Gestion des données

Données sur le web

Olivier Schwander
<olivier.schwander@sorbonne-universite.fr>

Master Statistiques
Sorbonne Université

2022-2023

Rôles d'un service web

Récupérer des données

- Un peu comme une base de données
- mais spécialisé pour un usage particulier

Utiliser un service

- Envoyer ses données
- Demander un traitement
- Récupérer un résultat

Web

Site web classique

- Navigateur qui demande des pages
- Serveur qui renvoie les pages

Programmer les échanges

- Programme qui récupère des informations

Examples

- Applications mobiles

Architecture

Client/serveur

- Client: une application quelconque
- Serveur: toujours un serveur web

Réponses

- Plus des pages web
- Données structurées

Réponses

Données stockées dans divers formats

- XML
- Json
- Binaire
- Propriétaire

Utilisation

- Comme un fichier classique
- On peut oublier d'où vient l'information

Protocoles

Representational state transfer (REST)

- Question codée dans l'URL

http://example.com/age/capitaine?unit=year&format=json

Autres: SOAP et WSDL

- Plus complexe
- Plus structuré

Hypertext Transfer Protocol (HTTP)

(version ultra-simplifiée)

Requêtes GET

- Demander une information
- Requête la plus classique

Requêtes POST

- Envoyer une information
- Soumission d'un formulaire

Uniform Resource Locator (URL)

Format

http://example.com/chemin/vers/la/ressource?arg1=valeur1&arg2=valeur2

- `http`: protocole
- `example.com`: serveur
- `chemin/vers/la/ressource`: identifiant de la ressource
- `?`: tout ce qui suit est un argument
- `arg1=valeur1&arg2=valeur2`: arguments

Examples

- http://www.bing.com/search?q=http
- https://schwander.isir.upmc.fr/enseignement/m2stat_gd/index.html

Requêtes

En général

- Récupérer des données: GET
- Envoyer des données: POST

Cas particuliers

- Petites données à envoyer: GET
- Requête compliquée: POST

Arguments

- Seules les requêtes GET prennent des arguments dans l'URL
- Autre mécanisme pour POST

Mécanisme général du protocole HTTP

Étapes d'une connexion

- Connexion au serveur: nom de domaine, adresse ip, port
  - Éventuellement avec *chiffrement* (http**S**)
- Requête:
  - Méthode: GET/POST/etc
  - Ressource (chemin, argument pour GET)
  - En-têtes: métadonnées (authentification, cookies)
  - Contenu (rien pour GET, données quelconques pour POST)
- Réponse:
  - En-têtes
  - Contenu (données quelconques)

Authentification

Nom d'utilisateur / mot de passe

- Possible à différents niveaux
- niveau protocole: prévu dans http
- niveau application: formulaire dans la page web

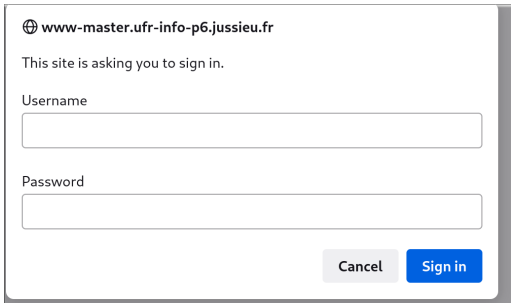
Jeton d'authentification

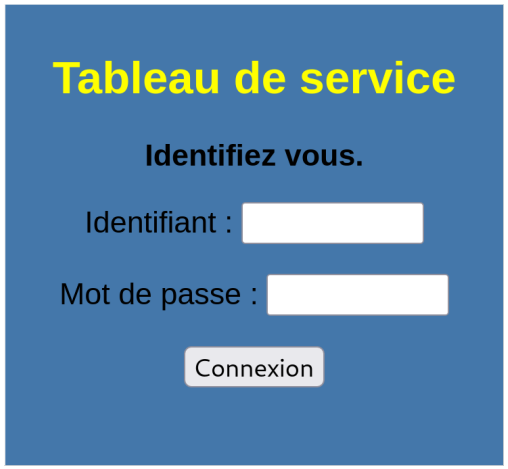
- Valeur pseudo aléatoire qui identifie un utilisateur
- dans une en-tête http
- dans un argument GET

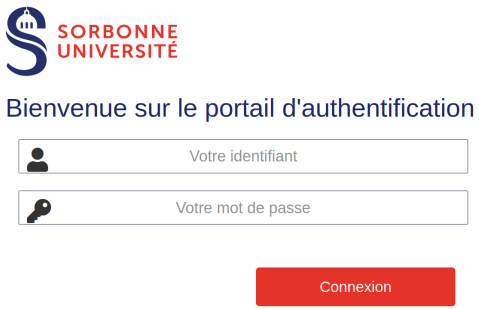
Authentification centralisée

- Zero-knowledge
- en fait un jeton d'authentification

Authentification







En Python

Requests: HTTP for Humans

Warning: Recreational use of other HTTP libraries may result in dangerous side-effects, including: security vulnerabilities, verbose code, reinventing the wheel, constantly reading documentation, depression, headaches, or even death. (source: http://docs.python-requests.org/en/master/)

Exemples

Base

```
>>> import requests
>>> result = requests.get("http://www.bing.com/search?q=htt
>>> print(result.text)
```

Json

- Requests se charge du chargement des données

```
>>> import requests
>>> r = requests.get('https://api.github.com/events')
>>> r.json()
[{u'repository': {u'open_issues': 0, u'url': 'https://githu
```

- On récupère directement une valeur Python

Trouver le service web

Documentation

- Si le service est officiellement supporté

Reverse engineering

- Analyse des requêtes d'une page web
- Étude d'une application mobile

Services web

Cas idéal

- Gentil fournisseur de service
- Documentation, formats ouverts, accès autorisé

~~Parfois~~ Souvent

- Pas de service web public
- Format incompréhensible
- Conditions d'utilisation incompatibles
- Pas l'info qui vous intéresse

Que faire sans service web ?

Définition

Extraction de données

- À partir d'une page web
- Une page prévue pour être affichée
- Reverse engineering

Non-coopératif

- Pas une API !
- Aucune garantie de stabilité
- Pas forcément autorisé

Pas de service web ?

Mais pourtant, on peut afficher les pages avec un navigateur.

Extraction des données

- La page web contient les données
- Récupérons-les manuellement

Étapes

- Faire des requêtes HTTP
- Extraire des données dans les réponses

Intérêts

Automatisation de tâches

- Scripts: actions à faire dans une application web
- Tests: simulation d'un.e utilisateurice humain.e
- Business intelligence: surveillance des prix sur un site web concurrent

Sources de données pour l'IA

- Téléchargement de textes ou d'images
- Construction d'un dataset
- Prédiction

Risques

Risque légal

- Pas forcément autorisé
- Conditions générales d'utilisation
- Comportement responsable indispensable

Risque technique

- Travail important
- Mises à jour fréquentes
- Risque d'être cassé souvent

Exemples en production

woob: Web Out Of Browser

https://woob.tech

- Interfaces pour énormément de sites
- Surtout des banques, mais pas que
- Support professionnel

Zotero

https://www.zotero.org/

- Base de données bibliographiques
- Récupérations des métadonnées de publications scientifiques
- Largement utilisé par les chercheureuses

Exemples en production

Scrapy

https://scrapy.org/

- Bibliothèque Python
- Facilite le parcours d'un site
- Support professionnel

Selenium

https://www.selenium.dev/

- Automatise un navigateur
- Firefox, Chrome, Edge
- Tests automatiques

Web crawling

- Moteurs de recherche
- Archivage, surveillance

Le web

Classique

- une requête GET sur une URL
- du code HTML en réponse: la page est un document
- la page est constituée uniquement par ce code HTML

Et moderne

- une requête GET sur une URL
- un morceau de l'information
- et du code Javascript: la page est un programme
- le code effectue d'autres requêtes, et construit la page qui s'affiche

Démarche

Comprendre les URL

- Surfer normalement sur le site
- Remplir les formulaire, regarder où on arrive
  http://www.bing.com/search?q=chat
  http://www.bing.com/search?q=chien
- Intuition, essais, erreurs

Comprendre la structure de la page

- Où est l'information ?
- Comment l'identifier à coup sûr ?
- Comment est-elle stockée ?
- Y a-t-il des requêtes pour construire le contenu de la page ?
- Et l'authentification ?

HyperText Markup Language (HTML)

(version ultra-simplifiée)

Du XML

- (à peu près)
- Structure d'arbre
- Des nœuds, appelés balises

Les balises

- `<p>...</p>`: paragraphe de texte
- `<a href=http://destination>titre</a>`: lien hypertexte
- `<div>...</div>`: container
- `<span>...</span>`: mise en forme
- et plus encore

Exemple

```
<!DOCTYPE html>
<html>
 <head>
  <title>
   Exemple de HTML
  </title>
 </head>
 <body>
  Ceci est une phrase avec un
    <a href="cible.html">hyperlien</a>.
  <p>
   Ceci est un paragraphe sans hyperlien.
  </p>
 </body>
</html>
```

Trouver son chemin

Attributs: meta-données sur les balises

- `id`: identifiant unique d'un nœud
- `class`: partagé par plusieurs nœuds

Utilité normale: mise en forme

- Nœud `id=maintitle` en rose avec des poneys
- Nœud `id=privacy` en tout petit
- Nœuds `class=specialoffer` en rouge clignotant

Corollaire

- On peut trouver les informations intéressantes

En Python

```
>>> import requests
>>> from bs4 import BeautifulSoup
>>> url = "https://fr.wikipedia.org/wiki/Uniform_Resource_L
>>> page = requests.get(url)
>>> print(page.status_code)
200
>>> soup = BeautifulSoup(page.text, "html.parser")
>>> soup.title
<title>Uniform Resource Locator \u2014 Wikip\xe9dia</title>
>>> soup.title.string
u'Uniform Resource Locator \u2014 Wikip\xe9dia'
>>> soup.find("h2")
<h2>Sommaire</h2>
```

Recherche et filtres

Trouver un nœud

- `soup.find("div")`
- `soup.find("div", class="offer)`
- `soup.find("div", id="firstHeading")`

Trouver tous les nœuds

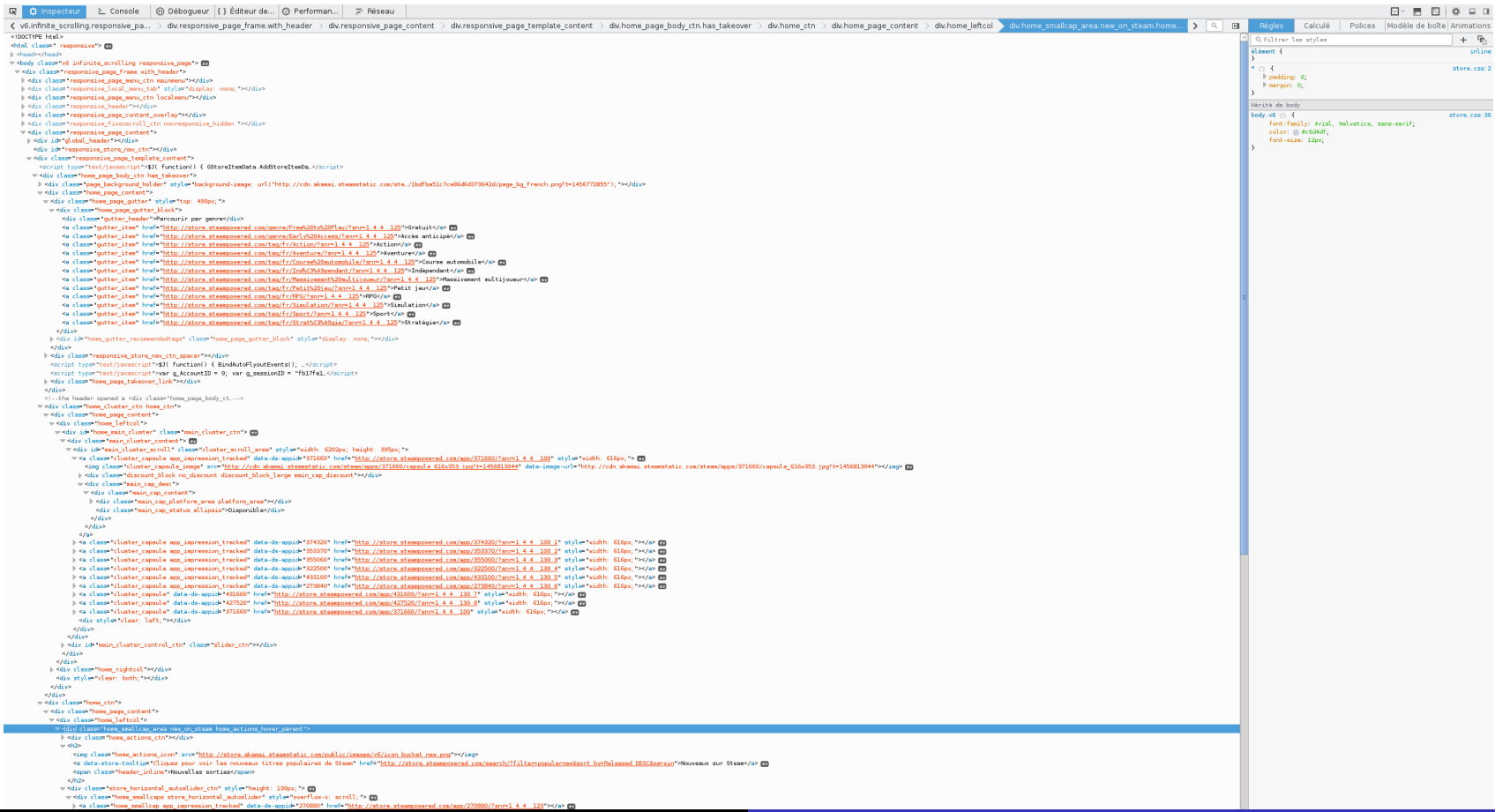
- `soup.find_all("div")`

Parents, enfants

- `.find_parents()`, `.find_parent()`
- `.content[0]`
- `.next_siblings .previous_siblings`

Inspecteur Firefox

- Accessible avec F12
- Pointer sur la page pour trouver le nœud correspondant
- Suivre les requêtes HTTP et leurs arguments



Pourquoi Javascript ?

(version ultra-simplifiée)

Exemple d'un webmail

- Requête 1: le squelette de l'interface
- Requête 2: une liste d'identifiants des messages
- Requêtes suivantes: les détails des messages

Intérêt pour le développement web

- Affichage plus rapide de l'interface
- Messages affichés au fur et à mesure
- Mise à jour morceau par morceau
- Globalement: plus agréable pour l'utilisateur

Mais pour l'extraction ?

Plus compliqué

- Tout n'est pas fourni dans la requête visible
- Plus de requêtes à faire
- Des requêtes compliquées

La page est un programme

- Exécuter ce programme
- Comprendre ce programme

Exécuter le programme de la page web

Il faut un vrai navigateur web.

Selenium

- À la base, un outil pour automatiser des tests sur des sites web
- Interface pour programmer des interactions avec un site

Architecture

- Un vrai navigateur web (Firefox ou Chrome)
- mais télécommandé
- par un script Python

Interaction

- Le navigateur travaille normalement
- Accès à la page construite par l'exécution du code
- Extraction en parlant au navigateur

Selenium

```
from selenium import webdriver
from selenium.webdriver.common.keys import Keys

driver = webdriver.Firefox()
driver.get("http://www.python.org")
assert "Python" in driver.title
elem = driver.find_element_by_name("q")
elem.clear()
elem.send_keys("pycon")
elem.send_keys(Keys.RETURN)
assert "No results found." not in driver.page_source
driver.close()
```