DA: Rest API (Tokens and Authoritzation)

Applications for mobile devices - Theory - Unit 5

Didac Florensa Cazorla

Any: 2021-2022
Curs: 102386
Institut: University of Lleida (Campus Igualada)
Titulació: Bachelor’s degree in Digital Interaction and Computing Techniques (GTIDIC)

Agenda

Introduction

Warm-up

- 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/JordiMateoUdL

- Ph.D Dídac Florensa Cazorla
- **Office**:
  - Office A.12 (Campus Igualada)
  - Office 3.08 (EPS Lleida)
- **Email**: didac.florensa@udl.cat

Users

CRUD for the Users

- CRUD
  - **Register**: An anonymous user can register on our platform and create an account.
    - username, mail, name, surname, genre, password.
  - **Read**: A user can check their user-values related to their account.
  - **Update**: A user can update account parameters (optional and mandatory).
  - **Delete**: A user can delete their account of the system.
- A user account must contain a public user profile that can check all the users authenticated in the system and a private profile that can only be read by their own user.
- A user must be authenticated in the system to manage their account (perform read, updated and delete actions).

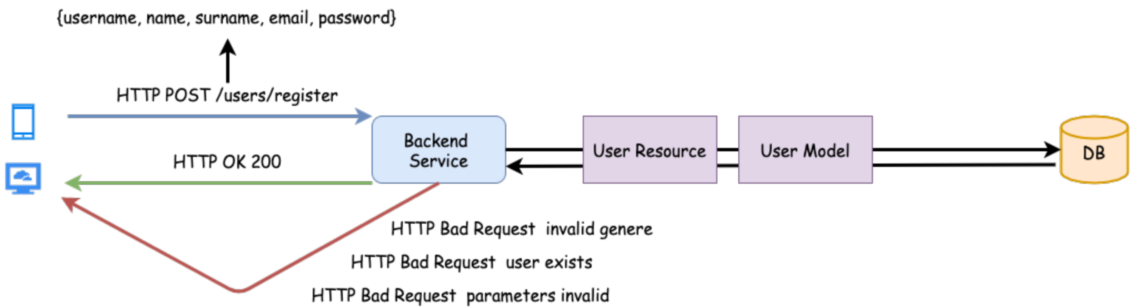
User Profile (Public)

```
class User(SQLAlchemyBase, JSONModel):
    ...
    @hybrid_property
    def public_profile(self):
      return {
        "created_at":
        self.created_at.strftime(settings.DATETIME_DEFAULT_FORMAT),
        "username": self.username,
        "genere": self.genere.value,
        "photo": self.photo,
      }
```

User Profile (Private)

```
class User(SQLAlchemyBase, JSONModel):
    @hybrid_property
    def json_model(self):
     return {
        "created_at":
        self.created_at.strftime(settings.DATETIME_DEFAULT_FORMAT),
        "username": self.username,"email": self.email,
        "name": self.name,"surname": self.surname,
            "birthdate": self.birthdate.strftime(
               settings.DATE_DEFAULT_FORMAT)
            if self.birthdate is not None else self.birthdate,
        "genere": self.genere.value, "phone": self.phone,
        "photo": self.photo_url }
```

Register (Create an account)


{username, name, surname, email, password}
HTTP POST /users/register
HTTP OK 200
Backend Service
User Resource
User Model
DB
HTTP Bad Request invalid genere
HTTP Bad Request user exists
HTTP Bad Request parameters invalid

- **app.py** -> application.add_route("/users/register", user_resources.ResourceRegisterUser())
- **resources/user_resource.py** -> ResourceRegisterUser() class

Register (ResourceRegisterUser)

```
class ResourceRegisterUser(DAMCoreResource):
  @jsonschema.validate(SchemaRegisterUser)
  def on_post(self, req, resp, *args, **kwargs):
    super(ResourceRegisterUser, self)
        .on_post(req, resp, *args, **kwargs)
        aux_user = User()
        try:
            try:
                aux_genere = GenereEnum(req.media["genere"].upper())
            except ValueError:
                raise falcon.HTTPBadRequest(
                    description=messages.genere_invalid)
```

Register (ResourceRegisterUser)

```
class ResourceRegisterUser(DAMCoreResource):
  @jsonschema.validate(SchemaRegisterUser)
  def on_post(self, req, resp, *args, **kwargs):
    super(ResourceRegisterUser, self)
        .on_post(req, resp, *args, **kwargs)
        ...
            aux_user.username = req.media["username"]
            aux_user.password = req.media["password"]
            aux_user.email = req.media["email"]
            aux_user.name = req.media["name"]
            aux_user.surname = req.media["surname"]
            aux_user.genere = aux_genere
```

Register (ResourceRegisterUser)

```
class ResourceRegisterUser(DAMCoreResource):
  def on_post(self, req, resp, *args, **kwargs):
    ...
        self.db_session.add(aux_user)
            try:
                self.db_session.commit()
            except IntegrityError:
                raise falcon.HTTPBadRequest(
                    description=messages.user_exists)
        except KeyError:
            raise falcon.HTTPBadRequest(
                description=messages.parameters_invalid)
        resp.status = falcon.HTTP_200
```

Validation schema

Important to introduce schema validation to helps keeping your API well-defined and ensure that your elements are in sync between API, DB and clients.

Register (Schema)

```python
# schemas.py
SchemaRegisterUser = {
    "type": "object",
    "properties": {
        "username": {"type": "string"},
        "password": {"type": "string"},
        "email": {"type": "string"},
        "name": {"type": "string"},
        "surname": {"type": "string"},
        "genere": {"type": "string"}
    },
    "required": ["username", "password", "email",
    "name", "surname", "genere"]}
```

Sample (example)

```
curl --location --request POST '127.0.0.1:8000/users/register' \
--header 'Content-Type: application/json' \
--data-raw '{
    "username": "mat.jor",
    "password": "4321",
    "email": "jmateo@diei.udl.cat",
    "name": "jor",
    "surname": "mat",
    "genere": "M"
}'
```

- Or we can use postman ^^.

DamCoreResource()

```
class DAMCoreResource(object):
    def __print_request(self, request):
        mylogger.debug("New request {} {}?{} from host: {}"
        .format(request.method, request.path,
        request.query_string,request.access_route))
    def __init__(self): self.db_session = None
    def on_get(self, req, resp, *args, **kwargs):
        self.__print_request(req)
    def on_post(self, req, resp, *args, **kwargs):
         self.__print_request(req)
    def on_put(self, req, resp, *args, **kwargs):
        self.__print_request(req)
    def on_head(self, req, resp, *args, **kwargs):
                                 self.__print_request(req)
```

JSONModel()

```
class JSONModel(object):
    # Indicates that is an abstract class
    __metaclass__ = abc.ABCMeta
    def _create_json_model(self, **attributes):
        ...
    @abc.abstractmethod
    def json_model(self):
        pass
    def to_json_model(self, **attributes):
        return self._create_json_model(**attributes)
```

create_json_model()

```
final_model = dict()
try:
    for current_key in attributes.keys():
        aux_attribute = getattr(self, attributes[current_key])
        if isinstance(aux_attribute, JSONModel)
        and aux_attribute is not None:
            final_model[current_key] = aux_attribute.json_model
        elif isinstance(aux_attribute, datetime.datetime):
            final_model[current_key] =
            aux_attribute.strftime(DATETIME_DEFAULT_FORMAT)
        elif isinstance(aux_attribute, datetime.date):
            final_model[current_key] =
            aux_attribute.strftime(DATE_DEFAULT_FORMAT)
```

create_json_model()

```
            elif isinstance(aux_attribute, datetime.time):
                final_model[current_key] = aux_attribute.strftime(TIME_DEFAULT_FORMAT)
            else:
                final_model[current_key] = aux_attribute
        return final_model
    except KeyError as e:
        raise falcon.HTTPInternalServerError(description=str(e))
```

Activity 01: Updating user model

Activity 01: Updating user model

Work in groups:

1. fork and clone the repository.
2. Create a new enum to model different roles (premium, freemium) users.
3. The role must be mandatory.
4. Update the database script to create 10 users premium and 10 users freemium.
5. Make a pull request to my repository with these changes.

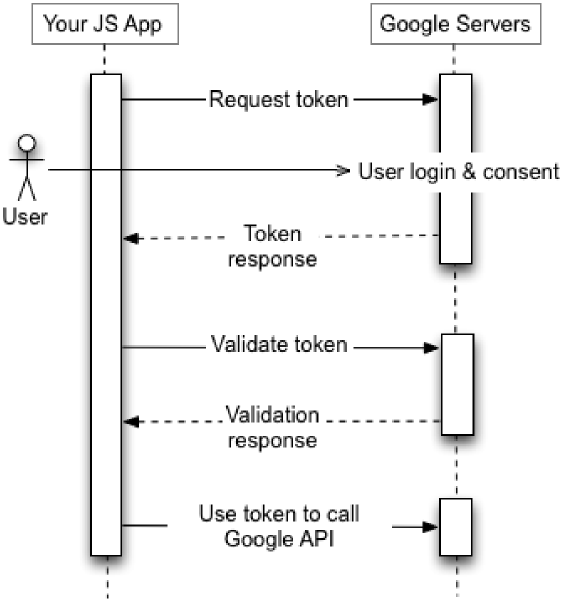
How to make a pull request:
https://www.freecodecamp.org/news/how-to-make-your-first-pull-request-on-github-3/

Campus Universitari Igualada - UdL

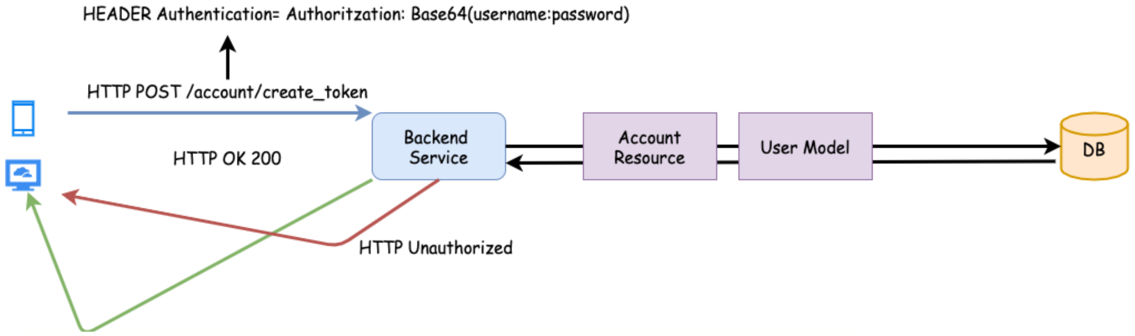
Universitat de Lleida

User Authentication (Login)

Tokens


Your JS App
Google Servers
Request token
User
User login & consent
Token response
Validate token
Validation response
Use token to call Google API

Implementation


HEADER Authentication= Authoritzation: Base64(username:password)
HTTP POST /account/create_token
HTTP OK 200
Backend Service
Account Resource
User Model
DB
HTTP Unauthorized

```
{
"token":
"27b85cd005ec2483435a4b527279bf543a208731caca930b43"
}
```

User token

```python
class UserToken(SQLAlchemyBase):
    __tablename__ = "users_tokens"
    id = Column(Integer, primary_key=True)
    token = Column(Unicode(50), nullable=False, unique=True)
    user_id = Column(Integer, ForeignKey("users.id",
    onupdate="CASCADE", ondelete="CASCADE"), nullable=False)
    user = relationship("User", back_populates="tokens")
```

create_token()

```
class User(SQLAlchemyBase, JSONModel):
    __tablename__ = "users"
    _
    ...
    @hybrid_method
    def create_token(self):
        if len(self.tokens) < settings.MAX_USER_TOKENS:
            token_string =
            binascii.hexlify(os.urandom(25)).decode("utf-8")
            aux_token = UserToken(token=token_string, user=self)
            return aux_token
        else: raise falcon.HTTPBadRequest(
                title=messages.quota_exceded,
                description=messages.maximum_tokens_exceded)
```

ResourceCreateUserToken()

```
class ResourceCreateUserToken(DAMCoreResource):
    def on_post(self, req, resp, *args, **kwargs):
        super(ResourceCreateUserToken, self)
        .on_post(req, resp, *args, **kwargs)
        basic_auth_raw = req.get_header("Authorization")
        if basic_auth_raw is not None:
            basic_auth = basic_auth_raw.split()[1]
            auth_username, auth_password =
                (base64.b64decode(basic_auth)
                    .decode("utf-8").split(":"))
            if (auth_username is None) or (auth_password is None)
                or (auth_username == "") or (auth_password == ""):
                raise falcon.HTTPUnauthorized(
                    description=messages.username_and_password_required)
```

ResourceCreateUserToken()

```
        else:
            raise falcon.HTTPUnauthorized(
                description=messages.authorization_header_required)
        current_user = self.db_session.query(User)
            .filter(User.email == auth_username).one_or_none()
        if current_user is None:
            current_user = self.db_session.query(User)
            .filter(User.username == auth_username).one_or_none()
```

ResourceCreateUserToken()

```
        if (current_user is not None)
             and (current_user.check_password(auth_password)):
            current_token = current_user.create_token()
            try:
                self.db_session.commit()
                resp.media = {"token": current_token.token}
                resp.status = falcon.HTTP_200
            except Exception as e:
                mylogger.critical("{}:{}".format(
                    messages.error_saving_user_token, e))
                self.db_session.rollback()
                raise falcon.HTTPInternalServerError()
        else:raise falcon.HTTPUnauthorized(
                                    description=messages.user_not_found)
```

Sample (example)

```
curl
--location
--request POST '127.0.0.1:8000/account/create_token' \
--header 'Content-Type: application/json' \
--header 'Authorization: dXNlcjI6cjQ1dGd0'
```

- Or we can use postman ^^.

Using Auth

Show Private Account

```
# GET /account/profile
@falcon.before(requires_auth)
class ResourceAccountUserProfile(DAMCoreResource):
    def on_get(self, req, resp, *args, **kwargs):
        super(ResourceAccountUserProfile, self)
            .on_get(req, resp, *args, **kwargs)
        current_user = req.context["auth_user"]
        resp.media = current_user.json_model
        resp.status = falcon.HTTP_200
```

Auth Hook

```
def requires_auth(req, resp, resource, params):
    auth_token = req.get_header("Authorization")
    if auth_token is not None:
        current_token = resource.db_session.query(UserToken)
        .filter(UserToken.token == auth_token).one_or_none()
        if current_token is not None:
            req.context["auth_user_token"] = current_token
            req.context["auth_user"] = current_token.user
        else:
            raise falcon.HTTPUnauthorized(
                description=messages.token_invalid)
    else: raise falcon.HTTPUnauthorized(
                        description=messages.token_required)
```

Show Private Account (example)

```
curl --location
--request GET '127.0.0.1:8000/account/profile' \
--header 'Authorization:
3b29bc475096e9ab556556c62e5b3c1db55f1b59f0dd24ee71'
```

- Or we can use postman ^^.

That's all

QUESTIONS?

About me

www — jordimateofornes.com
github — github.com/JordiMateo
twitter — @MatForJordi
gdc — Distributed computation group