# Projecte SDAMV2 M6.UF2

# "BOCCATO DI CARDINALE"

Amb aquest projecte pretenem implementar un programa fet amb Python que ens permeti fer la gestió bàsica d'una base de dades de botigues, restaurants i altres establiments on es pot trobar bon beure i bon menjar. La base de dades que utilitzarem és MongoDB, i les operacions de gestió les durem a terme mitjançant l'eina ODM anomenada MongoEngine. Aquesta eina ODM ens facilitarà molt la transformació d'objectes de Python a documents de MongoDB, i a l'inrevés.

L'objectiu és aplicar, en un cas pràctic, aspectes relatius a l'estructuració de dades en el model documental (posant una atenció especial en les qüestions relatives a la incrustació i referenciació d'objectes) i a la programació d'operacions CRUD sobre MongoDB, des de Python, mitjançant l'ús d'una eina ODM (Object-Document Mapping).

# BOCCATO DI CARDINALE – FASE 1

En aquesta primera fase, ens centrarem a preparar totes les eines que necessitem per al desenvolupament del projecte i a adquirir els coneixements bàsics sobre el funcionament de MongoEngine, l'eina ODM que utilitzarem per a la gestió de les dades.

## F1.1

### INSTAL·LACIÓ DE LES EINES NECESSÀRIES

Per al desenvolupament de les diverses fases d'aquest projecte caldrà tenir instal·lades i funcionant correctament les eines següents:

- El llenguatge de programació Python, amb la biblioteca PyMongo instal·lada.
- Un entorn de desenvolupament (PyCharm o Visual Studio Code).
- Un servidor MongoDB, amb els clients MongoDB Shell i MongoDB Compass.

Tot això són coses que ja hauríeu de tenir si heu anat seguint les sessions de classe. A més, per aquest projecte caldrà instal·lar específicament el package MongoEngine, que és la implementació de l'eina ODM (Object-Document Mapper) que farem servir. Per instal·lar aquest mòdul en Python cal, simplement, executar la comanda següent:

```
> pip install mongoengine
```

# F1.2

## PRIMERES PASSES AMB MongoEngine

Per començar a aprendre com funcionen els mecanismes de persistència d'objectes sobre MongoDB mitjançant MongoEngine, veurem com gestionar una petita part de la base de dades. Concretament, veurem com fer la gestió dels municipis.

Una de les coses que voldrem tenir representades a la base de dades per cadascun dels establiments, és el municipi on es troben. La informació sobre els municipis la voldrem tenir en una collection independent de la base de dades, ja que preveiem que es requereixi poder-los consultar de forma directa amb una relativa freqüència.

Quan fem servir una eina de mapatge objecte-relacional o objecte-documental, partirem sempre de la definició de les classes de la nostra aplicació. Per tant, per començar, definirem una classe anomenada `Municipi` que serà la que representarà aquest concepte. Vegem-ne primerament el codi:

```python
import mongoengine as me

# Classe Municipi que representarà aquest concepte en la nostra aplicació.
class Municipi(me.Document):
    id = me.SequenceField(primary_key=True)
    codi_postal = me.StringField(max_length=5, min_length=5, regex="\\d{5}")
    nom = me.StringField(max_length=128, required=True, unique_with="codi_postal")
```

Amb aquest primer exemple, ja podem veure algunes qüestions que cal tenir presents quan treballem amb MongoEngine:

- Cal fer la importació del package `mongoengine` per poder utilitzar totes les classes, funcions, directives, etc., que ens proporciona per gestionar la persistència d'objectes.
- Tota classe que generi objectes que vulguem que puguin ser guardats a la base de dades, ha de ser subclasse directa o indirecta de la classe `Document` de `mongoengine`.
- Els atributs de les classes persistents s'han de declarar com a atributs de classe i s'han de definir les seves característiques utilitzant les declaracions pròpies de `mongoengine` (els anomenats **field objects**). Els tipus més habituals són:
  - **`StringField`**: atributs de tipus cadena de caràcters.
  - **`BooleanField`**: atributs de tipus booleà.
  - **`DateTimeField`**: atributs de tipus data/hora.
  - **`FloatField`**: atributs de tipus numèric real.

  - **`IntFiled` / `LongField`**: atributs de tipus numèric enter, amb una capacitat de 32 i 64 bits, respectivament.
  - **`DictField`**: atributs de tipus diccionari.
  - **`ReferenceField`**: atributs de tipus referència a objectes d'altres collections (similar a les foreign keys del model relacional).
  - **`ListField`**: atributs de tipus llista de valors/objectes.
  - **`SequenceField`**: atributs que permetran emular el concepte de camp auto-numèric o auto-incremental típic de les bases de dades relacionals.

  Podeu veure la llista completa de tipus d'atributs que podem incloure en les classes persistents de `mongoengine` a la documentació oficial:

  http://docs.mongoengine.org/guide/defining-documents.html#fields

- En la declaració de cadascun dels atributs es pot indicar una sèrie de paràmetres que permeten configurar algunes qüestions relatives a la seva validació i a la seva representació sobre la base de dades:
  - **`max_length` / `min_length`**: longituds màxima i mínima, respectivament, del contingut d'un atribut.
  - **`required`**: indica si un atribut pot valer **`None`** (`required=`**`False`**) o no (`required=`**`True`**). Si un atribut no es defineix, equival a que el seu valor sigui **`None`**.
  - **`unique`**: indica si l'atribut pot tenir valors repetits en diversos objectes de la collection (`unique=`**`False`**), o no (`unique=`**`True`**).
  - **`primary_key`**: indica si l'atribut fa les funcions de clau primària en els objectes de la collection. Si definim un atribut amb `primary_key=`**`True`**, implica automàticament (per tant, no cal escriure-ho de forma explícita) que l'atribut sigui `required` i `unique`. A més, un atribut definit com a `primary_key`, tindrà el nom `_id` a la base de dades (independentment del nom que li hàgim donat a nivell d'aplicació) i substituirà l'identificador d'objecte que genera MongoDB de forma automàtica.
  - **`regex`**: permet indicar el format que haurà de tenir el valor d'un atribut mitjançant el mecanisme de les expressions regulars.
  - **`unique_with`**: permet indicar que volem que un conjunt format per dos o més atributs estiguin obligats a tenir valors diferents. És a dir, cadascun dels atributs referenciats en un `unique_with` podrà tenir valors repetits per ell mateix però, en conjunt, hauran de tenir valors únics.

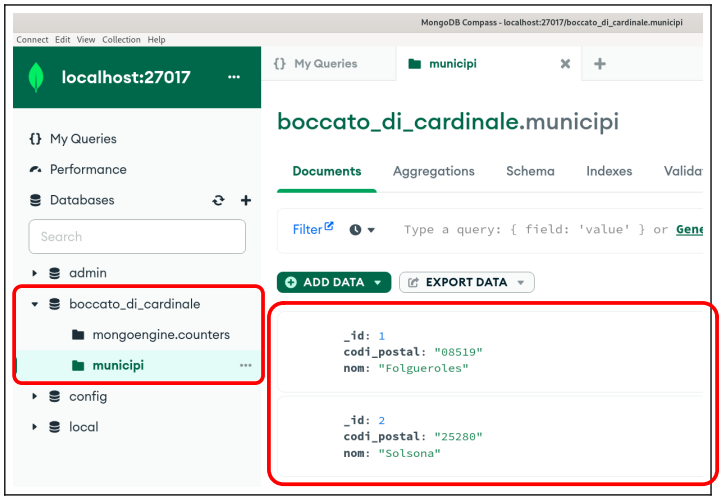
Podeu consultar la llista completa de paràmetres que es poden utilitzar per declarar els atributs a la documentació oficial:

http://docs.mongoengine.org/guide/defining-documents.html#field-arguments

Per fer una primera prova per comprovar com podem guardar objectes de tipus `Municipi` a la base de dades, proveu d'executar el següent programa d'exemple que ens permet veure com es porten a terme les **insercions**:

```python
import mongoengine as me

# Classe Municipi que representarà aquest concepte en la nostra aplicació.
class Municipi(me.Document):
    id = me.SequenceField(primary_key=True)
    codi_postal = me.StringField(max_length=5, min_length=5, regex="\\d{5}")
    nom = me.StringField(max_length=128, required=True, unique_with="codi_postal")

if __name__ == "__main__":

    # Connexió sobre la base de dades "boccato_di_cardinale". Si no existeix
    # aquesta base de dades, es crearà.
    me.connect(host="mongodb://127.0.0.1:27017/boccato_di_cardinale")

    # Creem dos objectes de tipus Municipi, executant directament el
    # mètode save() que és el que fa que es guardi a la base de dades.
    m1 = Municipi(
        nom="Folgueroles",
        codi_postal="08519"
    ).save()

    m2 = Municipi(
        nom="Solsona",
        codi_postal="25280"
    ).save()

    # Tanquem la connexió.
    me.disconnect()
```

Si ens connectem a MongoDB via MongoDB Compass després d'executar aquest programa per primera vegada, veurem que s'haurà creat la base de dades anomenada `boccato_di_cardinale`, que conté una collection anomenada `municipi` i que aquesta collection conté els dos objectes que hem fet persistents executant el mètode `save()`:



També veiem una collection anomenada `mongoengine.counters` que és la que es crea de forma automàtica per mantenir els comptadors associats als camps de tipus `SequenceField` que tinguem.

Si volem que la collection s'anomeni `municipis`, en comptes de `municipi`, ho podem indicar afegint un atribut especial anomenat `meta` a la classe `Municipi`. Aquest atribut ens permet especificar diverses opcions respecte de la configuració de les collections. Per indicar quin nom concret volem que tingui la collection que guardarà un determinat tipus d'objectes ho farem com es pot veure en l'exemple següent:

```
import mongoengine as me

# Classe Municipi que representarà aquest concepte en la nostra aplicació.
class Municipi(me.Document):
    id = me.SequenceField(primary_key=True)
    codi_postal = me.StringField(max_length=5, min_length=5, regex="\\d{5}")
    nom = me.StringField(max_length=128, required=True, unique_with="codi_postal")
    meta = {"collection": "municipis"}
...
```

Un cop ja tenim alguns objectes a la base de dades, passem a veure com podem fer algunes **consultes** senzilles sobre una collection:

```
import mongoengine as me

# Classe Municipi que representarà aquest concepte en la nostra aplicació.
class Municipi(me.Document):
    id = me.SequenceField(primary_key=True)
    codi_postal = me.StringField(max_length=5, min_length=5, regex="\\d{5}")
    nom = me.StringField(max_length=128, required=True, unique_with="codi_postal")
    meta = {"collection": "municipis"}

    def __str__(self):
        return "%-5s - %-80s" % (self.codi_postal, self.nom)

if __name__ == "__main__":

    # Connexió sobre la base de dades "boccato_di_cardinale".
    db = me.connect(host="mongodb://127.0.0.1:27017/boccato_di_cardinale")

    # Consultem tots els municipis de la base de dades.
    municipis = Municipi.objects()

    print("TOTS ELS MUNICIPIS:\n")
    for municipi in municipis:
        print(municipi)

    # Consultem el municipi que té un codi postal concret.
    municipi = Municipi.objects(codi_postal="08506")
    print("\n\nMUNICIPI AMB CODI POSTAL 08506:\n")
    print(municipi[0])

    # Consultem tots els municipis de la província de Barcelona, ordenats per nom.
    municipis = Municipi.objects(codi_postal__startswith="08").order_by("nom")

    print("\n\nMUNICIPIS DE LA PROVÍNCIA DE BARCELONA ORDENATS ALFABÈTICAMENT:\n")
    for municipi in municipis:
        print(municipi)

    # Tanquem la connexió.
    me.disconnect()
```

Fixeu-vos com, pel simple fet d'executar el mètode `objects()` de la classe `Municipi` (mètode que heretem de la classe `Document`), obtenim una llista amb tots el objectes de tipus `Municipi` que tenim a la base de dades i, a partir d'aquí, podem aplicar filtres, ordenacions, etc. Es poden consultar moltes més possibilitats de realitzar consultes a la documentació oficial:

http://docs.mongoengine.org/guide/querying.html

Les **eliminacions** d'objectes també són molt senzilles de fer. Simplement, seleccionarem l'objecte o objectes que vulguem eliminar amb el mètode `objects()` com si volguéssim fer una consulta i, sobre el resultat obtingut, executarem el mètode `delete()`. El mètode `delete()` ens retornarà com a resultat un nombre enter que indica quants objectes han estat eliminats:

```
1  import mongoengine as me
2
3  # Classe Municipi que representarà aquest concepte en la nostra aplicació.
4  class Municipi(me.Document):
5      id = me.SequenceField(primary_key=True)
6      codi_postal = me.StringField(max_length=5, min_length=5, regex="\\d{5}")
7      nom = me.StringField(max_length=128, required=True, unique_with="codi_postal")
8      meta = {"collection": "municipis"}
9
10     def __str__(self):
11         return "%-5s - %-80s" % (self.codi_postal, self.nom)
12
13 if __name__ == "__main__":
14
15     # Connexió sobre la base de dades "boccato_di_cardinale".
16     db = me.connect(host="mongodb://127.0.0.1:27017/boccato_di_cardinale")
17
18     # Eliminem un municipi. Ho fem seleccionant el municipi a eliminar per codi postal.
19     eliminats = Municipi.objects(codi_postal="43737").delete()
20     print("Quantitat de municipis eliminats: %d" % eliminats)
21
22     # Eliminem tots els municipis de la província de Barcelona.
23     eliminats = Municipi.objects(codi_postal__startswith="08").delete()
24     print("Quantitat de municipis eliminats: %d" % eliminats)
25
26     me.disconnect()
```

I, per acabar aquest apartat, vegem com podem portar a terme les **actualitzacions** o modificacions d'objectes. Aquesta operació es pot fer mitjançant l'ús de diversos mètodes de la classe `QuerySet` –el tipus d'objecte que ens retorna el mètode `objects()`– o de la classe `Document` –cadascun dels objectes que conté un `QuerySet`–.

Si sabem que la modificació que volem fer ha d'afectar a un únic objecte, podem utilitzar el mètode `update_one()` de la classe `QuerySet` o el mètode `modify()` de la classe `Document`. Si es tracta d'una modificació que previsiblement afecti a diversos objectes, haurem d'utilitzar el mètode `update()` o el mètode `modify()` de la classe `QuerySet`. Vegem-ne un exemple del primer cas, del cas en què vulguem fer una modificació sobre un únic objecte:

```python
import mongoengine as me

# Classe Municipi que representarà aquest concepte en la nostra aplicació.
class Municipi(me.Document):
    id = me.SequenceField(primary_key=True)
    codi_postal = me.StringField(max_length=5, min_length=5, regex="\\d{5}")
    nom = me.StringField(max_length=128, required=True, unique_with="codi_postal")
    meta = {"collection": "municipis"}

    def __str__(self):
        return "%-5s - %-80s" % (self.codi_postal, self.nom)

if __name__ == "__main__":

    # Connexió sobre la base de dades "boccato_di_cardinale".
    db = me.connect(host="mongodb://127.0.0.1:27017/boccato_di_cardinale")

    # Consultem el municipi abans de modificar-lo.
    municipis = Municipi.objects(codi_postal="43540")
    print("Abans de la modificació: %s" % municipis[0])

    # Fem la modificació.
    resultat = Municipi.objects(codi_postal="43540").update_one(nom="La Ràpita")

    # Si s'ha fet la modificació, mostrem el municipi actualitzat.
    if resultat:
        print("Després de la modificació: %s" % municipis[0].reload())

    me.disconnect()
```

**RF.01** Un cop aconseguiu que funcionin correctament tots aquests petits programes que permeten fer les operacions bàsiques amb els objectes de tipus `Municipi`, aviseu el professor perquè us faci la revisió del funcionament amb tots els elements que hem proposat fins ara.

# BOCCATO DI CARDINALE – FASE 2

En aquesta segona fase, aprofundirem en els mecanismes que ens proporciona MongoEngine per modelar la informació que volem guardar a la base de dades i donarem una estructura consistent i completa a les dades de la nostra aplicació.

## F2.1

### DESCRIPCIÓ DE LES DADES I DE LA SEVA ESTRUCTURA (I)

La base de dades de l'aplicació que hem de desenvolupar ha de permetre representar la informació d'establiments on es pot trobar bon beure i bon menjar. Inicialment, volem que es pugui tenir informació sobre botigues i restaurants, però no es descarta afegir-hi altres tipus d'establiments. La informació que volem tenir registrada de tots els tipus d'establiments és la següent:

- **Nom**: nom de l'establiment, de caràcter obligatori i que no admet repetits.
- **Domicili**: ubicació de l'establiment descrita en forma de carrer, plaça, etc. i número, de caràcter obligatori.
- **Municipi**: indicació del municipi on es troba l'establiment. Es tracta d'una informació descrita a través d'un tipus d'objecte compost pel nom del municipi i el seu codi postal. És una dada obligatòria.
- **Localitat**: si el lloc on es troba l'establiment és un nucli que no forma un municipi per ell mateix o no és cap de municipi, registrarem el seu nom. És una informació opcional.
- **Coordenades**: coordenades geogràfiques de la ubicació de l'establiment per facilitar-ne la localització en sistemes de geoposicionament.
- **Lloc web**: si l'establiment disposa de pàgina web pròpia, registrarem el seu URL en aquest camp. És de caràcter opcional.
- **Especialitats**: es tracta d'una llista o array de cadenes de caràcters que ens ha de permetre indicar quines són les especialitats de l'establiment (el seus productes estrella).

- **Observacions**: es tracta d'un camp que ens ha de permetre registrar un text lliure per poder deixar constància d'aspectes que es cregui necessari explicar sobre l'establiment. És de caràcter opcional.

- **Actiu**: camp booleà que ens ha de permetre indicar si l'establiment es troba actiu, o no. La idea és poder mantenir registrats a la base de dades els establiments que ja hagin clausurat la seva activitat per poder facilitar consultes històriques.

- **Telèfons**: es tracta d'una llista o array de cadenes de caràcters que ens ha de permetre indicar tots els telèfons de contacte que té l'establiment.

- **Emails**: llista o array de cadenes de caràcters per poder guardar correus electrònics de contacte de l'establiment.

Per definir l'estructura de la classe `Establiment` haurem de decidir si volem que el municipi sigui un objecte incrustat a cadascun dels objectes de tipus `Establiment` que tinguem, o bé si volem que sigui un objecte referenciat (és a dir, que existeixi una collection de municipis separada i que des dels objectes `Establiment` tinguem una referència cap al municipi que correspongui, a l'estil de les claus foranes del model relacional). Optarem per aquesta segona opció, ja que volem que els municipis puguin ser consultats de forma directa sense haver d'obtenir prèviament tots els establiments que tinguem. Vegem com quedaria la definició de la classe `Establiment`:

```
1  class Establiment(me.Document):
2      nom = me.StringField(max_length=128, required=True, unique=True)
3      coordenades = me.PointField(required=True)
4      domicili = me.StringField(max_length=256, required=True)
5      municipi = me.ReferenceField(Municipi)
6      localitat = me.StringField(max_length=128)
7      lloc_web = me.URLField()
8      especialitats = me.ListField(me.StringField(max_length=64, required=True))
9      observacions = me.StringField(max_length=256)
10     actiu = me.BooleanField(required=True, default=True)
11     telefons = me.ListField(me.StringField(max_length=32, required=True))
12     emails = me.ListField(me.StringField(max_length=64, required=True))
13     meta = {"collection": "establiments"}
```

Fixem-nos en les línies de codi que tenim ressaltades de color groc, que inclouen elements nous respecte del que havíem vist fins ara:

- **Atributs booleans**: veiem a la línia 10 que podem crear aquest tipus de camps amb la declaració `BooleanField` de MongoEngine.

- **Atributs URL**: veiem a la línia 7 que MongoEngine ens permet definir camps destinats a emmagatzemar aquest tipus d'element amb la declaració `URLField`.

- **Atributs de tipus punt geogràfic**: a la línia 3 veiem un camp de tipus `PointField`. És el tipus de MongoEngine que permet definir un camp mitjançant el qual es poden representar coordenades geogràfiques per expressar una ubicació a través d'un valor de longitud i un valor de latitud.

- **Atributs de tipus llista o array**: la base de dades MongoDB permet definir atributs de tipus llista o array per guardar conjunts de valors associats a un únic camp d'un objecte (el mateix concepte d'array que ens proporciona JSON). Els definim amb la declaració `ListField` que ha de contenir, entre el seus parèntesis, la declaració del tipus de valor que hi haurà en cadascun dels seus elements. En el nostre cas, a la línia 8, 11 i 12, estem definint, respectivament, els atributs anomenats `especialitats` , `telefons` i `emails` com a arrays o llistes de cadenes de caràcters.

- **Atributs de tipus referència**: a la línia 5 veiem com definim l'atribut `municipi` com una referència a un objecte de tipus `Municipi` (el que hem comentat a la fase anterior). La referència des de cada objecte `Establiment` cap al seu objecte `Municipi` es farà a través de l'atribut `_id` del municipi (en el nostre cas, l'identificador auto-numèric perquè és el que hem declarat com a primary key).

Un cop definida la classe `Establiment`, provarem de crear alguns objectes d'aquest tipus i els guardarem a la base de dades amb el mètode `save()`. Abans d'executar aquest nou programa, elimineu la base de dades sencera que teníeu creada com a resultat d'haver realitzat les proves que hem fet anteriorment.

```python
import mongoengine as me

# Classe Municipi que representarà aquest concepte en la nostra aplicació.
class Municipi(me.Document):
    id = me.SequenceField(primary_key=True)
    codi_postal = me.StringField(max_length=5, min_length=5, regex="\\d{5}")
    nom = me.StringField(max_length=128, required=True, unique_with="codi_postal")
    meta = {"collection": "municipis"}

# Classe Establiment que representarà aquest concepte en la nostra aplicació.
class Establiment(me.Document):
    nom = me.StringField(max_length=128, required=True, unique=True)
    coordenades = me.PointField(required=True)
    domicili = me.StringField(max_length=256, required=True)
    municipi = me.ReferenceField(Municipi)
    localitat = me.StringField(max_length=128)
    lloc_web = me.URLField()
    especialitats = me.ListField(me.StringField(max_length=64, required=True))
    observacions = me.StringField(max_length=256)
    actiu = me.BooleanField(required=True, default=True)
    telefons = me.ListField(me.StringField(max_length=32, required=True))
    emails = me.ListField(me.StringField(max_length=64, required=True))
    meta = {"collection": "establiments"}

if __name__ == "__main__":
    # Connexió sobre la base de dades "boccato_di_cardinale". Si no existeix
    # aquesta base de dades, es crearà.
    db = me.connect(host="mongodb://127.0.0.1:27017/boccato_di_cardinale")
```

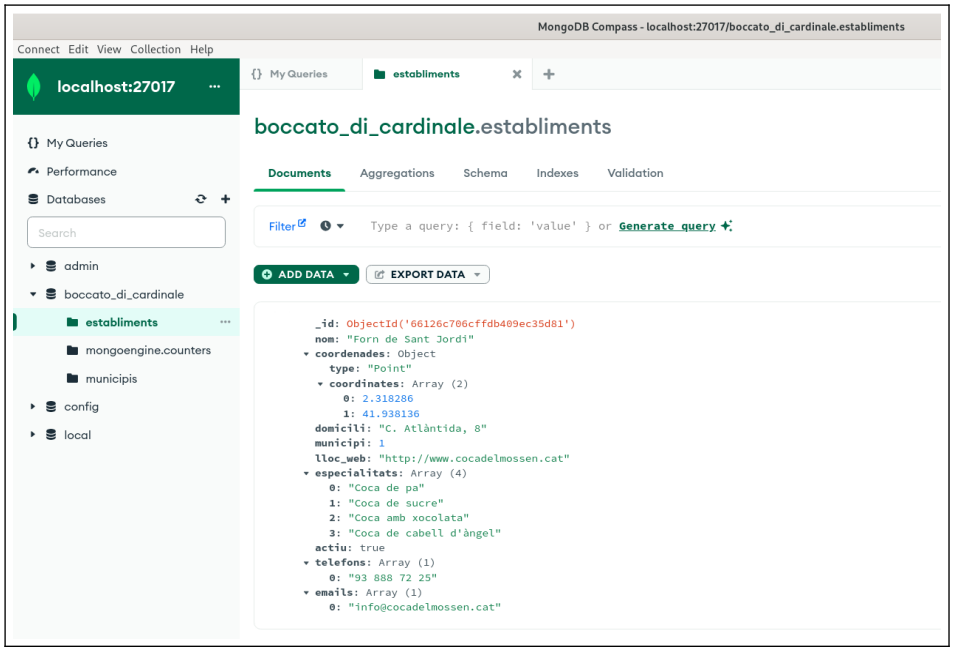
```
    # Creem un objecte de tipus Establiment i l'objecte de tipus Municipi que
    # correspon per poder definir la seva localització.
    m1 = Municipi(
        nom="Folgueroles",
        codi_postal="08519"
    )

    e1 = Establiment(
        nom="Forn de Sant Jordi",
        coordenades=(2.318286, 41.938136),
        domicili="C. Atlàntida, 8",
        municipi=m1,
        localitat=None,
        lloc_web="http://www.cocadelmossen.cat",
        especialitats=["Coca de pa", "Coca de sucre", "Coca amb xocolata",
                       "Coca de cabell d'àngel"],
        observacions=None,
        actiu=True,
        telefons=["93 888 72 25"],
        emails=["info@cocadelmossen.cat"]
    ).save()

    # Tanquem la connexió.
    me.disconnect()
```

Si executeu aquest programa havent eliminat prèviament tota la base de dades que teníem després d'haver fet les proves anteriors, veureu que apareix correctament el nou objecte `Establiment` en una collection anomenada `establiments` però, malgrat que a l'atribut `municipi` hi podem veure un identificador correcte, no ha aparegut cap collection anomenada `municipis` amb el municipi que hem creat. Per tant, no tenim aquesta referència resolta.

Un solució seria fer el `save()`, també, de l'objecte `Municipi`. Però hi ha una segona solució: executar el mètode `save()` amb el paràmetre `cascade=`**`True`**, la qual cosa provoca que es guardi l'objecte `Establiment` sobre el qual executem el `save()` i, també, tots els objectes referenciats des d'aquest (si és que no s'havien guardat prèviament):

```
...
    e1 = Establiment(
        nom="Forn de Sant Jordi",
        coordenades=(2.318286, 41.938136),
        domicili="C. Atlàntida, 8",
        municipi=m1,
        localitat=None,
        lloc_web="http://www.cocadelmossen.cat",
        especialitats=["Coca de pa", "Coca de sucre", "Coca amb xocolata",
                       "Coca de cabell d'àngel"],
        observacions=None,
        actiu=True,
        telefons=["93 888 72 25"],
        emails=["info@cocadelmossen.cat"]
    ).save(cascade=True)
...
```

Ara sí. Si mireu el contingut de la base de dades, veureu com s'ha creat l'objecte `Establiment` i l'objecte `Municipi`, i ambdós estan lligats a través de la referència feta amb l'identificador de municipi de tipus auto-numèric. Tot això ho haurà gestionat MongoEngine de forma automàtica.




Fixeu-vos en el següent exemple, en el qual consultem tots els establiments que tenim guardats a la base de dades:

```python
import mongoengine as me

# Classe Municipi que representarà aquest concepte en la nostra aplicació.
class Municipi(me.Document):
    id = me.SequenceField(primary_key=True)
    codi_postal = me.StringField(max_length=5, min_length=5, regex="\\d{5}")
    nom = me.StringField(max_length=128, required=True, unique_with="codi_postal")
    meta = {"collection": "municipis"}

# Classe Establiment que representarà aquest concepte en la nostra aplicació.
class Establiment(me.Document):
    nom = me.StringField(max_length=128, required=True, unique=True)
    coordenades = me.PointField(required=True)
    domicili = me.StringField(max_length=256, required=True)
    municipi = me.ReferenceField(Municipi)
    localitat = me.StringField(max_length=128)
    lloc_web = me.URLField()
    especialitats = me.ListField(me.StringField(max_length=64, required=True))
    observacions = me.StringField(max_length=256)
    actiu = me.BooleanField(required=True, default=True)
    telefons = me.ListField(me.StringField(max_length=32, required=True))
    emails = me.ListField(me.StringField(max_length=64, required=True))
    meta = {"collection": "establiments"}

if __name__ == "__main__":

    # Connexió sobre la base de dades "boccato_di_cardinale".
    db = me.connect(host="mongodb://127.0.0.1:27017/boccato_di_cardinale")

    # Consultem tots els municipis de la base de dades.
    establiments = Establiment.objects()

    # Mostrem el nom de l'establiment i el nom del municipi on es troba.
    for establiment in establiments:
        print(establiment.nom, establiment.municipi.nom, sep=", ")

    # Tanquem la connexió.
    me.disconnect()
```

La línia de codi que hem ressaltat amb color groc, permet comprovar com MongoEngine ens recupera automàticament l'objecte `Municipi` associat a cadascun dels establiments i el deixa referenciat a través de l'atribut `municipi` per poder-hi accedir quan i com vulguem.

# F2.2

## DESCRIPCIÓ DE LES DADES I DE LA SEVA ESTRUCTURA (II)

D'entrada, com ja hem dit anteriorment, volem tenir dos tipus concrets d'establiments: botigues i restaurants. Aquests dos tipus d'establiments els definirem com a subclasses de la classe `Establiment`, afegint-hi els atributs propis de cadascun.

Per poder crear subclasses a partir d'una classe gestionada per MongoEngine cal, en primer lloc, activar l'opció `allow_inheritance` a la classe pare, en el nostre cas, la classe `Establiment`:

```
1  import mongoengine as me
2
3  # Classe Municipi que representarà aquest concepte en la nostra aplicació.
4  class Municipi(me.Document):
5      id = me.SequenceField(primary_key=True)
6      codi_postal = me.StringField(max_length=5, min_length=5, regex="\\d{5}")
7      nom = me.StringField(max_length=128, required=True, unique_with="codi_postal")
8      meta = {"collection": "municipis"}
9
10 # Classe Establiment que representarà aquest concepte en la nostra aplicació.
11 class Establiment(me.Document):
12     nom = me.StringField(max_length=128, required=True, unique=True)
13     coordenades = me.PointField(required=True)
14     domicili = me.StringField(max_length=256, required=True)
15     municipi = me.ReferenceField(Municipi)
16     localitat = me.StringField(max_length=128)
17     lloc_web = me.URLField()
18     especialitats = me.ListField(me.StringField(max_length=64, required=True))
19     observacions = me.StringField(max_length=256)
20     actiu = me.BooleanField(required=True, default=True)
21     telefons = me.ListField(me.StringField(max_length=32, required=True))
22     emails = me.ListField(me.StringField(max_length=64, required=True))
23     meta = {"collection": "establiments",
24             "allow_inheritance": True}
```

Un cop fet això, ja podrem crear noves classes que siguin subclasses de la classe `Establiment`. En el nostre cas, tindrem les classes `Botiga` i `Restaurant`:

```
1  import mongoengine as me
2
3  class Municipi(me.Document):
4      id = me.SequenceField(primary_key=True)
5      codi_postal = me.StringField(max_length=5, min_length=5, regex="\\d{5}")
6      nom = me.StringField(max_length=128, required=True, unique_with="codi_postal")
7      meta = {"collection": "municipis"}
8
9  class Establiment(me.Document):
10     nom = me.StringField(max_length=128, required=True, unique=True)
11     coordenades = me.PointField(required=True)
```

```
    domicili = me.StringField(max_length=256, required=True)
    municipi = me.ReferenceField(Municipi)
    localitat = me.StringField(max_length=128)
    lloc_web = me.URLField()
    especialitats = me.ListField(me.StringField(max_length=64, required=True))
    observacions = me.StringField(max_length=256)
    actiu = me.BooleanField(required=True, default=True)
    telefons = me.ListField(me.StringField(max_length=32, required=True))
    emails = me.ListField(me.StringField(max_length=64, required=True))
    meta = {"collection": "establiments",
            "allow_inheritance": True}

class Botiga(Establiment):
    ...
...
class Restaurant(Establiment):
    ...
```

De les botigues, a més de tots els atributs que ja tenim heretats de la classe `Establiment`, en voldrem saber la informació següent:

- **Tipus de botiga**: de les botigues en voldrem saber el tipus (si es tracta d'un forn, una carnisseria, etc.). Crearem una classe que guardarem en una collection independent per representar aquest concepte de tipus de botiga. Cada tipus tindrà un nom principal i una llista de sinònims (per exemple, si el nom principal és carnisseria, podem tenir sinònims com xarcuteria, cansaladeria, etc). Farem que l'identificador d'aquest tipus d'objectes sigui un nombre enter correlatiu (un camp de tipus auto-numèric).

```
import mongoengine as me

# Classe Municipi que representarà aquest concepte en la nostra aplicació.
class Municipi(me.Document):
    id = me.SequenceField(primary_key=True)
    codi_postal = me.StringField(max_length=5, min_length=5, regex="\\d{5}")
    nom = me.StringField(max_length=128, required=True, unique_with="codi_postal")
    meta = {"collection": "municipis"}

# Classe Establiment que representarà aquest concepte en la nostra aplicació.
class Establiment(me.Document):
    nom = me.StringField(max_length=128, required=True, unique=True)
    coordenades = me.PointField(required=True)
    domicili = me.StringField(max_length=256, required=True)
    municipi = me.ReferenceField(Municipi)
    localitat = me.StringField(max_length=128)
    lloc_web = me.URLField()
    especialitats = me.ListField(me.StringField(max_length=64, required=True))
    observacions = me.StringField(max_length=256)
    actiu = me.BooleanField(required=True, default=True)
    telefons = me.ListField(me.StringField(max_length=32, required=True))
    emails = me.ListField(me.StringField(max_length=64, required=True))
    meta = {"collection": "establiments",
            "allow_inheritance": True}

# Classe TipusBotiga que ha de permetre representar els diversos tipus de botigues que podem tenir
# a la base de dades.
class TipusBotiga(me.Document):
    id = me.SequenceField(primary_key=True)
    nom = me.StringField(max_length=64, unique=True)
    sinonims = me.ListField(me.StringField(max_length=32, required=True, unique=True))
    meta={"collection": "tipus_botigues"}

```

```
34  # Classe Botiga que té tot el que hereta d'Establiment, més l'atribut per definir el tipus.
35  class Botiga(Establiment):
36      tipus = me.ReferenceField(TipusBotiga, required=True)
```

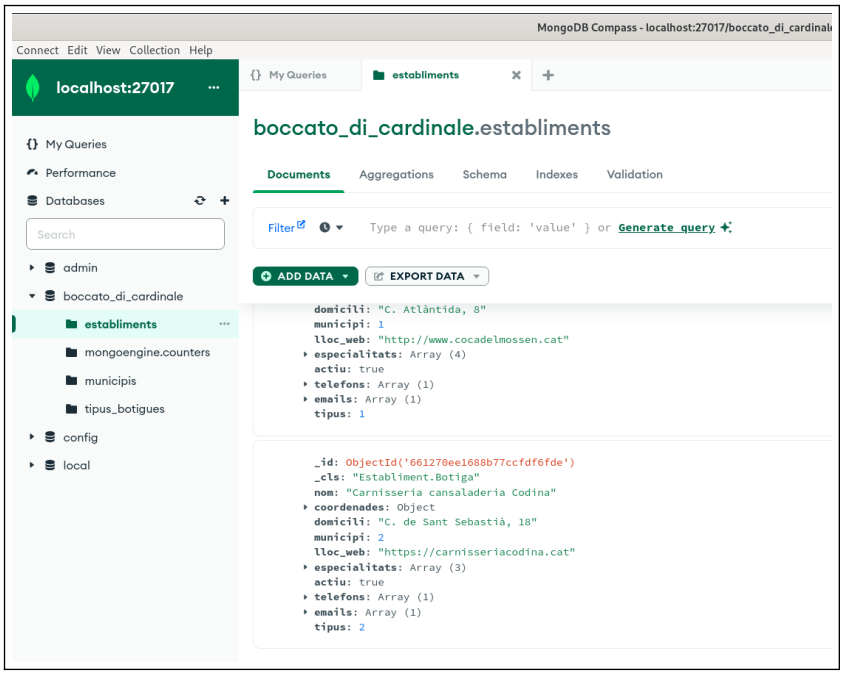
Fixeu-vos que l'atribut `id` de la classe `TipusBotiga` l'hem definit com a `SequenceField` (com en el cas dels identificadors de municipi).

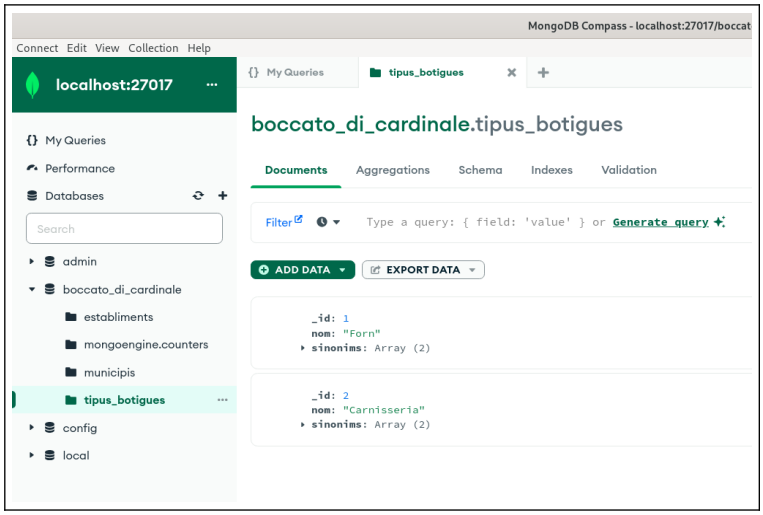
Un cop afegits aquests nous elements a l'estructura de les dades, podem ampliar la creació d'objectes per veure els resultats que obtenim sobre la base de dades:

```
...  ...
38  if __name__ == "__main__":
39
40      # Connexió sobre la base de dades "boccato_di_cardinale".
41      db = me.connect(host="mongodb://127.0.0.1:27017/boccato_di_cardinale")
42
43      # Creem dos objectes de tipus Botiga, amb els seus tipus i els seus municipis.
44      m1 = Municipi(
45          nom="Folgueroles",
46          codi_postal="08519"
47      )
48
49      m2 = Municipi(
50          nom="Taradell",
51          codi_postal="08552"
52      )
53
54      t1 = TipusBotiga(
55          nom="Forn",
56          sinonims=["Fleca", "Forn de pa"]
57      )
58
59      t2 = TipusBotiga(
60          nom="Carnisseria",
61          sinonims=["Cansaladeria", "Xarcuteria"]
62      )
63
64      b1 = Botiga(
65          nom="Forn de Sant Jordi",
66          coordenades=(2.318286, 41.938136),
67          domicili="C. Atlàntida, 8",
68          municipi=m1,
69          localitat=None,
70          lloc_web="http://www.cocadelmossen.cat",
71          especialitats=["Coca de pa", "Coca de sucre", "Coca amb xocolata",
72                         "Coca de cabell d'àngel"],
73          observacions=None,
74          actiu=True,
75          telefons=["93 888 72 25"],
76          emails=["info@cocadelmossen.cat"],
77          tipus=t1
78      ).save(cascade=True)
79
80      b2 = Botiga(
81          nom="Carnisseria cansaladeria Codina",
82          coordenades=(2.287847, 41.873690),
83          domicili="C. de Sant Sebastià, 18",
84          municipi=m2,
85          localitat=None,
86          lloc_web="https://carnisseriacodina.cat",
87          especialitats=["Botifarra d'ou", "Llonganissa", "Bull de ratafia amb tòfona"],
88          observacions=None,
89          actiu=True,
90          telefons=["93 880 11 75"],
91          emails=["info@carnisseriacodina.com"],
```

```
92          tipus=t2
93      ).save(cascade=True)
94
95      # Tanquem la connexió.
96      me.disconnect()
```

Executant aquest programa, obtindrem els objectes que es poden veure a continuació sobre la nostra base de dades:




Fixeu-vos que en la nova estructura de la base de dades han aparegut alguns elements que no havíem vist anteriorment:

- **Atribut `_cls` a la collection `establiments`**: aquest atribut que crea automàticament MongoEngine serveix per indicar, quan tenim l'herència activada, el tipus concret de cadascun dels objectes que tenim en una collection.

Dels restaurants, a més de tots els atributs que ja tenim heretats de la classe `Establiment`, en voldrem saber la informació següent:

- **Oferta gastronòmica**: dels restaurants en voldrem saber la seva oferta gastronòmica: diferents tipus de menú que ofereix, carta de plats, carta de vins, etc. Ho representarem amb una llista de cadenes de caràcters amb el nom de cada oferta disponible. Pot deixar-se buida si es desconeix en què consisteix l'oferta gastronòmica del restaurant.

```python
import mongoengine as me

# Classe Municipi que representarà aquest concepte en la nostra aplicació.
class Municipi(me.Document):
    id = me.SequenceField(primary_key=True)
    codi_postal = me.StringField(max_length=5, min_length=5, regex="\\d{5}")
    nom = me.StringField(max_length=128, required=True, unique_with="codi_postal")
    meta = {"collection": "municipis"}

# Classe Establiment que representarà aquest concepte en la nostra aplicació.
class Establiment(me.Document):
    nom = me.StringField(max_length=128, required=True, unique=True)
    coordenades = me.PointField(required=True)
    domicili = me.StringField(max_length=256, required=True)
    municipi = me.ReferenceField(Municipi)
    localitat = me.StringField(max_length=128)
    lloc_web = me.URLField()
    especialitats = me.ListField(me.StringField(max_length=64, required=True))
    observacions = me.StringField(max_length=256)
    actiu = me.BooleanField(required=True, default=True)
    telefons = me.ListField(me.StringField(max_length=32, required=True))
    emails = me.ListField(me.StringField(max_length=64, required=True))
    meta = {"collection": "establiments",
            "allow_inheritance": True}

# Classe TipusBotiga que ha de permetre representar els diversos tipus de botigues que podem tenir.
class TipusBotiga(me.Document):
    id = me.SequenceField(primary_key=True)
    nom = me.StringField(max_length=64, unique=True)
    sinonims = me.ListField(me.StringField(max_length=32, required=True, unique=True))
    meta={"collection": "tipus_botigues"}

# Classe Botiga que té tot el que hereta d'Establiment, més l'atribut per definir el tipus.
class Botiga(Establiment):
    tipus = me.ReferenceField(TipusBotiga, required=True)

# Classe Restaurant que té tot el que hereta d'Establiment, més l'atribut per definir l'oferta.
class Restaurant(Establiment):
    oferta_gastronomica = me.ListField(
                              me.StringField(max_length=32, required=True, unique=True),
                              required=False)
```

Finalment, per a tots els tipus d'establiments, voldrem que es pugui representar el seu horari d'obertura (quins dies i, per cada dia, quines franges horàries). Això ho resoldrem amb dues classes noves: la classe `HorariDiari` i la classe `FranjaHoraria`.

- **`FranjaHoraria`**: permet representar, com el seu nom indica, una franja horària que queda definida mitjançant una hora inicial i una hora final.

- **`HorariDiari`**: permet representar l'horari d'obertura d'un dia determinat, mitjançant un atribut numèric que pot valer entre 1 i 7 per representar el dia de la setmana (1 → dilluns fins a 7 → diumenge), i una llista d'objectes de tipus `FranjaHoraria` **incrustats**.

Cada objecte `Establiment` podrà estar associat amb fins a un màxim de 7 objectes de tipus `HorariDiari` que permetran representar l'horari d'obertura de tots els dies de la setmana. Aquesta associació la resoldrem amb una llista d'objectes **incrustats**. Vegem com quedaria el codi:

```python
import mongoengine as me

# Classe Municipi que representarà aquest concepte en la nostra aplicació.
class Municipi(me.Document):
    id = me.SequenceField(primary_key=True)
    codi_postal = me.StringField(max_length=5, min_length=5, regex="\\d{5}")
    nom = me.StringField(max_length=128, required=True, unique_with="codi_postal")
    meta = {"collection": "municipis"}

# Classe que representa el concepte de franja horària, amb una hora d'inici i una hora final.
class FranjaHoraria(me.EmbeddedDocument):
    hora_obertura =
        me.StringField(min_length=5, max_length=5, regex="([01]\\d|2[0-3]):[0-5]\\d", required=True)
    hora_tancament =
        me.StringField(min_length=5, max_length=5, regex="([01]\\d|2[0-3]):[0-5]\\d", required=True)

# Classe que representa l'horari diari, indicant el dia de la setmana i les franges horàries.
class HorariDiari(me.EmbeddedDocument):
    dia = me.IntField(min_value=1, max_value=7, required=True)
    franges = me.ListField(me.EmbeddedDocumentField(FranjaHoraria))

# Classe Establiment que representarà aquest concepte en la nostra aplicació.
class Establiment(me.Document):
    nom = me.StringField(max_length=128, required=True, unique=True)
    coordenades = me.PointField(required=True)
    domicili = me.StringField(max_length=256, required=True)
    municipi = me.ReferenceField(Municipi)
    localitat = me.StringField(max_length=128)
    lloc_web = me.URLField()
    especialitats = me.ListField(me.StringField(max_length=64, required=True))
    observacions = me.StringField(max_length=256)
    actiu = me.BooleanField(required=True, default=True)
    telefons = me.ListField(me.StringField(max_length=32, required=True))
    emails = me.ListField(me.StringField(max_length=64, required=True))
    horari = me.ListField(me.EmbeddedDocumentField(HorariDiari), max_length=7, required=True)
    meta = {"collection": "establiments",
            "allow_inheritance": True}

# Classe TipusBotiga que ha de permetre representar els diversos tipus de botigues que podem tenir.
class TipusBotiga(me.Document):
    id = me.SequenceField(primary_key=True)
    nom = me.StringField(max_length=64, unique=True)
    sinonims = me.ListField(me.StringField(max_length=32, required=True, unique=True))
    meta={"collection": "tipus_botigues"}

```

```python
# Classe Botiga que té tot el que hereta d'Establiment, més l'atribut per definir el tipus.
class Botiga(Establiment):
    tipus = me.ReferenceField(TipusBotiga, required=True)

# Classe Restaurant que té tot el que hereta d'Establiment, més l'atribut per definir l'oferta.
class Restaurant(Establiment):
    oferta_gastronomica = me.ListField(
                                me.StringField(max_length=32, required=True, unique=True),
                                required=False)
```

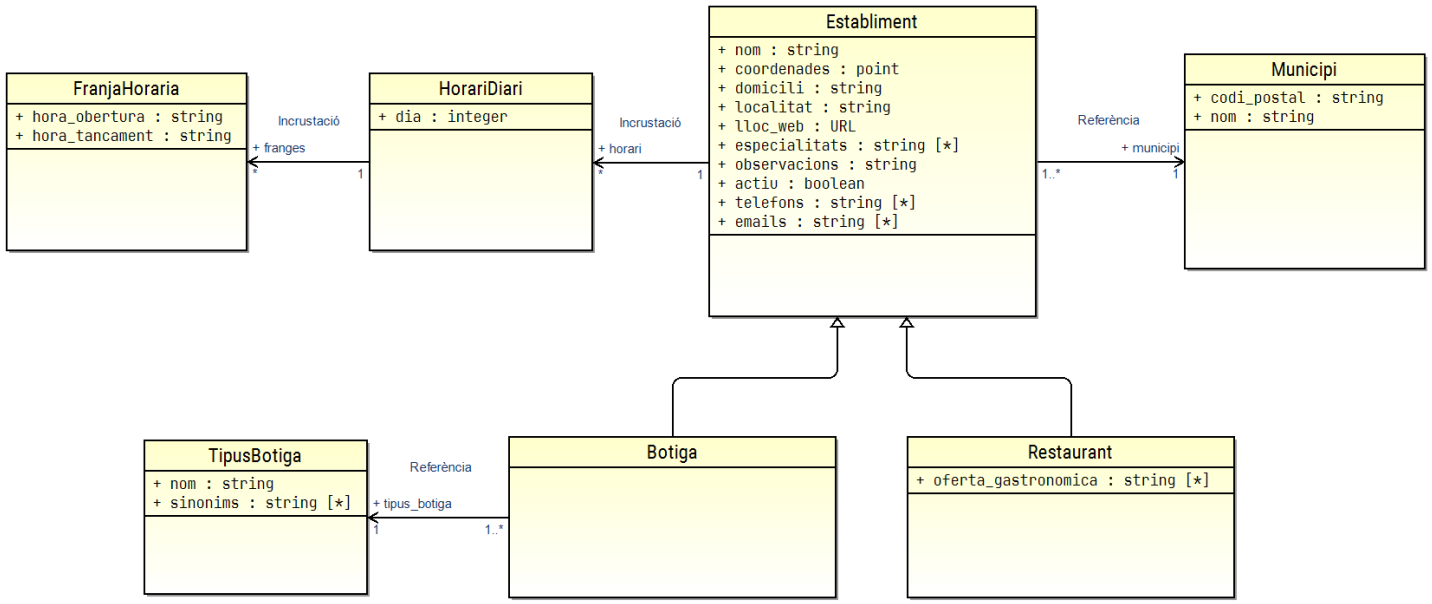
Respecte del codi que hem afegit (el que es pot observar ressaltat amb color groc), cal comentar els aspectes següents:

- **`EmbeddedDocument`**: les dues classes que hem afegit, les hem declarat com a subclasses de `EmbeddedDocument` en comptes de fer-ho de `Document`. Això és necessari fer-ho per aquelles classes que volem que generin objectes que puguin ser incrustats dins d'altres objectes, en comptes de ser referenciats.

- **`EmbeddedDocumentField`**: és la directiva que hem d'utilitzar per declarar atributs que volem que es representin mitjançant objectes incrustats a la base de dades, indicant entre parèntesis el tipus d'objecte que volem incrustar.

- **`regex`**: com ja s'ha dit anteriorment, podem definir expressions regulars per validar el contingut d'atributs de tipus `String`. En el nostre cas, hem utilitzat l'expressió regular que obliga a respectar el format `HH:MM` per a les hores, fent que `HH` hagi de ser un valor entre 00 i 23, i `MM` hagi de ser un valor entre 00 i 59.

El diagrama de classes que representa l'estructura definitiva de les dades de la nostra aplicació és el que es pot veure a continuació:



# F2.3

## ESTRUCTURA MODULAR DEL CODI DE L'APLICACIÓ

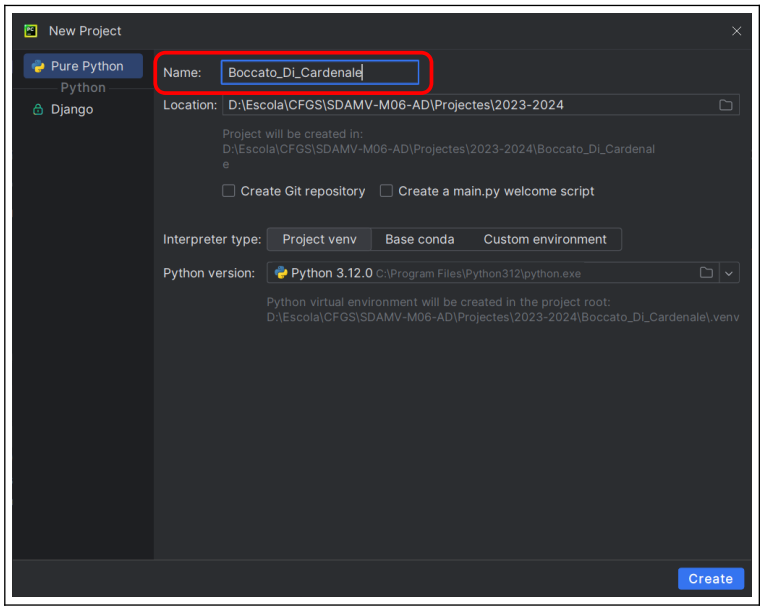
Quan desenvolupem una aplicació de mida mitjana o gran, es converteix en una necessitat estructurar el seu codi de forma modular. Per fer-ho, caldrà veure alguns dels elements que ens ofereix el llenguatge Python per donar aquesta estructura als programes. En primer lloc, veiem els dos conceptes principals que ens ofereix aquest llenguatge per poder implementar una estructura modular:

- **Mòdul**: en el llenguatge Python, un mòdul és un arxiu que conté codi font que pot ser reutilitzat en altres arxius de codi font i, fins i tot, en diverses aplicacions. Habitualment, un mòdul acostuma a tenir declaracions de funcions (si estem desenvolupant software amb un enfocament estructurat) o declaracions de classes (si estem utilitzant un enfocament orientat a objectes).

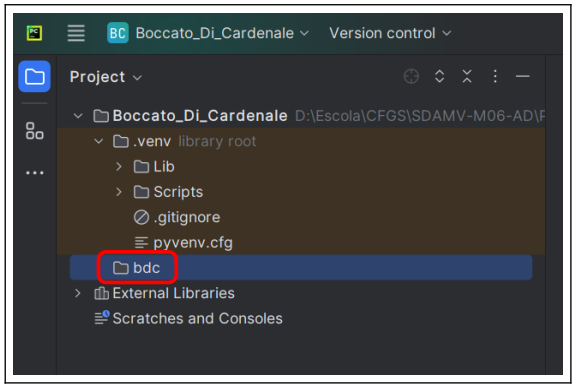
- **Package**: en el llenguatge Python, un package és un directori o carpeta que conté un o diversos mòduls. Aquest directori, a més, ha de contenir obligatòriament un mòdul o arxiu anomenat `__init__.py` perquè pugui fer les funcions de package.

En el cas de la nostra aplicació, crearem un package que anomenarem `bdc`, dins del qual hi crearem mòduls que continguin elements reutilitzables com, per exemple, les classes que defineixen els objectes que acabem guardant a la base de dades. Per veure com podem fer això, seguiu els passos que s'indiquen a continuació:

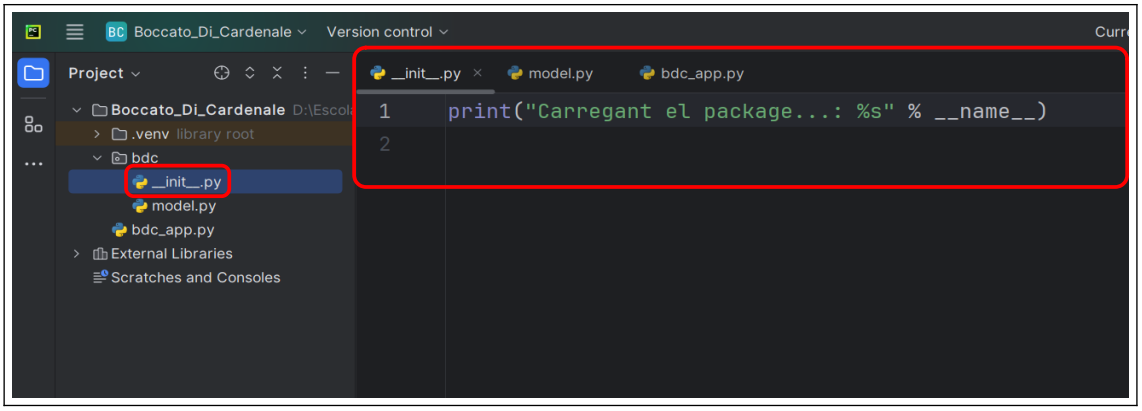
**a.** Creeu un nou projecte anomenat `Boccato_Di_Cardenale`:



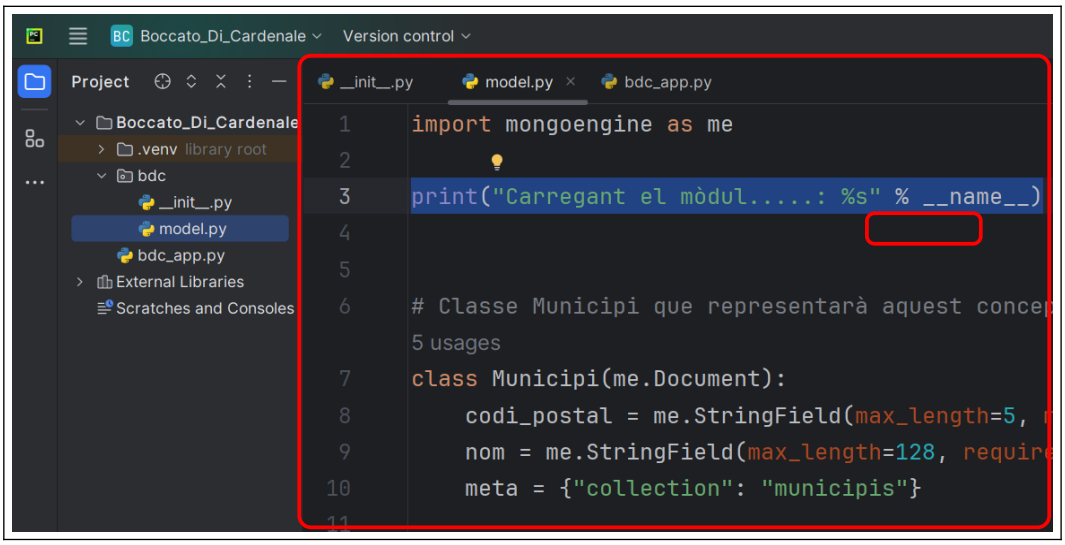
**b.** Dins d'aquest nou projecte, creeu una carpeta anomenada `bdc`:




**c.** Creeu un arxiu anomenat `__init__.py` dins d'aquesta carpeta. Si volem que la carpeta `bdc` pugui fer les funcions de package, és obligatori que tingui un fitxer anomenat així a dins seu. És molt habitual deixar buit aquest arxiu però, si hi posem algun tipus de codi, s'executarà en el moment que es faci la importació del package. Nosaltres hi posarem un `print()` com el següent per poder comprovar en quin moment s'executa:




**d.** Creeu un arxiu anomenat `model.py` dins del package `bdc`. Dins d'aquest package, hi posarem totes les definicions de classes que hem proposat fins ara (no el codi principal per fer proves d'insercions, consultes, etc.). Hi afegirem una instrucció `print()` com el següent a l'inici per poder observar, també, en quin moment s'executa:

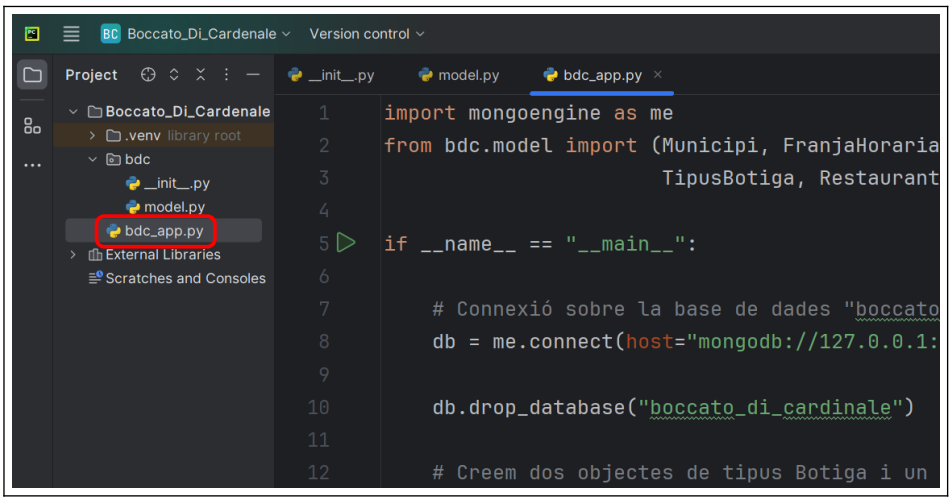


**e.** Creeu un arxiu dins del projecte, que no es trobi inclòs en el package `bdc`. Anomeneu-lo `bdc_app.py` i poseu-hi el codi següent:

```python
import mongoengine as me
from bdc.model import (Municipi, FranjaHoraria, HorariDiari, Botiga,
                       TipusBotiga, Restaurant)

if __name__ == "__main__":

    # Connexió sobre la base de dades "boccato_di_cardinale".
    db = me.connect(host="mongodb://127.0.0.1:27017/boccato_di_cardinale")

    db.drop_database("boccato_di_cardinale")

    # Creem dos objectes de tipus Botiga i un de tipus Restaurant, amb tots els
    # objectes referenciats i incrustats necessaris per completar-ne les dades.
    m1 = Municipi(nom="Folgueroles", codi_postal="08519")
    m2 = Municipi(nom="Taradell", codi_postal="08552")
    m3 = Municipi(nom="Toses", codi_postal="17536")

    t1 = TipusBotiga(nom="Forn", sinonims=["Fleca", "Forn de pa"])
    t2 = TipusBotiga(nom="Carnisseria", sinonims=["Cansaladeria", "Xarcuteria"])

    f1a = FranjaHoraria(hora_obertura="08:00", hora_tancament="13:30")
    f1b = FranjaHoraria(hora_obertura="16:30", hora_tancament="20:00")

    f2a = FranjaHoraria(hora_obertura="08:30", hora_tancament="13:30")
    f2b = FranjaHoraria(hora_obertura="17:00", hora_tancament="19:30")
    f2c = FranjaHoraria(hora_obertura="08:30", hora_tancament="14:00")

    f3a = FranjaHoraria(hora_obertura="09:00", hora_tancament="16:00")
    f3b = FranjaHoraria(hora_obertura="20:00", hora_tancament="22:00")

    h1dl = HorariDiari(dia=1, franges=None)
    h1dm = HorariDiari(dia=2, franges=[f1a, f1b])
    h1dc = HorariDiari(dia=3, franges=[f1a, f1b])
    h1dj = HorariDiari(dia=4, franges=[f1a, f1b])
    h1dv = HorariDiari(dia=5, franges=[f1a, f1b])
    h1ds = HorariDiari(dia=6, franges=[f1a, f1b])
    h1dg = HorariDiari(dia=7, franges=[f1a])

    h2dl = HorariDiari(dia=1, franges=None)
    h2dm = HorariDiari(dia=2, franges=[f2a, f2b])
    h2dc = HorariDiari(dia=3, franges=[f2a, f2b])
    h2dj = HorariDiari(dia=4, franges=[f2a, f2b])
    h2dv = HorariDiari(dia=5, franges=[f2c, f2b])
    h2ds = HorariDiari(dia=6, franges=[f2c])
    h2dg = HorariDiari(dia=7, franges=None)

    h3dl = HorariDiari(dia=1, franges=[f3a])
    h3dm = HorariDiari(dia=2, franges=None)
    h3dc = HorariDiari(dia=3, franges=[f3a])
    h3dj = HorariDiari(dia=4, franges=[f3a])
    h3dv = HorariDiari(dia=5, franges=[f3a, f3b])
    h3ds = HorariDiari(dia=6, franges=[f3a, f3b])
    h3dg = HorariDiari(dia=7, franges=[f3a])

    e1 = Botiga(
        nom="Forn de Sant Jordi",
        coordenades=(2.318286, 41.938136),
        domicili="C. Atlàntida, 8",
        municipi=m1,
        localitat=None,
        lloc_web="http://www.cocadelmossen.cat",
        especialitats=["Coca de pa", "Coca de sucre", "Coca amb xocolata",
                       "Coca de cabell d'àngel"],
        observacions=None,
        actiu=True,
        telefons=["93 888 72 25"],
```

```
        emails=["info@cocadelmossen.cat"],
        horari=[h1dl, h1dm, h1dc, h1dj, h1dv, h1ds, h1dg],
        tipus=t1
    ).save(cascade=True)

    e2 = Botiga(
        nom="Carnisseria cansaladeria Codina",
        coordenades=(2.287847, 41.873690),
        domicili="C. de Sant Sebastià, 18",
        municipi=m2,
        localitat=None,
        lloc_web="https://carnisseriacodina.cat",
        especialitats=["Botifarra d'ou", "Llonganissa", "Bull de ratafia amb tòfona"],
        observacions=None,
        actiu=True,
        telefons=["93 880 11 75"],
        emails=["info@carnisseriacodina.com"],
        horari=[h2dl, h2dm, h2dc, h2dj, h2dv, h2ds, h2dg],
        tipus=t2
    ).save(cascade=True)

    e3 = Restaurant(
        nom="Can Casanova",
        coordenades=(2.047425, 42.323665),
        domicili="Camí de Toses, 2",
        municipi=m3,
        localitat="Fornells de la Muntanya",
        lloc_web="https://www.restaurantcancasanova.cat/ca",
        especialitats=["Escudella barrejada", "Carn d'olla"],
        observacions="Veure detalls de l'oferta gastronòmica a la web.",
        actiu=True,
        telefons=["972 73 60 75", "972 73 63 01"],
        emails=None,
        horari=[h3dl, h3dm, h3dc, h3dj, h3dv, h3ds, h3dg],
        oferta_gastronomica=["Menú diari (laborables)", "Carta",
                             "Carta de temporada", "Menús de grup"]
    ).save(cascade=True)

    # Tanquem la connexió.
    me.disconnect()
```

Aquest arxiu us ha de quedar situat, com hem dit, a nivell de projecte i no dins del package `bdc` com ho estan els arxius `__init__.py` i `model.py`:



**f.** Fixeu-vos com, ara, en el programa principal, podem utilitzar totes les classes de la capa model simplement fent els **`import`**s corresponents. Si tenim, per exemple, un programa per carregar objectes a la base de dades i un altre per realitzar consultes, no caldrà repetir el codi que declara les classes del model a tots dos arxius de codi font.

**g.** Fixeu-vos que, quan s'executa el programa, podem veure per consola el missatge que ens mostra el `print()` que tenim a l'arxiu `__init__.py` del package i, també, el missatge que ens mostra el `print()` que tenim a l'arxiu `model.py` del mòdul. Per tant, podem deduir que totes aquelles instruccions que tinguem en un arxiu de package o en un arxiu de mòdul que no es trobin dins d'una funció o dins dels mètodes d'una classe, s'executaran automàticament en el moment que els importem.

**h.** Una altra cosa que val la pena tenir present és el valor de la variable `__name__`. Fixeu-vos que, en el cas de l'arxiu `__init__.py` d'un package, en el moment que s'executa, aquesta variable té com a valor el nom del package. En el cas de l'arxiu `.py` d'un mòdul, en el moment d'executar-se, aquesta variable té com a valor el nom del mòdul.

**i.** Per tant, la variable `__name__` ens permet saber, en el context d'un determinat bloc de codi, en quin arxiu de package o arxiu de mòdul es troba aquell codi. Si preguntem pel valor d'aquesta variable dins del codi de l'arxiu que executem de forma directa per iniciar el programa veurem que no ens dóna com a resultat el nom de l'arxiu sinó que ens dóna com a resultat "`__main__`". Amb això podreu deduir la utilitat que té el condicional **`if`** `__name__ == "__main__":` que típicament posem com a punt d'inici dels programes de Python.

**RF.02** Un cop hàgiu comprovat el funcionament de tots els aspectes que s'han explicat fins ara (d'utilització de MongoEngine, d'estructuració de les dades en classes i d'estructuració d'aplicacions en mòduls i packages), aviseu el professor perquè us faci la revisió del funcionament amb tots els elements que hem proposat fins ara.

# BOCCATO DI CARDINALE – FASE 3

En aquesta tercera fase, proposarem desenvolupar algunes funcionalitats d'una hipotètica aplicació de gestió d'aquesta base de dades d'establiments, treballant sobre l'estructura de classes que hem proposat.

## F3.1

### DESENVOLUPAMENT DE CONSULTES SENZILLES

En primer lloc, cal desenvolupar la part d'una petita aplicació que se us dóna parcialment implementada, que permet veure, en primer lloc, tots els municipis que tenim a la base de dades (en cadascun dels quals hi haurà un o més establiments) i, per cada municipi, permet consultar tots els establiments que hi ha. A continuació es descriu l'estructura de l'aplicació que se us dóna i s'explica què cal fer i completar per aconseguir el seu correcte funcionament:

**a.** L'aplicació està estructurada de forma modular, segons l'arquitectura MVC (model-view-controller).

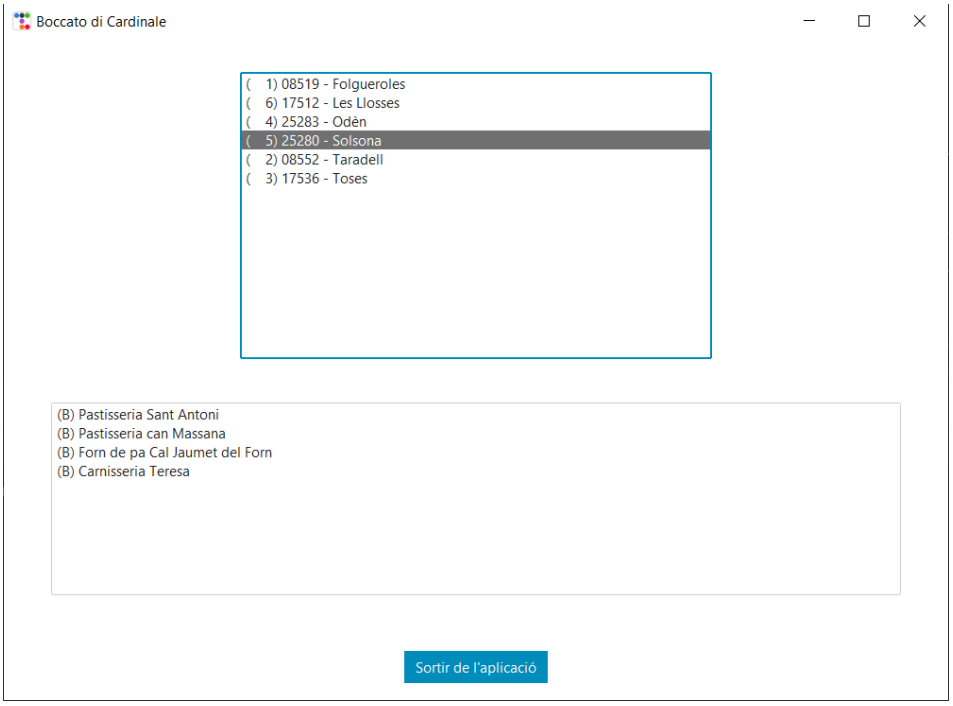
**b.** L'aplicació té un package o mòdul anomenat `bdc`, que conté dos arxius de codi font que es descriuen a continuació:

- **`model.py`** → és l'arxiu que conté tota l'estructura del model de l'aplicació, és a dir, de les dades que gestiona. Aquí hi trobareu totes les classes que hem anat comentant durant les dues primeres fases del projecte.
- **`controller.py`** → és l'arxiu que conté les accions que cal executar per dur a terme les operacions que es poden sol·licitar a través de la interfície d'usuari de l'aplicació (la capa view). Conté dues funcions que no tenen codi i que són les que heu de completar (estan documentades perquè tingueu el detall de què és el que han de fer exactament).

**c.** Fora del package, en el directori principal del projecte, hi trobareu dos arxius de codi font que es poden executar (contenen punt d'inici d'execució `__main__`):

- **`bdc_app.py`** → programa que crea una base de dades a partir de zero, amb l'estructura que tenim definida a l'arxiu `model.py` i, a més, hi insereix una certa quantitat d'objectes per poder disposar-ne com a joc de proves.

– **`bdc_app_query_gui.py`** → programa que genera una petita aplicació amb interfície gràfica d'usuari que, quan s'hagi completat correctament el codi de l'arxiu `controller.py`, permetrà veure els municipis que tenim a la base de dades ordenats per nom i, seleccionant un dels municipis, podrem veure els establiments que hi ha:




**d.** Un cop entès el que s'ha d'aconseguir, en primer lloc, reviseu el codi que se us dóna fet en els diversos arxius del projecte procurant entendre què fa cada bloc. Cal llegir bé, com ja s'ha indicat, els comentaris que trobareu a l'arxiu `controller.py` que expliquen amb detall què han de fer les dues funcions que s'han de completar. D'aquesta manera, hauríeu de tenir una visió clara del què cal fer.

**e.** Completeu el codi de les dues funcions que hi ha a l'arxiu `controller.py` per fer que portin a terme la funció que es descriu i que, com a resultat final, facin que en executar l'arxiu `bdc_acc_query_gui.py` l'aplicació que hem descrit funcioni com s'ha indicat.

**RQ.01** Un cop tingueu l'aplicació en funcionament havent completat les dues funcions de l'arxiu `controller.py`, aviseu el professor perquè us en faci la revisió del seu funcionament.

# F3.2

## INSERCIÓ DE VALORS EN ATRIBUTS DE TIPUS LLISTA

En segon lloc, cal desenvolupar una petita aplicació que permeti afegir noves especialitats a algun dels establiments ja existents a la base de dades. Per fer-ho, caldrà implementar un programa (que treballi via consola, sense interfície gràfica) seguint els passos que s'indiquen a continuació:

**a.** Per al desenvolupament d'aquest programa, agafeu com a base el package `bdc` del projecte de la fase 3.1, amb els arxius `model.py` i `controller.py`.

**b.** Per començar, mostreu una llista amb tots els establiments de la base de dades, ordenats per nom, numerant-los a partir de l'1. Feu que a la llista es vegi el nom de cada establiment, el codi postal i el nom del municipi on es troba ubicat:

```
Llistat d'establiments que hi ha a la base de dades:

   1: Can Casanova                              (17536 - Toses)
   2: Carnisseria Teresa                        (25280 - Solsona)
   3: Carnisseria cansaladeria Codina           (08552 - Taradell)
   4: Forn de Sant Jordi                        (08519 - Folgueroles)
   5: Forn de pa Cal Jaumet del Forn            (25280 - Solsona)
   6: La Botiga de ca l'Àngel                   (25283 - Odèn)
   7: Mas el Lladré                             (17512 - Les Llosses)
   8: Pastisseria Sant Antoni                   (25280 - Solsona)
   9: Pastisseria can Massana                   (25280 - Solsona)
```

**c.** Poseu el codi que porta a terme la consulta dels establiments que tenim a la base de dades en una funció dins de l'arxiu `controller.py`, i crideu aquesta funció des de l'arxiu que conté el codi principal del programa.

**d.** A continuació, s'haurà de demanar el número d'establiment a l'usuari, sobre el qual s'afegirà la nova especialitat.

**e.** Un cop escollit l'establiment, haurem de mostrar totes les seves especialitats, numerant-les, també, a partir del número 1:

```
Llistat d'establiments que hi ha a la base de dades:

   1: Can Casanova                              (17536 - Toses)
   2: Carnisseria Teresa                        (25280 - Solsona)
   3: Carnisseria cansaladeria Codina           (08552 - Taradell)
   4: Forn de Sant Jordi                        (08519 - Folgueroles)
   5: Forn de pa Cal Jaumet del Forn            (25280 - Solsona)
   6: La Botiga de ca l'Àngel                   (25283 - Odèn)
   7: Mas el Lladré                             (17512 - Les Llosses)
   8: Pastisseria Sant Antoni                   (25280 - Solsona)
   9: Pastisseria can Massana                   (25280 - Solsona)

Introdueix el número d'establiment que vols modificar: 4

Les especialitats d'aquest establiment són:

   1: Coca de pa
   2: Coca de sucre
   3: Coca amb xocolata
   4: Coca de cabell d'àngel
```

**e.** Un cop fet això, demanarem a l'usuari que introdueixi la nova especialitat que vol afegir a l'establiment. La nova especialitat s'haurà d'afegir després de totes les que ja hi hagués prèviament. Podeu consultar la forma de dur a terme aquesta actualització a l'apartat següent de la documentació oficial de mongoengine:

https://docs.mongoengine.org/guide/querying.html#atomic-updates

**RQ.02** Un cop tingueu aquest programa en funcionament de manera que es puguin afegir noves especialitats als establiments que tenim a la base de dades, aviseu el professor perquè us en faci la revisió del seu funcionament.

# F3.3

## ELIMINACIÓ DE VALORS EN ATRIBUTS DE TIPUS LLISTA

Per acabar aquesta tercera fase, cal desenvolupar una petita aplicació que permeti eliminar especialitats a algun dels establiments ja existents a la base de dades. Per fer-ho, caldrà implementar un programa (que treballi via consola, sense interfície gràfica) seguint els passos que s'indiquen a continuació:

**a.** Per al desenvolupament d'aquest programa, agafeu com a base el package `bdc` del projecte de la fase 3.1, amb els arxius `model.py` i `controller.py`.

**b.** Igual que en el cas anterior, mostreu tots els establiments de la base de dades numerant-los a partir de l'1 i, a continuació, demaneu a l'usuari el número d'establiment al qual se li vol eliminar alguna de les seves especialitats.

**c.** Un cop escollit l'establiment, igual que hem fet abans, haurem de mostrar totes les seves especialitats, numerant-les, també, a partir del número 1.

**d.** Un cop fet això, demanarem a l'usuari que introdueixi el número de l'especialitat que vol eliminar de l'establiment seleccionat i procedirem a dur a terme l'eliminació.

**E.01 / RQ.03** Un cop tingueu aquest programa en funcionament de manera que es puguin eliminar especialitats dels establiments que tenim a la base de dades, aviseu el professor perquè us en faci la revisió del seu funcionament.

## AVALUACIÓ DE LA FEINA FETA EN LA TERCERA FASE DEL PROJECTE

Un cop finalitzada aquesta tercera fase, cal entregar, a través de Moodle, el projecte Python que hàgiu creat, empaquetat en un fitxer comprimit en format `.zip`, `.gz` o `.rar`. El nom d'aquest arxiu haurà de ser el següent:

```
BoccatoDiCardenale-Fase3-Cognoms.zip (.gz o .rar)
```

L'avaluació del programa Python d'aquesta entrega es farà tenint en compte els aspectes que es detallen a continuació donant, a cadascun, el pes que s'indica sobre la nota final:

| Aspecte | Ponderació |
|---|---|
| **FUNCIONAMENT**<br>Entenent com a funcionament el fet que el programa realitzi correctament les funcions que es descriuen en l'enunciat. | 60% |
| **ESTRUCTURACIÓ DEL CODI**<br>El codi del programa està ben organitzat: les operacions que es realitzen es fan en un ordre lògic i correcte, no es repeteix codi d'una manera innecessària, no hi ha estructures de codi incoherents, les operacions es fan d'una forma mínimament òptima i eficient, etc. | 10% |
| **FIABILITAT I ROBUSTESA**<br>Valoració sobre si hi ha una bona comprovació de les situacions d'error que pot experimentar l'aplicació, sobre si hi ha una bona resposta en aquests casos (ja sigui intentant recuperar-se de la situació d'error, o bé donant un missatge explicatiu sobre l'error que s'ha produït i les seves causes) i, també especialment, si s'ha utilitzat correctament el sistema de control d'errors i excepcions de Python per a la gestió de les situacions de funcionament anormal del programa. | 20% |
| **PRESENTACIÓ DEL CODI**<br>Claredat, llegibilitat i presentació del codi font del programa: bona tabulació del codi, documentació del codi fent servir comentaris clars, concisos i útils, utilitzar noms de variables que ajudin a entendre la seva utilitat i amb valors inicials explícits quan sigui necessari, etc. | 10% |

## TIPUS D'ENTREGUES I REVISIONS

En diversos punts d'aquest projecte es plantejaran revisions i entregues per poder fer un seguiment progressiu de la feina que es demana que feu:

- **Entregues**: per fer-les, caldrà penjar a la tasca de Moodle corresponent el projecte complet comprimit (tota la carpeta del projecte). Si no es diu el contrari, totes les entregues tindran una qualificació numèrica obtinguda segons els criteris que s'indiquin.

- **Revisions**: es poden proposar tres tipus de revisions que són les que es detallen a continuació:

  – **Revisions Formatives (RF)**: es tracta de revisions que tenen dos objectius. En primer lloc, que el professor pugui supervisar la feina feta fins el moment i, com a resultat de la revisió, pugui donar feedback de les coses que cal rectificar o millorar i de les coses que estan ben treballades. I, en segon lloc, tenir un punt de referència en el calendari sobre en quin punt del desenvolupament s'hauria de trobar el projecte.

    Com que l'objectiu d'aquestes revisions és estrictament formatiu, no comporten cap tipus de qualificació. El feedback rebut hauria de servir per reforçar els aprenentatges adquirits i hauria de permetre obtenir un millor resultat en les revisions i entregues que sí que impliquen una qualificació.

  – **Revisions Competencials (RC)**: es tracta de revisions que tenen com a objectiu obtenir evidències de l'**assoliment de competències** per part de l'alumne. Per tant, són revisions durant les quals el professor pot:

    ▷ Plantejar preguntes de qualsevol mena sobre el que s'està revisant.

    ▷ Demanar que es facin petits canvis o modificacions del codi, "en viu i en directe", que permetin certificar que l'alumne comprèn mínimament el funcionament i l'estructura d'allò que s'està presentant.

    ▷ Demanar que s'analitzin alguns aspectes del codi en termes d'eficiència, organització, tolerància a fallades, etc., amb **esperit crític** (és a dir, amb consciència de quins són els punts forts i febles de la solució presentada).

    Aquestes revisions tindran una qualificació numèrica com a resultat, basada en una rúbrica, que estarà en funció del nivell d'assoliment de competències demostrat.

  – **Revisions Qualificatives (RQ)**: es tracta de revisions que tenen com a objectiu obtenir una qualificació numèrica sobre el **resultat** de la feina feta, provant-ne el seu funcionament. Es faran basant-se en una rúbrica per avaluar els mateixos aspectes a tots i cadascun dels alumnes/grups sobre el projecte presentat.