets envoyés / reçus.

JWT (JSON web token)

- Jwt est un standard (IETF 7519) (rfc 7519)

- Jwt sont des "documents" JSON qui contiennent des informations (claims)

- Les informations sont appellés "claims", il existe plusieurs type de claims dependament des besoins

- les Jwt sont structurés: 3 parties separés par "."

  1. header: deux propriétés: - typ: le type de token
     - Alg: l'algorithm utilisé pour Signer le token

  2. Payload: contiens les claims (informations)

  3. Signature: Signature electronique (crypto) qui garantie la validité du token:
     - authenticité
     - non-repudiation

- les Jwt sont encodés: base64Url (rfc 4648)
  ↳ Utilisation de caractères qui sont "safe" pour l'envoi sous format texte (HTTP)

- Pour tester les tokens il y a un site: jwt.io

Dans le cadre des API Rest on peut distinguer deux types de tocken :

- ID tokens : renseigne des informations sur un utilisateur :
  - id de l'utilisateur
  - email
  ...

- ACCESS tokens : renseignes des informations d'acces a certaines ressources de l'API

Claims :

Voici une liste des principales claims :

- iss : issuer, l'entité (serveur, service, API) qui a emit le token
- sub : la personne a qui le token appartient (id_user)
- aud : audience, la/les targets (endpoint) pour lesquelles le token a été emis
- iat : issued at time, quand le token a été emis
- exp : expiration, quand le token n'est plus valide

pour en savoir plus sur les claims vous pouvez vous rendre sur :

iana.org/assignments/jwt/jwt.xhtml

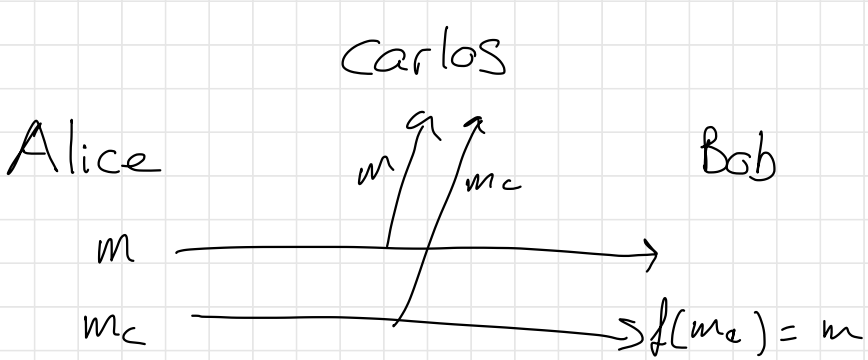
https://iana.org/assignments/jwt/jwt.xhtml

Signing algorithms :

Les tokens peuvent être signer avec deux types d'algorithmes

- HS256 : chiffre le token avec un secret
- RS256 : Chiffre le token avec une private key (RSA)

Signature : $\overset{Pob}{Priv}$

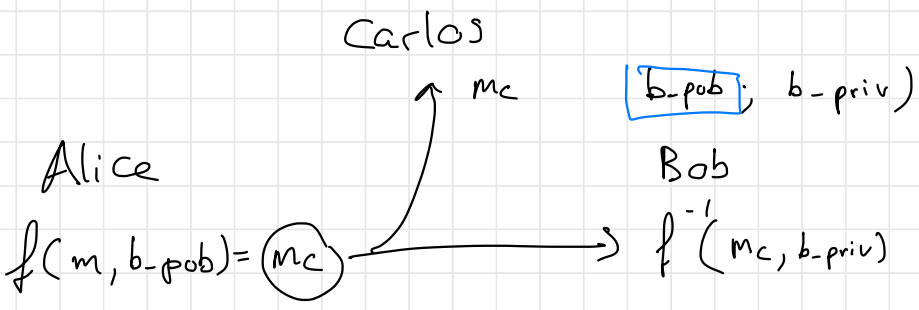


$f(m, c) \to m_c$

$f^{-1}(m_c, c) \to m$

Symetric

Asymetric



$b\_pob = \frac{1}{5}$

$b\_priv = 5$

Alice

det

$f(x, k) = x \cdot k$

$f(m, b\_pob \overset{\frac{1}{5}}{=}) = m \cdot b\_pob$

$= m \cdot \frac{1}{5}$

$= \frac{m}{5} \to m_c$

Bob

$f^{-1}(y, Q) = y \cdot Q$

$f^{-1}(m_c, b\_priv) = \underbrace{m \cdot b\_pub}_{\frac{m}{5}} \cdot \underbrace{b\_priv}_{5} = m$

a_pub
a_priv
Alice

b_pub
b_priv
Bob

$f(m, a\_priv) \rightarrow m_c$

(m, m_c)