SaLeeMas - Picshare - Product requirements & needs

Purpose

Project objectives

- Use the templating system to render web views
- Understand how a website interacts with the filesystem
- Know how to handle file uploads
- Self-organise as a team
- Know how to use Flask to answer client requests.
- Know how to build a complete web application, with an actual user front-end, that is able to answer client requests over the network.
- Upgrade a previous back-end API application, used by programmers & machines, which only answered with raw data,
- into a full-fledged web application that is able to interact with actual web users - one that will answer requests with web HTML pages.
- Build a website application to share images through 3 pages:
    - The upload page that able users to upload their own pictures,
    - The index page (homepage) that able to display pictures to all other users.
    - The picture page that able to display and to comment one picture selected by a user.

(c) This project is largely inspired by 9gag ...

Project organisation

For this group project, we are using the following methodology:

- We have to push our Backlog on our repository by the end of Tuesday June the 15th. We have also to share our Backlog through Trello Web application. Our mentor will review our Backlog to make sure that we are going in the right direction.
- At the beginning of each week, you will hold a Planning meeting where we plan how the team will work during the week. We will send the planning to our mentor.
- On Fridays, we will hold a Debriefing meeting where we will discuss what worked and what didn't work in our group, being honest with ourselves. The goal it's to help us become better at organising our group work.
- We will push notes from each Debriefing on our repository.

Site Architecture

Git-lab repository: git clone https://gitlab.matrice.io/picshare/sbelalo-picshare

arborescence of directories:

- Project management: ./sbelalo/proj_man
- Delivrables : .sbelalo/proj_man/docs
- Meeting notes : .sbelalo/proj_man/docs/notes

Functional Architecture

Core functionalities

Homepage

Must have - Priority One - P1

- to display all pictures -must have - P1
- to order displayed pictures by their creation date in descending order (the most recent image is displayed first) -must have - P1
- to navigate to the picture page , when user clicks on the picture thumb -must have - P1

Should have - Priority One - P1

- to filter displayed pictures, through query parameters - should be - P2.

Picture page

- to display a given image and its title / description (when we click on an image from the homepage)

Upload page

- to upload a new picture, with a title, a category and a description.
- the category has to be chosen among a pick-list of available categories.
- Uploaded pictures must be stored in the filesystem.
- Homonym check: When two pictures have the same name the uploader should not save the image, until the user edit its name first?)
- Validation check: if the uploaded file is an image (jpeg, png, gif...) before upload it.
- Picture Comment: it must be possible to add a comment on the page that display a given image.
- Picture Comment: it must be possible to display the comment(s) on the page for a given image.

Backlog features

Core functionalities called epics

You will implement the following functionalities:

- Add picture,
- Store picture, into the file-system
- Comment picture,
- Display picture,
- User account ?
- where all images are displayed. Images will be ordered by their creation date in descending order (the most recent image is displayed first).

For each core functionality or epic the feature should be detailed into user stories.

For example "the homepage can be filtered by category, through query parameters" means that 1) you need to check the query parameters in the index route, 2) if a category is present, check that it's a valid one, 3) if a valid category is present, filter the SQL query with the specified category.

Backlog user stories

Uploaded picture

Display picture

- check the query parameters in the index route,
- check if a category is present,
- check if a catgeory is a valid,
- filter the SQL query with the specified category,
- display picture,

Technical Architecture

Big picture

MVC architecture

- Model: createSQLite 3 Database, write and execute SQL via PGAdmin / ?
- View / Controller: HTML 5, CSS 3, Flask, Jinja & Bootstrap
  - Use the templating system: Jinja & Bootstrap, to render web views or manage HTML / CSS pages directly
  - Use Flask to answer client requests
- File System:
  - Understand how a website interacts with the filesystem
  - Know how to handle file uploads

Additional guidelines

- You must have a file named `requirements.txt` at the root of your repository, that lists all the dependencies used on the project. (relevant mostly if you are doing bonuses)

- You are allowed to use a front-end library such as Bootstrap. You won't get additional points for doing so.

No matter what you use, your website must work entirely locally. Everything should work normally, even without internet access.

Finally, there's no Sentinel for this project! 🤖

That means you are free to organise your files however you wish to.

Site Architecture

Git-lab repository: git clone https://gitlab.matrice.io/picshare/sbelalo-picshare

arborescence of directories:

- root: ./sbelalo-picture
- site: ./sbelalo-picture/picshare
- static: ./sbelalo-picture/picshare/static
- css: ./sbelalo-picture/picshare/static/css
- images: ./sbelalo-picture/picshare/static/images
  - *(different from uploaded pictures)*
- js: ./sbelalo-picture/picshare/static/js
- templates : ./sbelalo-picture/picshare/templates
  - HTML files: (layout.html ....)
  - category page: ./sbelalo-picture/picshare/templates/category.html
  - home page: : ./sbelalo-picture/picshare/templates/index.html
  - layout: ./sbelalo-picture/picshare/templates/layout.html
  - page: ./sbelalo-picture/picshare/templates/page.html
  - picture page: ./sbelalo-picture/picshare/templates/picture.html
  - hash-tag: ./sbelalo-picture/picshare/templates/tag.html
  - hash-tag lists: ./sbelalo-picture/picshare/templates/taglists.html
  - hash-tags: ./sbelalo-picture/picshare/templates/tags.html
  - upload: ./sbelalo-picture/picshare/templates/upload.html
  - user account: ./sbelalo-picture/picshare/templates/user.html
- uploaded files : (All pictures that are uploaded by all users)

./sbelalo-picture/picshare/uploads

- Python files : ./sbelalo-picture/picshare/controller/run.py
- Database files : /sbelalo-picture/picshare/model/init.db
- Dependancies file: ./sbelalo-picture/picshare/requirements.txt

Tutorial: rendering pages with Flask

<DECODE VIDEO : picshare2-video1.mp4>

- Our application is based on the Jinja template engine - already seen with Pelican
- Jinja template is our structure in wich we will insert our html data.
- Flask library will include render_template() function that returns a html page (ex: 'index.html') with the data as parameters (ex: subtitle='My Subtitle').
- Jinja template will display dinamicaly data sent by flask returns inside {{ data_parameter}} (ex: {{ subtitle }})
- Jinja template will display dinamicaly content of HTML sections according to {% litteral condition %} (ex: {% if subtitle %}... {% else %}... {% endif %})
- Jinja template will inheritate from a parent template dinamicaly using the {% extends "layout.html" %}
- Jinja template will display list of items by looping with {% for picture in index.object_list %}

Website design

Your website will have to look clean & proper! Therefore, your layout must have:

- A header with the name of the site, a link to the homepage and a link to the upload page
- A footer with a copyright line
- A sidebar with links to filter images by their category

The rest is up to your creativity.

You will use templates appropriately. **Anything that's shared between all templates will go into a `layout.html` template**. Other templates will inherit from this template.

If you are unsure about how to work with templates, you can start by building a sample homepage (with a few pictures) in HTML.

Getting started

Here's a video to get you started with file uploads:

<DECODE VIDEO : picshare2-video2.mp4>

Bonuses

This is the bonus section - the one where everything is possible! 🦄

Below is a list of possible features to implement as bonuses.

Don't forget that your Backlog is a commitment. Don't aim too low, but don't aim too high either! Choose a target that is manageable and reasonable. The goal is to do as much as possible, and to do it *well*.

Available features

- Hashtags. When a user includes hashtags in a comment, the hashtags will be added to the image. The hashtags themselves will become links to the homepage, filtered by the given hashtag (hint: you might need a many-to-many relationship to store hashtags)
- The most utilised hashtags will be available through the sidebar menu.

- Use a simple Docker system to serve your website in a container. You should be able to connect to the website normally through your browser.
- Add an “export” button to a filtered category page, to export all the images from that category as a PDF file. Use `reportlab` to generate the PDF.
- Add a voting system, where users can give a mark (from 0 to 5 stars) to each image. You will add a form to leave a vote on the “show” page of an image, similarly to how comments work. You will display the average score of an image, and the total number of voters will be displayed when we hover the average score.
- Use AJAX to handle the voting system
- Use `flask-resize` to resize images once they have been uploaded
- Use `rq` or `celery` to handle image resizing in the background
- Transform the upload page into a modal displayed onto the homepage.