Backend Fent País

Iskra Desenvolupament

October 27, 2022

Contents

1 Introducció

Ens han encarregat renovar el backend de Fent País, i els nostre arquitecte ens ha recomanat fer servir Guillotina. Fent País és un servei que permet fer reserves de capses regals. Una capsa regal està formada per N experiències, l'usuari pot triar una d'aquestes experiències.

2 Entorn de treball

Farem servir un entorn virtual de python, necessitem python3, des del directori on hi ha **setup.py**. També heu de tenir docker instal·lat. **INSTAL·LEU** docker.

```
cd backend
pytho3 -m venv venv
source venv/bin/activate
pip install -r requirements.txt
pip install -r requirements-test.txt
pip install -e .
```

Listing 1: Entorn virtual

Si teniu un error a l'hora d'instal·lar els requeriments, probablement haureu d'actuallitzar pip

```
pip install --upgrade pip
```

Listing 2: Entorn virtual

Això instal·larà les dependències necessàries i el paquet de fentpais en mode desenvolupament. El codi està tot fet, només haureu d'anar descomentant les línies.

3 Breu Explicació de traversal i guillotina

Guillotina és un traversal. És com un arbre o similar al sistema de fitxers UNIX, organitzats en carpetes i paths. En el cas de guillotina, cada url mapeja un objecte diferent.

Guillotina té l'estructua bàsica següent:

```
db/container/{objecte}
```

db: La base de dades

container: La primera gran carpeta del sistema de tipus IContainer. Sempre n'hi ha d'haver un com a mínim

3.1 Arquitectura

L'arquitectura més basica que fa servir guillotina és:

1. **Postgresql**: Base de dades relacional
2. **Redis**: Base de dades key value en memòria

En el nostre cas, la postgres ens servirà de base de dades i de catàleg, que vol dir que podrem indexar els nostres objectes per cercar-los ràpidament. Podríem tenir un ElasticSearch també, però en aquest cas farem servir la mateixa postgres.

4 Requeriments del sistema

4.1 Traversal

1. Creació de la carpeta **db/container/tipus_capsa** quan es crea el container
2. Creació de la carpeta **db/container/reserves** quan es crea el container
3. Creació de l'endpoint @getExperiencies en el context de la carpeta tipus_capsa **db/container/tipus_capsa/@getExperiencies**. Aquest endpoint ha de retornar totes les experiències de tots els tipus de capsa
4. Creació de l'endpoint @getExperiencies en el context de la carpeta foo_tipus_capsa **db/container/tipus_cpasa/foo_tipus_capsa/@getExperiencies**. Aquest endpoint ha de retornar totes les experiències només de la capsa foo_tipus_capsa
5. Quan es crea una reserva, comprovar que el tipus de capsa existeix, si no existeix anul·lar la reserva i retornar un **412**
6. Un usuari/client només podrà crear reserves. No podrà afegir tipus_capsa ni experiències, però sí que les podrà veure
7. Només un usuari root podrà crear tipus de capsa i experiencies
8. El sistema ha de tenir tests i testejar totes les funcionalitats

5 Disseny

5.1 Traversal

Tenint en compte els requeriments del sistema, tindrem una estructura de dades i de traversal com la següent.

```
/db/container/tipus_capsa/foo_tipus_capsa/foo_experiencia_1
/db/container/tipus_capsa/foo_tipus_capsa/foo_experiencia_N
/db/container/tipus_capsa/foo_tipus_capsa/@getExperiencies
/db/container/tipus_capsa/@getExperiencies
/db/container/tipus_capsa/foo_tipus_capsa_N/foo_experiencia_N
/db/container/tipus_capsa/reserves/foo_reserva_1
/db/container/tipus_capsa/reserves/foo_reserva_N
/db/container/tipus_capsa/users/foo_user
```

Listing 3: Estructura dades

Reserves i tipus_capsa són carpetes, i dintre d'aquestes carpetes hi haurà objectes de tipus reserva i experiència respectivament. La carpeta users contindrà objectes de tipus User.

5.2 Interfícies

Les interfícies defineixen els camps dels objectes. També es poden definir mètodes o corutines, sense la seva implementació. Les interfícies ja estan definides dintre de la carpeta **interfaces**

5.2.1 Reserves: interfaces/reserves.py

Definim les interfícies per la carpeta reserves i el tipus reserva. Veim que la reserva només té un camp, el tipus_capsa, que serà l'id d'un objecte tipus_capsa que referenciarà la capsa.

```
from guillotina import interfaces, schema

class IReservesFolder(interfaces.IFolder):
  pass


class IReserva(interfaces.IItem):
  tipus_capsa = schema.TextLine()

```

Listing 4: Estructura dades

5.2.2 Experiencies: interfaces/experiencia.py

```
class IExperiencia(interfaces.IItem):
  index_field("nom", type="text")
  nom = schema.TextLine(required=True)

  index_field("description", type="text")
  descripcio = schema.TextLine(required=True)

  categories = schema.List(
      title="Categories",
      description="Categories de la experiencia",
      value_type=schema.TextLine(),
      required=False
  )
```

```
14
```

Listing 5: Estructura dades

Experiència tindrà un nom, una descripció i una llista de categories. El mètode index_field indexa aquest camp al catàleg, en aquest cas a la Postgres

El següent codi indexa el camp categories, que és una llista, com un string de paraules. Normalment, sempre que indexem dades més complexes, les serialitzem

```
1   @directives.index_field.with_accessor(
2     IExperiencia,
3     "categories",
4     type="text",
5     field="categories",
6     store=True
7     )
8     async def index_tests_respondre(obj):
9        results = ""
10    for category in obj.categories:
11       results = results + category
12       if results != "":
13           return results
14
```

Listing 6: Estructura dades

@directives no és una interfície com a tal, podríem definir-la en un altre fitxer, però en aquest cas ho he fet en el mateix fitxer **experiencia.py**

5.2.3 Experiencies: interfaces/tipus_capsa.py

```
1   from guillotina import interfaces, schema
2
3
4   class ITipusCapsaFolder(interfaces.IFolder):
5       async def get_experiencies():
6           """
7           Get all experiences
8           """
9
10
11  class ITipusCapsa(interfaces.IFolder):
12      async def get_experiencies():
13          """
14          Get experiences of the capsa
15          """
16
17      nom = schema.Text(required=True)
18
19      descripcio = schema.TextLine(required=True)
20
21      preu = schema.Float(required=True)
22
```

Listing 7: Estructura dades

Tipus capsa té un nom, una descripció i un preu. També definim la interfície de dos corutines, get_experiencies pels dos objectes.

6 Desenvolupament

En aquesta secció anirem descomentant el codi, i provant les diferents funcionalitats.

6.1 Definició del container

Primer definirem quins tipus hi poden haver dintre del container. Com hem vist en els requeriments, les carpetes que podrem afegir són ReservesFolder i TipusCapsaFolder. Per fer-ho descomentarem les línies que hi ha entre el comentari **TODO_1** i **END_TODO_1** del fitxer **container.py**. El codi que descomentareu fa servir la directiva **@configure.contenttype** per configurar el tipus, el camp **allowed_types**. El camp behaviors ens permet tenir camps extres que ja venen configurats pel mateix behavior. Aquest és una part del codi que descomentareu.

```
@configure.contenttype(
    type_name="Container",
    schema=IContainer,
    behaviors=[
        'guillotina.behaviors.dublincore.IDublinCore'
    ],
    allowed_types=["ReservesFolder", "TipusCapsaFolder"]
)
class Container(Container):
  pass
```

Listing 8: Definició container

6.2 Creació de les carpetes reserves i tipus_capsa

Començarem creant les dues carpetes reserves i tipus_capsa automàticament quan es crea el container. Per fer això descomentarem les línies que hi ha entre el comentari **TODO_2** i **END_TODO_2** del fitxer **container.py**

Si us hi fixeu, hi ha un decorador configurant un subscriber:

```
  @configure.subscriber(for_=(IContainer, IObjectAddedEvent))
```

Listing 9: Subscriber container

Això fa executar la corutina **creacio_container** sempre que un objecte de tipus IContainer es crea.

Les línies que heu de descomentar el que fan és crear la carpeta amb id reserves del tipus ReservesFolder. Finalment notifiquem que hem creat una carpeta de tipus ReservesFolder, perquè altres subscribers es puguin beneficiar de la creació de la carpeta.

```
  reserves_folder = await create_content_in_container(
        parent=obj,
        type_="ReservesFolder",
        check_security=False,
        id_="reserves",
        title="Reserves' folder"
    )
    await notify(ObjectAddedEvent(reserves_folder, reserves_folder, reserves_folder.
    id))
```

Listing 10: Creació reserves folder

El següent codi crea la carpeta tipus_capsa del tipus TipusCapsaFolder

```
  tipus_capsa_folder = await create_content_in_container(
        parent=obj,
        type_="TipusCapsaFolder",
        check_security=False,
```

```
        id_="tipus_capsa",
        title="Tipus capses' folder"
    )
    await notify(ObjectAddedEvent(tipus_capsa_folder, tipus_capsa_folder,
    tipus_capsa_folder.id))

```

Listing 11: Creació tipus_capsa

Ara en el fitxer **content/reserves.py**, hem de definir el content de tipus ReservesFolder. Per fer això descomentem el que hi ha entre **TODO_3** i **END_TODO_3**

```
  @configure.contenttype(
    type_name="ReservesFolder",
    schema=IReservesFolder,
    behaviors=[
        'guillotina.behaviors.dublincore.IDublinCore'
    ],
    allowed_types=["Reserva"],
    globally_addable=False
)
class ReservesFolder(content.Folder):
    async def get_reserves(self):
        search_instance = query_utility(ICatalogUtility)
        return search_instance.search({"type_name": "Reserva"})

```

Listing 12: Configuració content ReservesFolder

Ara en el fitxer **content/tipus_capsa.py**, hem de definir el content de tipus TipusCapsaFolder. Per fer això descomentem el que hi ha entre **TODO_4** i **END_TODO_4**

```
    @configure.contenttype(
    type_name="TipusCapsaFolder",
    schema=ITipusCapsaFolder,
    behaviors=[
        'guillotina.behaviors.dublincore.IDublinCore'
    ],
    allowed_types=["TipusCapsa"],
    globally_addable=False
    )
    class TipusCapsaFolder(content.Folder):
    async def get_experiencies(self):
        search_instance = query_utility(ICatalogUtility)
        return await search_instance.search(
            context=self,
            query={"type_name": "Experiencia"}
)
```

Listing 13: Configuració content ReservesFolder

Aquest codi defineix el tipus TipusCapsaFolder, i també implementa la corutina get_experiences, que fent servir el cataleg, fa una cerca per retornar totes les experiències de totes les capses

6.2.1 Testing

Un cop descomentades les línies, podem passar el test que comprova que ho hem fet correctament, en el directori de la carpeta backend/fentpais, correm pytest:

```
    DATABASE=pg pytest tests/test_container.py

```

Listing 14: Testing 1

Aquest test crea un container i comprova que les carpetes s'han creat correctament. Si passa el test, felicitats, les carpetes reserves i tipus_capsa han estat creades en la creació del container. Si no és així, torneu a mirar les línies que estan dintre dels blocs TODO_1, TODO_2, TODO_3 i TODO_4 container.py, reserves.py i tipus_capsa.py respectivament.

6.3 Creació d'una capsa dintre de la carpeta tipus_capsa

En el fitxer **content/tipus_capsa.py**, hem de definir el content de tipus TipusCapsa. Per fer això descomentem el que hi ha entre **TODO_5** i **END_TODO_5**

```
@configure.contenttype(
    type_name="TipusCapsa",
    schema=ITipusCapsa,
    allowed_types=["Experiencia"],
    behaviors=[
        'guillotina.behaviors.dublincore.IDublinCore'
    ],
    globally_addable=False
)
class TipusCapsa(content.Folder):
    async def get_experiencies(self):
        search_instance = query_utility(ICatalogUtility)
        return await search_instance.search(
            context=self,
            query={"type_name": "Experiencia", "metadata": "*"}
)
```

Listing 15: Definició content type TipusCapsa

Aquest codi defineix el tipus TipusCapsa, i també implementa la corutina get_experiences, que fent servir el cataleg, fa una cerca per retornar les experiències que hi ha dintre de la capsa

6.3.1 Testing

Un cop descomentades les línies, podem passar el test que comprova que ho hem fet correctament, en el directori de la carpeta backend/fentpais, correm pytest:

```
    DATABASE=pg pytest tests/test_capsa.py -k test_creacio_capsa

```

Listing 16: Testing 2

6.4 Creació d'una experiència dintre d'una capsa

En el fitxer **content/experiencia.py**, hem de definir el content de tipus TipusCapsa. Per fer això descomentem el que hi ha entre **TODO_6** i **END_TODO_6**

```
@configure.contenttype(
    type_name="Experiencia",
    schema=IExperiencia,
    behaviors=[
        'guillotina.behaviors.dublincore.IDublinCore'
    ],
    globally_addable=False
)
class Experiencia(content.Item):
    pass
```

Listing 17: Definició content type Experiencia

6.4.1 Testing

Un cop descomentades les línies, podem passar el test que comprova que ho hem fet correctament, en el directori de la carpeta backend/fentpais, correm pytest:

```
    DATABASE=pg pytest tests/test_capsa.py -k test_creacio_experiencia

```

Listing 18: Testing 3

6.5 Creació dels endpoints @getExperiencies

En el fitxer **content/tipus_capsa.py**, hem de definir els endpoints **@getExperiencies** descomentant les línies de codi entre els comentaris **TODO_7** i **END_TODO_7**

```
@configure.service(context=ITipusCapsa, name="@getExperiences", method="GET",
    permission="guillotina.ViewContent")
@configure.service(context=ITipusCapsaFolder, name="@getExperiences", method="GET",
    permission="guillotina.ViewContent")
class GetExperiencesCapsa(Service):
    async def __call__(self):
        return await self.context.get_experiencies()
```

Listing 19: Definició endpoints

El **@configure.service** defineix els endpoints per dos contexts diferents, un per la carpeta tipus_capsa i capsa. Fixeu-vos en un detall, self.context pot ser un objecte de TipusCapsaFolder o TipusCapsa, com que hem implementat les dues corutines amb el mateix nom, podem aprofitar la mateixa crida get_experiencies. Mireu també el permission: **guillotina.ViewContent**, només podrà cridar aquest endpoint qui tingui aquest permís.

6.5.1 Testing

Un cop descomentades les línies, podem passar el test que comprova que ho hem fet correctament, en el directori de la carpeta backend/fentpais, correm pytest:

```
  DATABASE=pg pytest tests/test_capsa.py -k
    test_tipus_capsa_get_experiencies_endpoint
```

Listing 20: Testing 4

6.6 Creació de les reserves

En el fitxer **content/reserves.py**, hem de definir el content Reserva, descomentant el codi entre **TODO_8** i **END_TODO_8**

```
@configure.contenttype(
    type_name="Reserva",
    schema=IReserva,
    behaviors=[
        'guillotina.behaviors.dublincore.IDublinCore'
    ],
    globally_addable=False
)
class Reserva(content.Item):
    async def check_tipus_capsa(self):
        search_instance = query_utility(ICatalogUtility)
        container = get_current_container()
        tipus_capsa_folder = await container.async_get("tipus_capsa")
```

```
        results = await search_instance.unrestrictedSearch(context=tipus_capsa_folder
    , query={"id": self.tipus_capsa})
        if results["items_total"] == 0:
            raise HTTPPreconditionFailed()
```

Listing 21: Definició content type Reserva

En el codi configurem el contingut Reserva, i creem un mètode que comprova que el camp tipus_capsa, que és un id que referencia a una capsa, existeix. Si no existeix retornem un 412. Guillotina és transaccional, si aixequem una excepció, la transacció s'anul·la, i l'objecte no es crearà

6.6.1 Testing

Si ho hem fet bé podrem passar el test:

```
DATABASE=pg pytest tests/test_reserva.py -k test_creacio_experiencia_correcte
```

Listing 22: Testing 5

En el fitxer **content/reserves.py**, ara definirem el subscriber que comprova que quan es crea o es modifica una reserva, el tipus_capsa és corecte. Descomentem el codi entre **TODO_9** i **END_TODO_9**

```
@configure.subscriber(for_=(IReserva, IObjectAddedEvent))
@configure.subscriber(for_=(IReserva, IObjectModifiedEvent))
async def creacio_modificacio_reserva(obj, event):
    await obj.check_tipus_capsa()

```

Listing 23: Definició subscriber Reserva

6.6.2 Testing

Si ho hem fet bé podrem passar el test:

```
DATABASE=pg pytest tests/test_reserva.py -k test_creacio_experiencia_412
```

Listing 24: Testing 6

6.7 Permisos usuari

Qui pot fer què? és una pregunta molt important a l'hora de definir un projecte. En aquest cas, un usuari logejat pot fer:

1. **POT FER** GET de la carpeta tipus_capsa, per veure totes les capses de la carpeta
2. **NO POT FER** GET de la carpeta reserves, no pot veure el llistat de reserves
3. **POT FER** GET de la seva reserva
4. **POT FER** GET d'una capsa
5. **POT FER** GET d'una experiència

En el nostre cas, l'usuari tindrà el ROL **guillotina.Member**, un rol consta de N **permisos**. Els permisos bàsics són:

1. guillotina.AccessContent. És l'heterònim de la comanda ls en UNIX. No permet fer GET del propi objecte, però permet navegar entre la carpeta. Per exemple, si tenim aquest permís a la carpeta reserves db/container/reserves/la_meva_reserva, i tenim el permís guillotina.ViewContent en la_meva_reserva Podrem fer un GET de la_meva_reserva, però no podrem fer un GET de reserves.

2. guillotina.ViewContent. És l'heterònim de la comanda cd en UNIX. Podrem fer un GET del recurs

3. guillotina.AddContent. Podem fER POST i crear contingut a la carpeta.

Començarem donant permís de guillotina.AccessContent al container, tots els usuaris que tinguin el rol: guillotina.Member, per fer això descomentarem el codi entre **TODO_10** i **END_TODO_10**, del fitxer **content/container.py**. Penseu també que podem fer que els permisos assignats a carpetes, s'heredin entre els seus fills.

```
manage_roles_instance = IRolePermissionManager(obj)
    manage_roles_instance.grant_permission_to_role_no_inherit("guillotina.
    AccessContent", "guillotina.Member")
    manage_roles_instance.grant_permission_to_role("guillotina.AccessContent", "
    guillotina.Editor")
    manage_roles_instance.grant_permission_to_role("guillotina.ViewContent", "
    guillotina.Editor")

```

Listing 25: Permisos container

El mètode grant_permission_to_role_no_inherit només dona permís a la carpeta, sense que els seus fils puguin heredar el permís del pare, en canvi grant_permission_to_role li dona el permís al recurs i a tots els seus fills.

Fem el mateix per la carpeta reserves, diem que l'usuari pugui afegir contingut a la carpeta, i que pugui accedir-hi, però que no pugui fer GET de la carpeta, ja que no li donem el permís guillotina.ViewContent

```
manage_roles_instance = IRolePermissionManager(reserves_folder)
manage_roles_instance.grant_permission_to_role_no_inherit("guillotina.AccessContent",
     "guillotina.Member")
manage_roles_instance.grant_permission_to_role_no_inherit("guillotina.AddContent", "
    guillotina.Member")
```

Listing 26: Permisos reserves

En canvi, per la carpeta tipus_capsa, no deixem afegir contingut, però sí que pugui fer GET de la carpeta per veure tots les capses. Fixeu-vos que el mètode que fem servir és grant_permission_to_role, que farà que també puguin fer GET de les experiències de dintre de les capses, ja que els permisos s'hereden entre els seus fills

```
manage_roles_instance = IRolePermissionManager(tipus_capsa_folder)
manage_roles_instance.grant_permission_to_role("guillotina.AccessContent", "
    guillotina.Member")
manage_roles_instance.grant_permission_to_role("guillotina.ViewContent", "guillotina.
    Member")
```

Listing 27: Permisos tipus_capsa

6.7.1 Testing

Si ho hem fet bé podrem passar el test:

```
  DATABASE=pg pytest tests/test_roles.py
```

Listing 28: Testing 7

7 Evaluation

We did some experiments ...

8 Conclusions and Future Work

From our experiments we can conclude that ...