

Documentation du projet Picshare

Groupe : SaLeeMas

Objectif

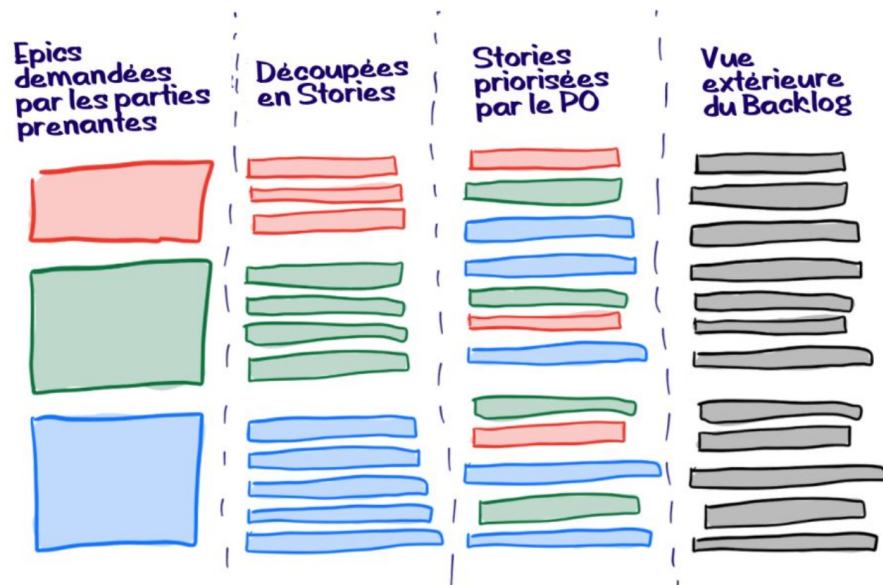
Objectifs du projet

- ☐ Utilisez le moteur de templates pour rendre les vues Web (render_templates).
- ☐ Comprendre comment un site Web interagit avec le système de fichiers (OS).
- ☐ Savoir gérer les téléchargements de fichiers (uploads).
- ☐ S'organiser en équipe (trello, backlog product, planning).
- ☐ Savoir utiliser Flask pour répondre aux requêtes Client (client requests).
- ☐ Savoir construire une application web complète, avec un véritable front-end utilisateur, capable de répondre aux requêtes Client sur le réseau.
- ☐ Mettre à niveau une application API back-end précédente, utilisée par les programmeurs et les machines, qui ne répondait qu'avec des données brutes, en une application Web à part entière capable d'interagir avec les utilisateurs Web réels - une application qui répondra aux demandes avec des pages Web HTML.
- ☐ Construire une application de site Web pour partager des images à travers 3 pages :
 - ☐ - La page de téléchargement (upload) qui permet aux utilisateurs de télécharger leurs propres photos,
 - ☐ - La page d'index (page d'accueil) qui permet d'afficher des images à tous les autres utilisateurs.
 - ☐ - La page image (picture) qui permet d'afficher et de commenter une image sélectionnée par un utilisateur.

(c) Ce projet est largement inspiré de [9gag] (<https://9gag.com/>) ...

Organisation du projet

BACKLOG



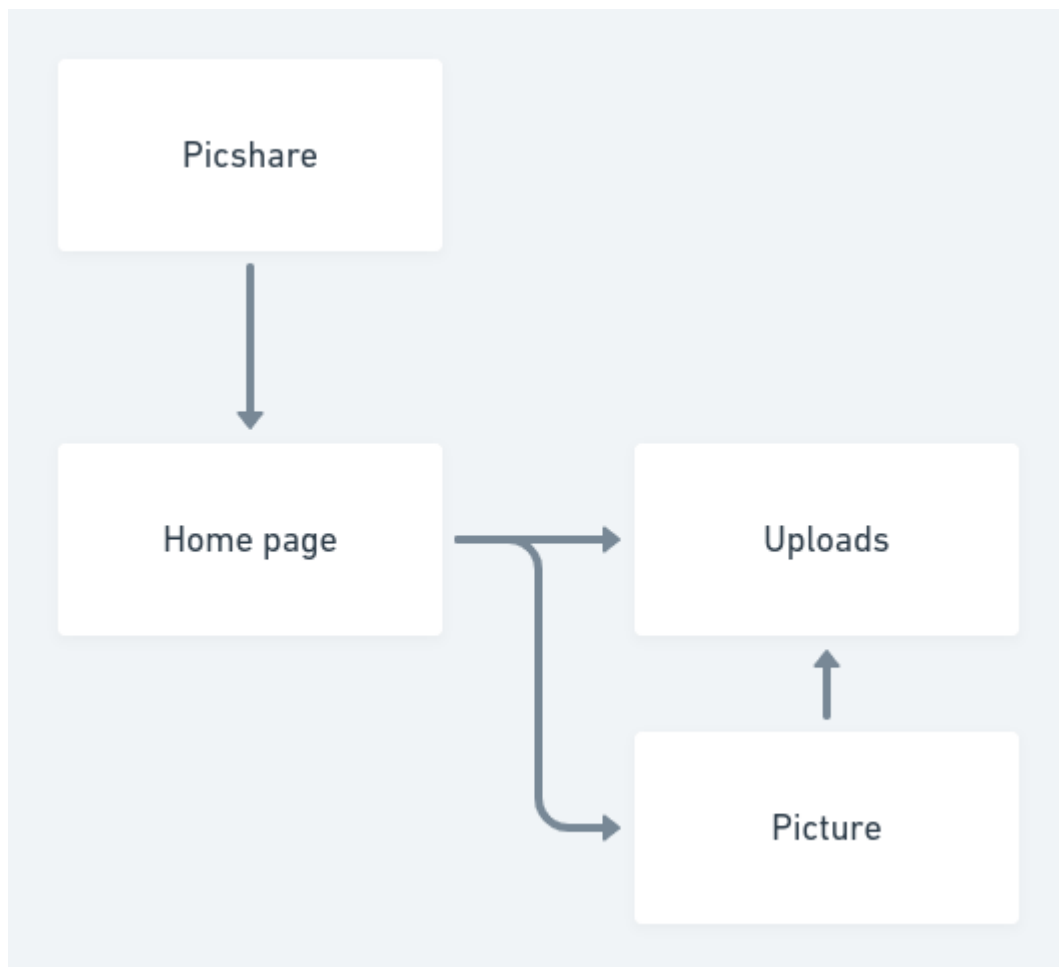
Pour ce projet en groupe, nous avons utilisé la méthodologie suivante :

- Nous avons établi un contrat d'engagements au sein de l'équipe, afin de les partager et de les respecter tout au long du projet.
- Nous avons recueilli et analysé le besoin et les exigences attendu(e)s pour le projet.
- Nous avons établi, après analyse du besoin :
 - un plan du site (site map), avec sa navigation (nav),
 - un prototypage de page en fil-de-fer (wireframing réalisé sous whimsical),
 - un carnet de commandes (product backlog) inspiré de la méthodologie Agile (réalisé sous trello).

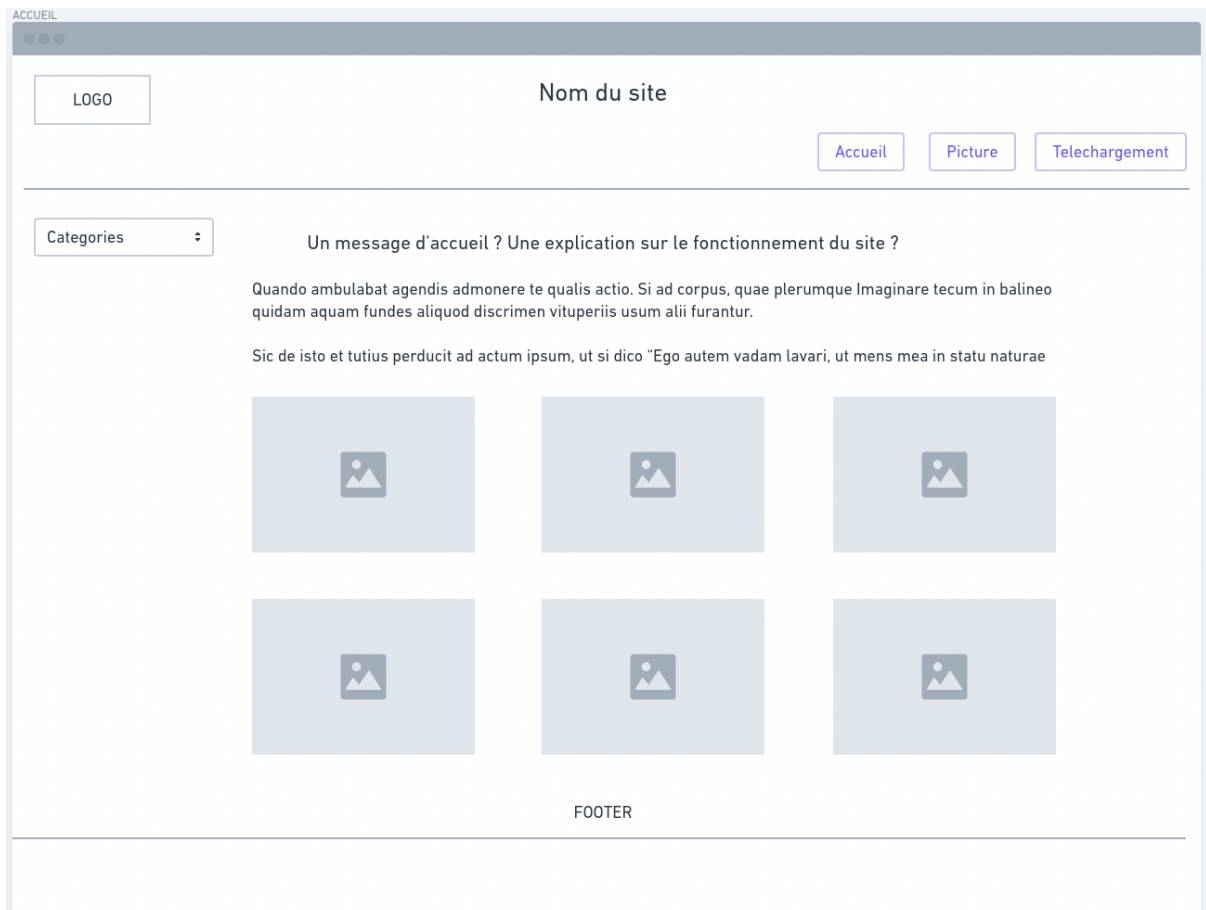
- Nous avons d'abord poussé (git push) notre backlog sur notre référentiel (Git-lab) le premier jour du projet (mardi 15 juin 2021). Notre mentor a examiné notre backlog pour s'assurer que nous allions dans la bonne direction.
- Au début de chaque semaine, nous avons organisé une réunion de planification où nous avons planifié la façon dont l'équipe devait travailler pendant la semaine. Nous avons envoyé notre planning à notre mentor via notre backlog Trello (due dates).
- Au début de chaque journée (9h45), nous avons réalisé une réunion de 15 minutes (scrum), en s'inspirant de la mêlée Agile, pour que chacun explique rapidement (5 minutes) aux autres, ses achèvements passés (done) ses encours du jour (work in progress), ainsi que ses difficultés rencontrées sans rentrer dans les détails (issues) et ses nouvelles tâches à réaliser (to do), en s'inspirant de la mêlée Agile (scrum). Parfois les réunions ont eu lieu à 16h, lorsque notre emploi du temps nous y contraignait.
- en temps réel, le tableau de bord trello, nous a permis de matérialiser cette organisation et l'avancement de la réalisation product backlog.
- Chaque jour, des réunions en binôme informelles ont eues lieu afin de résoudre les points bloquants (issues) et continuer d'avancer.
- Chaque fin de semaine vendredi, nous avons réalisé une réunion de débriefing (check-out) où nous avons discuté de ce qui a fonctionné et de ce qui n'a pas fonctionné dans notre groupe, en étant honnête avec nous-mêmes. Le but était de nous aider à mieux s'organiser dans notre travail de groupe.
- Nous avons poussé (git push) les notes de chaque Débriefing sur notre référentiel (git-lab).

Architecture fonctionnelle

Plan du site



Page d'accueil (home page)



MUST HAVE - Priorité 1 - P1

- pour afficher toutes les images,
- pour classer les images affichées par leur date de création dans l'ordre décroissant (l'image la plus récente est affichée en premier),
- pour accéder à la page de l'image, lorsque l'utilisateur clique sur la vignette de l'image,
- pour afficher la page de téléchargement d'image (upload).

SHOULD BE - Priorité 2 - P2

- pour filtrer les images affichées, via des paramètres de requête.

NICE TO HAVE - Priorité 3 - P3

- pour servir votre site Web dans un conteneur, en utilisant un simple système Docker :
 - > pour se connecter au site Web normalement via un navigateur.
- pour exporter toutes les images de la catégorie sélectionnée sous forme de fichier PDF :
 - > Lorsque l'utilisateur à filtrer les images selon une catégorie sélectionnée, pour afficher un bouton « exporter » sur la page.
 - > pour générer le PDF avec un reportlab.
- pour afficher une fenêtre modale « télécharger » sur la page d'accueil au lieu d'afficher la page d'upload.

Page de téléchargement (upload page)

TELECHARGEMENT


LOGO

Nom du site

Accueil Picture Publication

HEADER

Téléchargez votre image



Choisissez une catégorie

Titre de la photo

Ajoutez une description de votre image

Enregistrer

FOOTER

MUST HAVE - Priorité 1 - P1

- pour télécharger une nouvelle photo, avec un titre, une catégorie sélectionnée et une description.
- pour sélectionner une catégorie à associer à une image téléchargée, parmi une liste de sélection de catégories disponibles.
- pour stocker les images téléchargées dans le système de fichiers.
- pour vérifier l'homonymie entre les images téléchargées : lorsque deux images ont le même nom de fichier, le téléchargement ne doit pas enregistrer l'image tant que l'utilisateur n'a pas modifié son nom de fichier en premier
- pour contrôler le type de fichier : si le fichier téléchargé est bien une image (de type : jpeg, png, gif...) avant de la télécharger.

NICE TO HAVE - Priorité 3 - P3

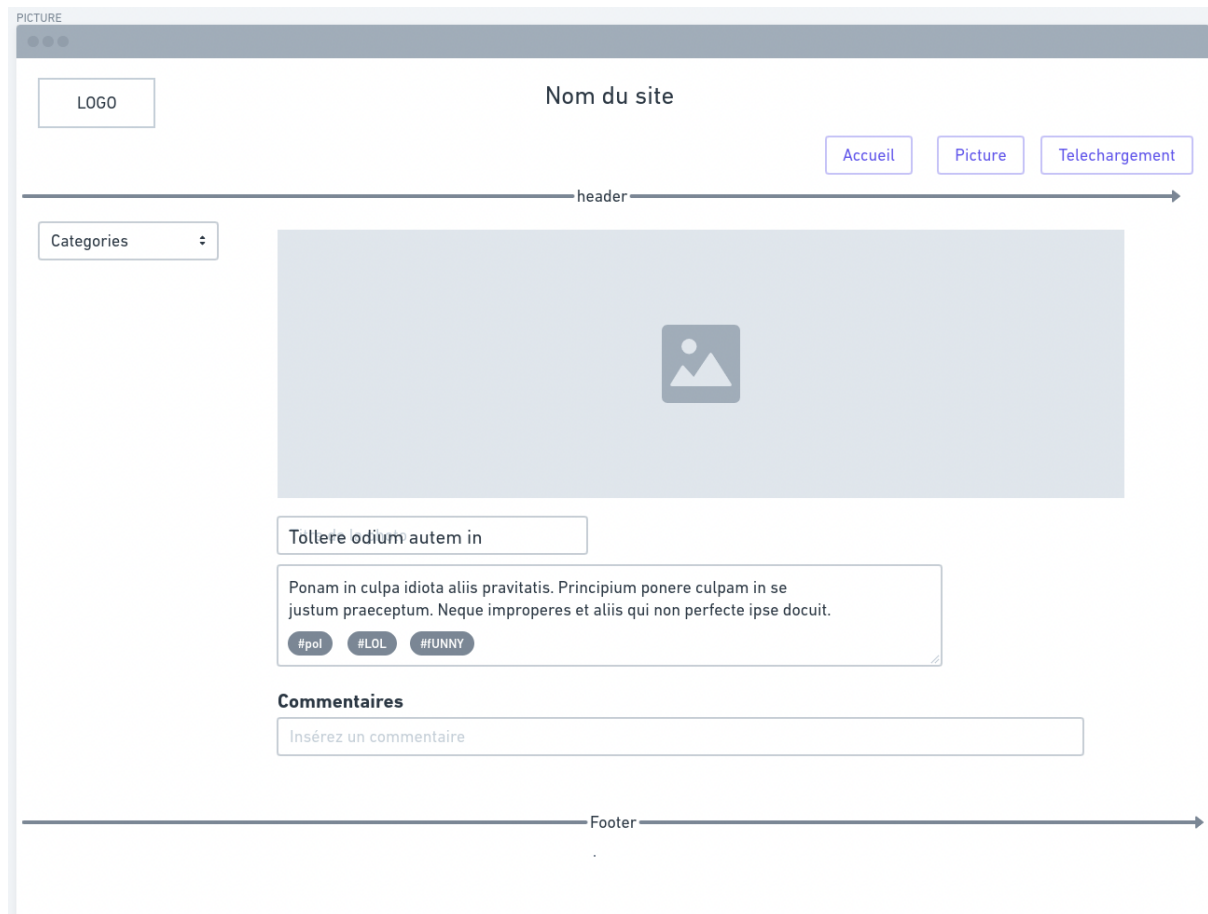
- pour gérer des mot-dièses (# hashtags) :

> lorsqu'un utilisateur inclut des hashtags dans un commentaire, les hashtags seront ajoutés à l'image,

> Les hashtags eux-mêmes deviendront des liens vers la page d'accueil, filtrée par le hashtag donné,

- > Les hashtags les plus utilisés seront disponibles via le menu de la barre latérale.
- pour redimensionner les images une fois qu'elles ont été téléchargées avec flask-resize
- pour gérer le redimensionnement de l'image en arrière-plan avec rq ou céleri.
- pour afficher une (fenêtre) modale sur la page d'accueil au lieu de la page d'accueil.

Page d'image (picture page)



MUST HAVE - Priorité 1 - P1

- pour afficher une image donnée, son titre, sa description,

SHOULD BE - Priorité 2 - P2

- pour commenter une image (comment) : il doit être possible d'ajouter un commentaire sur la page qui affiche une image donnée.

Éléments de design communs à toutes les pages (layout page)

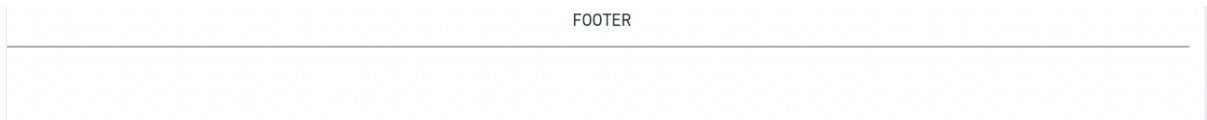
header



side-bar



footer



Notre mise en page partout dans le site doit contenir :

- Un en-tête (header) avec le nom du site, un lien vers la page d'accueil et un lien vers la page de téléchargement,
- Une barre latérale (side-bar) avec des liens pour filtrer les images par catégorie,
- Un pied de page (footer) avec une ligne de copyright.

Architecture technique

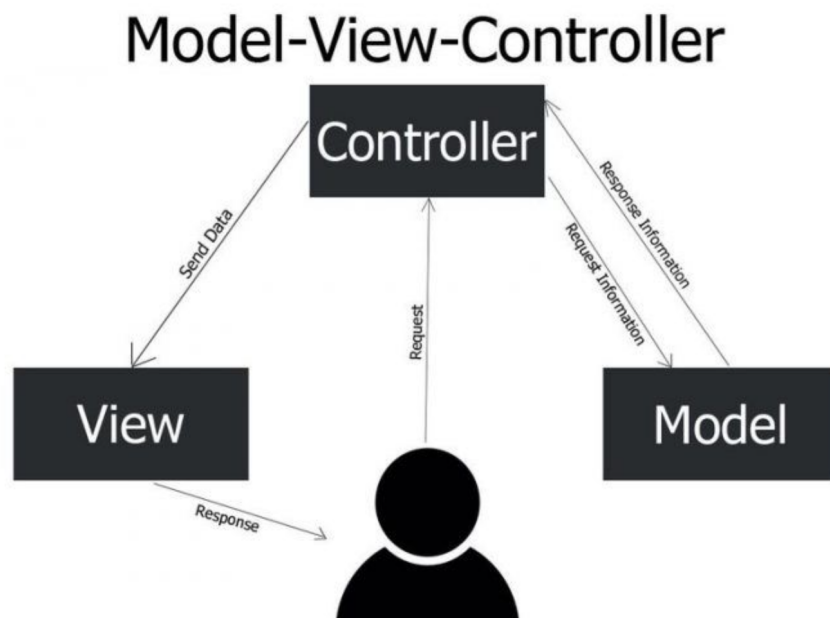
Technologies et outils

VOUS ALLEZ UTILISER :



Architecture MVC

MVC



- Modèle : créer une Base de données (database) SQLite 3, lancer des requêtes SQL sur la database avec DB Browser for SQLite.
- Vue / Contrôleur : comprendre comment utiliser Flask, pour rendre les vues Web (render_templates) et Jinja & Bootstrap ou bien directement HTML & CSS.
- Système de fichiers (OS) :

- > comprendre comment un site Web interagit avec l'OS,
- > savoir gérer les téléchargements de fichiers (uploads).

Le site web doit pouvoir fonctionner entièrement localement sans accès internet.

Architecture du site

L'architecture du site est entièrement décrite dans le fichier README.md à la racine du projet :

`sbelalo-picshare/README.md`

```
#####
Date : 2021-6-19
Projet : Picshare
Groupe : SaLeeMas
#####
Dossier Git-lab : git clone
https://gitlab.matrice.io/picshare/sbelalo-picshare
#####
Organisation du projet
#####
Dossier gestion de projet : sbelalo-picshare/trello
Dossier notes de réunions : sbelalo-picshare/notes
Dossier de livrables : sbelalo-picshare/delivery
#####
Architecture du site
#####
Dossier racine : sbelalo-picshare
Fichier README de description de l'architecture du site :
sbelalo-picshare/README.md
Dossier du site : sbelalo-picshare/picshare
Dossier static : sbelalo-picshare/picshare/static
Un dossier CSS : sbelalo-picshare/picshare/static/css
Un dossier IMAGES : sbelalo-picshare/picshare/static/images
(pas celle qu'on rajoute dans le site)
Un dossier JS : sbelalo-picshare/picshare/static/js
Dossier templates : sbelalo-picshare/picshare/templates
Fichiers HTML :
sbelalo-picshare/picshare/templates/layout.html
sbelalo-picshare/picshare/templates/index.html
sbelalo-picshare/picshare/templates/upload.html
sbelalo-picshare/picshare/templates/picture.html
Dossier uploads : sbelalo-picshare/picshare/uploads
Fichiers UPLOADS : (Comprend toutes les images uploadées par les
```

```
utilisateurs)
/!\ les noms de fichiers, ne doivent pas contenir d'espace /!\
Fichiers Python :
sbelalo-picshare/picshare/run.py
sbelalo-picshare/picshare/index.py
sbelalo-picshare/picshare/upload.py
sbelalo-picshare/picshare/picture.py
Fichiers Database :
sbelalo-picshare/picshare/init_db.py
Un fichier de dépendances aux bibliothèques employées dans le projet :
sbelalo-picshare/picshare/requirements.txt
```

Prérequis techniques

, Les prérequis techniques pour démarrer le développement du projet sont entièrement décrits dans le fichier README.md à la racine du projet :
`sbelalo-picshare/README.md`

```
#####
Environnement Virtuel
#####
TeKa:
Installation de virtualenv :
>>> pip3 install virtualenv
Création de l'environnement virtuel env :
>>> virtualenv -p python3 virt-env
Activation de virt-env :
>>> source virt-env/bin/activate
désactivation de virt-env :
>>> deactivate
fichier pour ignorer les fichiers de virt-env à ne pas ajouter dans le
Git :
>>> touch sbelalo-picshare/picshare/.gitignore
>>> subl sbelalo-picshare/picshare/.gitignore
# liste des fichiers à ignorer pour Git
virt-env/
#####
Flask / sbelalo-picshare/picshare/
#####
>>> pip3 install -r requirements.txt Flask
>>> export FLASK_APP=app.py
>>> export FLASK_ENV=development
#####
```

```
requirements.txt
#####
>>> pip3 install -r requirements.txt
>>> pip3 freeze > requirements.txt
```

Rendu des vues des pages avec Flask

Décodage du tutoriel vidéo #1 de Laurie Mézard :
<picshare2-video1.mp4>

- Notre application est basée sur le moteur de template Jinja - déjà vu avec Pelican
- Le modèle Jinja est notre structure dans laquelle nous insérons nos données html.
- La bibliothèque Flask inclue la fonction `render_template()` qui renvoie une page html (ex : `'index.html'`) avec les données en paramètres (ex : `subtitle='My Subtitle'`).
- Le modèle Jinja affiche dynamiquement les données envoyées par les retours de Flask dans `{{ data_parameter }}` (ex : `{{ subtitle }}`)
- Le modèle Jinja affiche dynamiquement le contenu des sections HTML selon `{% condition littérale %}` (ex : `{% if subtitle %}... {% else %}... {% endif %}`)
- Le modèle Jinja affichera la liste des éléments en boucle avec `{% for picture in index.object_list %}`
- Le modèle Jinja héritera d'un modèle parent en utilisant dynamiquement `{% extend "layout.html" %}` : tout ce qui est partagé entre tous les modèles ira dans le modèle `"layout.html"`.