

DAM: Sharing static media with Falcon API

Applications for mobile devices & Course 2019-2020

Jordi Mateo Fornés jordi.mateo@udl.cat



- Dr. Jordi Mateo Fornés
- **Office:**
 - Office A.12 (Campus Igualada)
 - Office 3.08 (EPS Lleida)
- **Email:** jordi.mateo@udl.cat
- **Twitter:** <https://twitter.com/MatForJordi>
- **Github:** <https://github.com/JordiMateoUdL>



- **Applications for mobile devices.**
- Grau en Tècniques d'Interacció Digital i de Computació
- Campus Igualada - Escola Politècnica Superior - Universitat de Lleida
- All the code developed in this course can be found in this repository: DAM Course.

Agenda

- Storing static content in sqlalchemy models.
- Serving static content as result of API requests.
- Store static content using the API requests.

Serving Static Contents

Implementation

- Build a container **aimed** to serve *static contents*.
- We are going to use **NGINX**. More information NGINX
- We are going to use this server to exchange (upload/store and retrieve) pictures.

Services:

- Backend using Gunicorn to deploy Falcon API, port 8000.
- NGINX to serve static content, port 8001.
- MySqlServer, port 3306.

Dockerfile

```
# docker/docker-compose.yml
static-file-server:
  image: nginx:latest
  ports:
    - 8001:80
  volumes:
    - ../../static/./usr/share/nginx/html/static
```

Notice: You will need to create the *static* folder at the correct level. In this folder, *../../static/*, we will create our static resources structure to serve the requests. We are coping this folder into the default nginx static folder in the container */usr/share/nginx/html/static*.

Notice: We redirect the *container* port **80** (nginx default) to *localhost* **8001**.

Dockerfile (2)

```
# docker/docker-compose.yml
backend:
  build: "./backend"
  ports:
    - "8000:8000"
  environment:
    - DAMCore_DB_HOST=mysql_db_container
  volumes:
    - ../../damcore/./app
    - ../../static/./static
```

Notice: We are sharing static folder between containers.

- We are going to create a rule to store our static content (images), using this pattern:

```
${host}${static_folder}${media_folder}\  
${resource-url}${image_filename}
```

- Example: We want the picture of the user

```
http://localhost:8001/static/media/users/2/photo/image.png
```

Pattern Implementation: URL generation (1)

```
def _generate_media_url(class_instance,
    class_attribute_name, default_image=False):
    class_base_url = urljoin(
        urljoin(urljoin("http://{}"
            .format(
                settings.STATIC_HOSTNAME),
                settings.STATIC_URL),
                settings.MEDIA_PREFIX),
        class_instance.__tablename__ + "/"
    )
    class_attribute = getattr(class_instance,
        class_attribute_name)
```

Pattern Implementation: URL generation (2)

```
if class_attribute is not None:
    return urljoin(urljoin(urljoin(urljoin(
        class_base_urlclass_attribute),
        str(class_instance.id) + "/"),
        class_attribute_name + "/"),
        class_attribute)
else:
    if default_image:
        return urljoin(urljoin(class_base_url,
            class_attribute_name + "/"), settings.DEFAULT_IMAGE_NAME)
    else:
        return class_attribute
```

Pattern Implementation: PATH generation

```
def _generate_media_path(class_instance, class_attribute_name):  
    class_path = "{0}/{1}{2}/{3}/{4}/".format(  
        settings.STATIC_URL,  
        settings.MEDIA_PREFIX,  
        class_instance.__tablename__,  
        str(class_instance.id),  
        class_attribute_name)  
    return class_path
```

Update User model (models.py)

```
class User(SQLAlchemyBase, JSONModel):  
    __tablename__ = "users"  
    photo = Column(Unicode(255))  
  
    @hybrid_property  
    def photo_url(self):  
        return _generate_media_url(self, "photo")  
  
    @hybrid_property  
    def photo_path(self):  
        return _generate_media_path(self, "photo")
```

GET

```
application.add_route("/account/profile",  
account_resources.ResourceAccountUserProfile())
```

#POST

```
application.add_route("/account/profile/update_profile_image",  
account_resources.ResourceAccountUpdateProfileImage())
```

Just ensure that in `json_model` in `models.py`:

```
@hybrid_property
def json_model(self):
    return {
        ...
        "photo": self.photo_url
    }
```

POST (1)

```
@falcon.before(requires_auth)
class ResourceAccountUpdateProfileImage(DAMCoreResource):
    def on_post(self, req, resp, *args, **kwargs):
        super(ResourceAccountUpdateProfileImage, self).on_post(req,
            # Get the user from the token
            current_user = req.context["auth_user"]
            resource_path = current_user.photo_path
            # Get the file from form
            incoming_file = req.get_param("image_file")
```


POST (2)

```
# Run the common part for storing
filename = utils
.save_static_media_file(incoming_file, resource_path)
# Update db model
current_user.photo = filename
self.db_session.add(current_user)
self.db_session.commit()
```

save_static_media_file (utils.py) (1)

```
def save_static_media_file(incoming_file, resource_path):  
    # Generate and id with the currrent timestamp  
    imgId = str(int(datetime.datetime.now().timestamp() * 1000))  
    # Build filename using the id generated  
    filename = imgId + "." + incoming_file.filename.split(".")[1]  
    # Check if folder exists or not in the server  
    if not os.path.exists(resource_path):  
        os.makedirs(resource_path)
```

save_static_media_file (utils.py) (2)

```
# Create a file path
file_path = resource_path + filename
# Write to a temporary file to prevent
# incomplete files from being used
temp_file_path = file_path + "~"
with open(temp_file_path, "wb") as f:
    f.write(incoming_file.file.read())
# File has been fully saved to disk move it into place
os.rename(temp_file_path, file_path)
return filename
```

Testing (1)

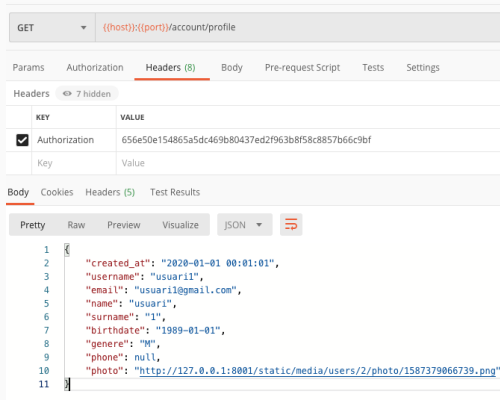


Figure 1: Screenshot from Postman, operation `getAccountProfile()`

Testing (2)


POST ▼ `{{host}}:{{port}}/account/profile/update_profile_image`

Params Authorization Headers (10) **Body** ● Pre-request Script Tests Settings

☐ none ☒ form-data ☐ x-www-form-urlencoded ☐ raw ☐ binary ☐ GraphQL

KEY	VALUE
<input checked="" type="checkbox"/> image_file	<input type="text" value="cepa.png"/> ✕
Key	Value

Body Cookies Headers (5) Test Results

Pretty Raw Preview Visualize JSON ▼ 

1 |

Figure 2: Screenshot from Postman, operation `updateAccountProfile()`

Testing(3)

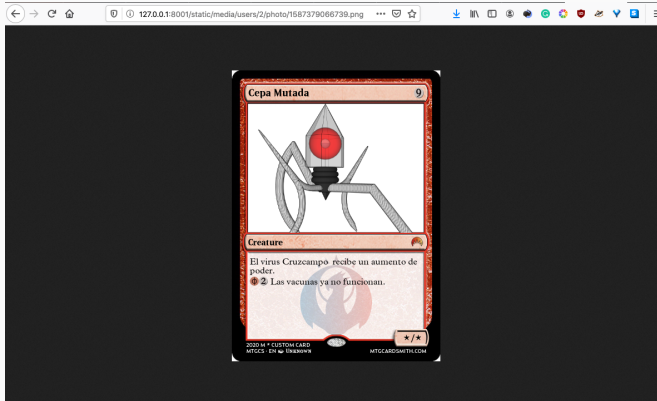


Figure 3: Screenshot from browser, getting the uploaded image

That is all

Well done! Thanks for your attendance! Questions?

www — jordimateofoernes.com

github — github.com/JordiMateo

twitter — @MatForJordi

gdc — Distributed computation group