

Backend Fent País

Iskra Desenvolupament

October 27, 2022

Contents

1	Introducció	1
2	Entorn de treball	2
3	Breu Explicació de traversal i guillotina	2
3.1	Arquitectura	2
4	Requeriments del sistema	2
4.1	Traversal	2
5	Disseny	4
5.1	Traversal	4
5.2	Interfícies	4
5.2.1	Reserves: interfaces/reserves.py	4
5.2.2	Experiències: interfaces/experiencia.py	4
5.2.3	Experiències: interfaces/tipus_capsa.py	5
6	Desenvolupament	6
6.1	Definició del container	6
6.2	Creació de les carpetes reserves i tipus_capsa	6
6.3	Creació d'una capsa dintre de la carpeta tipus_capsa	8
6.4	Creació d'una experiència dintre d'una capsa	8
6.5	Creació dels endpoints @getExperiències	9
6.6	Creació de les reserves	9
7	Evaluation	11
8	Conclusions and Future Work	12

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.

```

1 python3 -m venv venv
2 source venv/bin/activate
3 pip install -r requirements.txt
4 pip install -r requirements-test.txt
5 pip install -e .

```

Listing 1: 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'estructura bàsica següent:

```

1 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 bàsica 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 **db/container/tipus_capsa/getExperiencies**. Aquest endpoint ha de retornar totes les experiències de tots els tipus de capsas
4. Creació de l'endpoint **@getExperiencies** en el context de la carpeta **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 experiències
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.

```

1 /db/container/tipus_capsa/foo_tipus_capsa/foo_experiencia_1
2 /db/container/tipus_capsa/foo_tipus_capsa/foo_experiencia_N
3 /db/container/tipus_capsa/foo_tipus_capsa/@getExperiencias
4 /db/container/tipus_capsa/@getExperiencias
5 /db/container/tipus_capsa/foo_tipus_capsa_N/foo_experiencia_N
6 /db/container/tipus_capsa/reserves/foo_reserva_1
7 /db/container/tipus_capsa/reserves/foo_reserva_N
8 /db/container/tipus_capsa/users/foo_user

```

Listing 2: 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.

```

1 from guillotina import interfaces, schema
2
3 class IReservesFolder(interfaces.IFolder):
4     pass
5
6
7 class IReserva(interfaces.IItem):
8     tipus_capsa = schema.TextLine()
9

```

Listing 3: Estructura dades

5.2.2 Experiencias: interfaces/experiencia.py

```

1 class IExperiencia(interfaces.IItem):
2     index_field("nom", type="text")
3     nom = schema.TextLine(required=True)
4
5     index_field("description", type="text")
6     descriptio = schema.TextLine(required=True)
7
8     categories = schema.List(
9         title="Categories",
10        description="Categories de la experiencia",
11        value_type=schema.TextLine(),
12        required=False
13    )

```

Listing 4: 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 5: 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 6: 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 podem 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`, fent 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.

```

1 @configure.contenttype(
2     type_name="Container",
3     schema=IContainer,
4     behaviors=[
5         'guillotina.behaviors.dublincore.IDublinCore'
6     ],
7     allowed_types=["ReservesFolder", "TipusCapsaFolder"]
8 )
9 class Container(Container):
10     pass

```

Listing 7: 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:

```

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

```

Listing 8: 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 subscriptors es puguin beneficiar de la creació de la carpeta.

```

1 reserves_folder = await create_content_in_container(
2     parent=obj,
3     type_="ReservesFolder",
4     check_security=False,
5     id_="reserves",
6     title="Reserves' folder"
7 )
8 await notify(ObjectAddedEvent(reserves_folder, reserves_folder, reserves_folder.
9     id))

```

Listing 9: Creació reserves folder

El següent codi crea la carpeta tipus_capsa del tipus `TipusCapsaFolder`

```

1 tipus_capsa_folder = await create_content_in_container(
2     parent=obj,
3     type_="TipusCapsaFolder",
4     check_security=False,
5     id_="tipus_capsa",

```

```

6         title="Tipus capsas' folder"
7     )
8     await notify(ObjectAddedEvent(tipus_capsa_folder, tipus_capsa_folder,
9         tipus_capsa_folder.id))

```

Listing 10: 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**

```

1  @configure.contenttype(
2      type_name="ReservesFolder",
3      schema=IReservesFolder,
4      behaviors=[
5          'guillotina.behaviors.dublincore.IDublinCore'
6      ],
7      allowed_types=["Reserva"],
8      globally_addable=False
9  )
10 class ReservesFolder(content.Folder):
11     async def get_reserves(self):
12         search_instance = query_utility(ICatalogUtility)
13         return search_instance.search({"type_name": "Reserva"})
14

```

Listing 11: 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**

```

1  @configure.contenttype(
2      type_name="TipusCapsaFolder",
3      schema=ITipusCapsaFolder,
4      behaviors=[
5          'guillotina.behaviors.dublincore.IDublinCore'
6      ],
7      allowed_types=["TipusCapsa"],
8      globally_addable=False
9  )
10 class TipusCapsaFolder(content.Folder):
11     async def get_experiencies(self):
12         search_instance = query_utility(ICatalogUtility)
13         return await search_instance.search(
14             context=self,
15             query={"type_name": "Experiencia"}
16 )

```

Listing 12: Configuració content ReservesFolder

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

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`:

```

1  DATABASE=pg pytest tests/test_container.py
2

```

Listing 13: 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**

```

1 @configure.contenttype(
2     type_name="TipusCapsa",
3     schema=ITipusCapsa,
4     allowed_types=["Experiencia"],
5     behaviors=[
6         'guillotina.behaviors.dublincore.IDublinCore'
7     ],
8     globally_addable=False
9 )
10 class TipusCapsa(content.Folder):
11     async def get_experiencies(self):
12         search_instance = query_utility(ICatalogUtility)
13         return await search_instance.search(
14             context=self,
15             query={"type_name": "Experiencia", "metadata": "*"}
16 )

```

Listing 14: Definició content type TipusCapsa

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

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:

```

1 DATABASE=pg pytest tests/test_capsa.py -k test_creacio_capsa
2

```

Listing 15: 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**

```

1 @configure.contenttype(
2     type_name="Experiencia",
3     schema=IExperiencia,
4     behaviors=[
5         'guillotina.behaviors.dublincore.IDublinCore'
6     ],
7     globally_addable=False
8 )
9 class Experiencia(content.Item):
10     pass

```

Listing 16: Definició content type Experiencia

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:

```

1 DATABASE=pg pytest tests/test_capsa.py -k test_creacio_experiencia
2

```

Listing 17: 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`

```

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

Listing 18: Definició endpoints

El `@configure.service` defineix els endpoints per dos contextos 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.

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`:

```

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

Listing 19: 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`

```

1 @configure.contenttype(
2     type_name="Reserva",
3     schema=IReserva,
4     behaviors=[
5         'guillotina.behaviors.dublincore.IDublinCore'
6     ],
7     globally_addable=False
8 )
9 class Reserva(content.Item):
10     async def check_tipus_capsa(self):
11         search_instance = query_utility(ICatalogUtility)
12         container = get_current_container()
13         tipus_capsa_folder = await container.async_get("tipus_capsa")
14         results = await search_instance.unrestrictedSearch(context=tipus_capsa_folder
15         , query={"id": self.tipus_capsa})
16         if results["items_total"] == 0:
17             raise HTTPPreconditionFailed()
```

Listing 20: 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

Si ho hem fet bé podrem passar el test:

```

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

Listing 21: 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 correcte. Descomentem el codi entre **TODO_9** i **END_TODO_9**

```
1 @configure.subscriber(for_=(IReserva, IObjectAddedEvent))
2 @configure.subscriber(for_=(IReserva, IObjectModifiedEvent))
3 async def creacio_modificacio_reserva(obj, event):
4     await obj.check_tipus_capsa()
5
```

Listing 22: Definició subscriber Reserva

Si ho hem fet bé podrem passar el test:

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

Listing 23: Testing 6

7 Evaluation

We did some experiments ...

8 Conclusions and Future Work

From our experiments we can conclude that ...