0x02. AirBnB clone - MySQL

Group project **Python** **OOP** **Back-end** **SQL** **MySQL** **ORM** **SQLAlchemy**

Weight: 2

Project to be done in teams of 2 people (your team: Mohamed Madian, Deiaa Elzyat)

Project over - took place from Jan 12, 2024 6:00 AM to Jan 18, 2024 6:00 AM

An auto review will be launched at the deadline

In a nutshell…

- **Auto QA review:** 130.5/191 mandatory
- **Altogether: 68.32%**
  - Mandatory: 68.32%
  - Optional: no optional tasks

Background Context

Environment variables will be your best friend for this project!

- `HBNB_ENV` : running environment. It can be “dev” or “test” for the moment (“production” soon!)
- `HBNB_MYSQL_USER` : the username of your MySQL
- `HBNB_MYSQL_PWD` : the password of your MySQL
- `HBNB_MYSQL_HOST` : the hostname of your MySQL
- `HBNB_MYSQL_DB` : the database name of your MySQL

- `HBNB_TYPE_STORAGE` : the type of storage used. It can be “file” (using `FileStorage`) or `db` (using `DBStorage`)

(/)

Resources

Read or watch:

- cmd module (/rltoken/OG2OW5Pbjs-ds3ZHT0ow4g)
- **packages** concept page
- unittest module (/rltoken/g0tzN6ea1hWCj5OF99HB9w)
- args/kwargs (/rltoken/F6YRBSrkkkTTMVc66iaMgA)
- SQLAlchemy tutorial (/rltoken/GYWCmxokUZKAr-T93iQPcQ)
- How To Create a New User and Grant Permissions in MySQL (/rltoken/m4ogDCoKVm3Us0FybYh1tA)
- Python3 and environment variables (/rltoken/FJCSaX1TCf0HAOzhsH_eWA)
- SQLAlchemy (/rltoken/bWxESLJVYGNonjOYg8fOVg)
- MySQL 8.0 SQL Statement Syntax (/rltoken/n6ePnCDwnbQMbxGgeoe1VA)

Learning Objectives

At the end of this project, you are expected to be able to explain to anyone (/rltoken/3nWKduPHOPRUFGNEtT-SMw), **without the help of Google**:

General

- What is Unit testing and how to implement it in a large project
- What is `*args` and how to use it
- What is `**kwargs` and how to use it
- How to handle named arguments in a function
- How to create a MySQL database
- How to create a MySQL user and grant it privileges
- What ORM means
- How to map a Python Class to a MySQL table
- How to handle 2 different storage engines with the same codebase
- How to use environment variables

Copyright - Plagiarism

- You are tasked to come up with solutions for the tasks below yourself to meet with the above learning objectives.
- You will not be able to meet the objectives of this or any following project by copying and pasting someone else’s work.
- You are not allowed to publish any content of this project.
- Any form of plagiarism is strictly forbidden and will result in removal from the program.

Requirements

Python Scripts

- Allowed editors: `vi` , `vim` , `emacs`

- All your files will be interpreted/compiled on Ubuntu 20.04 LTS using python3 (version 3.8.5)
- All your files should end with a new line
- The first line of all your files should be exactly `#!/usr/bin/python3`
- A `README.md` file, at the root of the folder of the project, is mandatory
- Your code should use the pycodestyle (version `2.8.*`)
- All your files must be executable
- The length of your files will be tested using `wc`
- All your modules should have documentation (`python3 -c 'print(__import__("my_module").__doc__)'`)
- All your classes should have documentation (`python3 -c 'print(__import__("my_module").MyClass.__doc__)'`)
- All your functions (inside and outside a class) should have documentation (`python3 -c 'print(__import__("my_module").my_function.__doc__)'` and `python3 -c 'print(__import__("my_module").MyClass.my_function.__doc__)'`)
- A documentation is not a simple word, it's a real sentence explaining what's the purpose of the module, class or method (the length of it will be verified)

Python Unit Tests

- Allowed editors: `vi`, `vim`, `emacs`
- All your files should end with a new line
- All your test files should be inside a folder `tests`
- You have to use the unittest module (/rltoken/g0tzN6ea1hWCj5OF99HB9w)
- All your test files should be python files (extension: `.py`)
- All your test files and folders should start by `test_`
- Your file organization in the tests folder should be the same as your project: ex: for `models/base_model.py`, unit tests must be in: `tests/test_models/test_base_model.py`
- All your tests should be executed by using this command: `python3 -m unittest discover tests`
- You can also test file by file by using this command: `python3 -m unittest tests/test_models/test_base_model.py`
- All your modules should have documentation (`python3 -c 'print(__import__("my_module").__doc__)'`)
- All your classes should have documentation (`python3 -c 'print(__import__("my_module").MyClass.__doc__)'`)
- All your functions (inside and outside a class) should have documentation (`python3 -c 'print(__import__("my_module").my_function.__doc__)'` and `python3 -c 'print(__import__("my_module").MyClass.my_function.__doc__)'`)
- We strongly encourage you to work together on test cases, so that you don't miss any edge cases

SQL Scripts

- Allowed editors: `vi`, `vim`, `emacs`
- All your files will be executed on Ubuntu 20.04 LTS using `MySQL 8.0`
- Your files will be executed with `SQLAlchemy` version `1.4.x`
- All your files should end with a new line
- All your SQL queries should have a comment just before (i.e. syntax above)
- All your files should start by a comment describing the task
- All SQL keywords should be in uppercase (`SELECT`, `WHERE` ...)
- A `README.md` file, at the root of the folder of the project, is mandatory

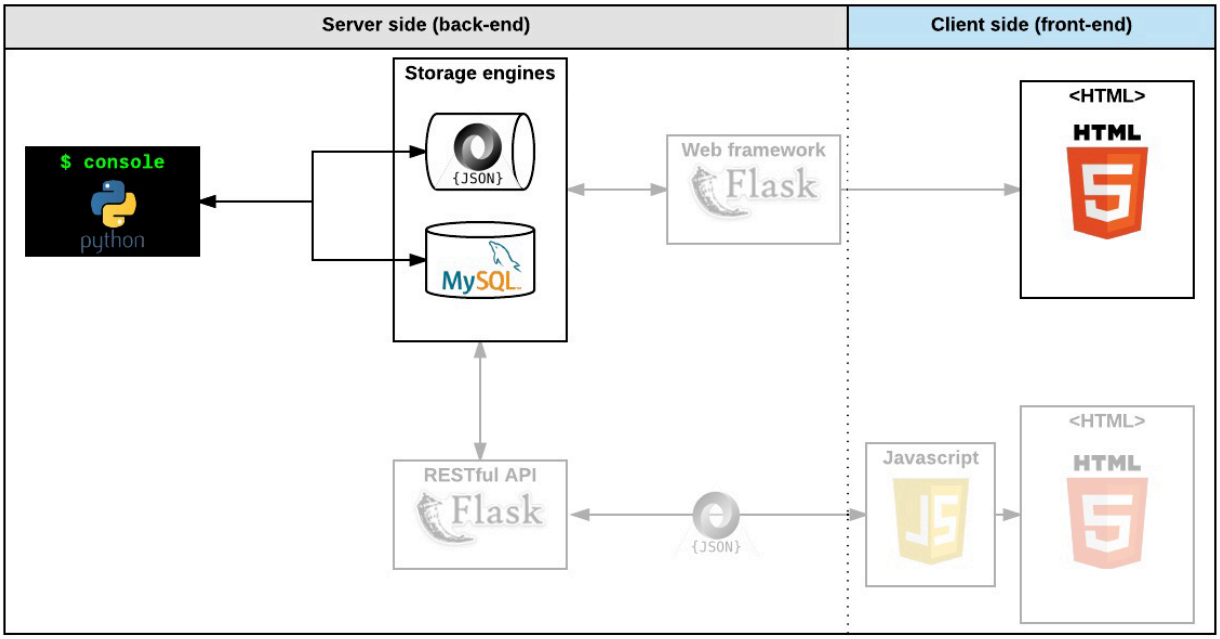
- The length of your files will be tested using `wc`

(/)

GitHub

There should be one project repository per group. If you clone/fork/whatever a partner's project repository with the same name before the second deadline, you risk a 0% score.

More Info



Comments for your SQL file:

```
$ cat my_script.sql
-- first 3 students in the Batch ID=3
-- because Batch 3 is the best!
SELECT id, name FROM students WHERE batch_id = 3 ORDER BY created_at DESC LIMIT 3;
$
```

Video library (2 total)

Search by title

Tasks

0. Fork me if you can!

mandatory

Score: 100.0% (*Checks completed: 100.0%*)

In the industry, you will work on an existing codebase 90% of the time. Your first thoughts upon looking at it might include:

- "Who did this code?"
- "How it works?"
- "Where are unittests?"
- "Where is this?"
- "Why did they do that like this?"
- "I don't understand anything."
- "... I will refactor everything..."

But the worst thing you could possibly do is to **redo everything**. Please don't do that! **Note: the existing codebase might be perfect, or it might have errors. Don't always trust the existing codebase!**

For this project you will fork this codebase (https://github.com/justinmajetich/AirBnB_clone.git):

- update the repository name to `AirBnB_clone_v2`
- update the `README.md` with your information **but don't delete the initial authors**

If you are the owner of this repository, please create a new repository named `AirBnB_clone_v2` with the same content of `AirBnB_clone`

Repo:

- GitHub repository: `AirBnB_clone_v2`

Done! | Check your code | Get a sandbox | QA Review

1. Bug free!

mandatory

Score: 49.58% (*Checks completed: 58.33%*)

Do you remember the `unittest` module?

This codebase contains many test cases. Some are missing, but the ones included cover the basic functionality of the program.

```
guillaume@ubuntu:~/AirBnB_v2$ python3 -m unittest discover tests 2>&1 /dev/null | tail -n 1
OK
guillaume@ubuntu:~/AirBnB_v2$
```

All your unittests **must** pass without any errors at anytime in this project, **with each storage engine!**. Same for PEP8!

```
guillaume@ubuntu:~/AirBnB_v2$ HBNB_ENV=test HBNB_MYSQL_USER=hbnb_test HBNB_MYSQL_PWD=hbnb_te
st_pwd HBNB_MYSQL_HOST=localhost HBNB_MYSQL_DB=hbnb_test_db HBNB_TYPE_STORAGE=db python3 -m
unittest discover tests 2>&1 /dev/null | tail -n 1
OK
guillaume@ubuntu:~/AirBnB_v2$
```

Some tests won't be relevant for some type of storage, please skip them by using the `skipIf` feature of the Unittest module - 26.3.6. Skipping tests and expected failures (/rltoken/Gx_1dJOPPeAyeM6NaXDmjg). Of course, the number of tests must be higher than the current number of tests, so if you decide to skip a test, you should write a new test!

How to test with MySQL?

First, you create a specific database for it (next tasks). After, you have to remember what the purpose of an unittest?

"Assert a current state (objects/data/database), do an action, and validate this action changed (or not) the state of your objects/data/database"

For example, "you want to validate that the `create State name="California"` command in the console will add a new record in your table `states` in your database", here steps for your unittest:

- get the number of current records in the table `states` (my using a `MySQLdb` for example - but not SQLAlchemy (remember, you want to test if it works, so it's better to isolate from the system))
- execute the console command
- get (again) the number of current records in the table `states` (same method, with `MySQLdb`)
- if the difference is `+1` => test passed

Repo:

- GitHub repository: `AirBnB_clone_v2`

☐ Done? | Check your code | Ask for a new correction | QA Review

2. Console improvements

mandatory

Score: 100.0% (*Checks completed: 100.0%*)

Update the `def do_create(self, arg):` function of your command interpreter (`console.py`) to allow for object creation with given parameters:

- Command syntax: `create <Class name> <param 1> <param 2> <param 3>...`
- Param syntax: `<key name>=<value>`
- Value syntax:
  - String: `"<value>"` => starts with a double quote
    - any double quote inside the value must be escaped with a backslash `\`
    - all underscores `_` must be replace by spaces . Example: You want to set the string `My little house` to the attribute `name` , your command line must be `name="My_little_house"`
  - Float: `<unit>.<decimal>` => contains a dot `.`
  - Integer: `<number>` => default case
- If any parameter doesn't fit with these requirements or can't be recognized correctly by your program, it must be skipped

Don't forget to add tests for this new feature!

Also, this new feature will be tested here only with `FileStorage` engine.

```
guillaume@ubuntu:~/AirBnB_v2$ cat test_params_create
create State name="California"
create State name="Arizona"
all State

create Place city_id="0001" user_id="0001" name="My_little_house" number_rooms=4 number_bath
rooms=2 max_guest=10 price_by_night=300 latitude=37.773972 longitude=-122.431297
all Place
guillaume@ubuntu:~/AirBnB_v2$ cat test_params_create | ./console.py
(hbnb) d80e0344-63eb-434a-b1e0-07783522124e
(hbnb) 092c9e5d-6cc0-4eec-aab9-3c1d79cfc2d7
(hbnb) [[State] (d80e0344-63eb-434a-b1e0-07783522124e) {'id': 'd80e0344-63eb-434a-b1e0-07783
522124e', 'created_at': datetime.datetime(2017, 11, 10, 4, 41, 7, 842160), 'updated_at': dat
etime.datetime(2017, 11, 10, 4, 41, 7, 842235), 'name': 'California'}, [State] (092c9e5d-6cc
0-4eec-aab9-3c1d79cfc2d7) {'id': '092c9e5d-6cc0-4eec-aab9-3c1d79cfc2d7', 'created_at': datet
ime.datetime(2017, 11, 10, 4, 41, 7, 842779), 'updated_at': datetime.datetime(2017, 11, 10,
4, 41, 7, 842792), 'name': 'Arizona'}]
(hbnb) (hbnb) 76b65327-9e94-4632-b688-aaa22ab8a124
(hbnb) [[Place] (76b65327-9e94-4632-b688-aaa22ab8a124) {'number_bathrooms': 2, 'longitude':
-122.431297, 'city_id': '0001', 'user_id': '0001', 'latitude': 37.773972, 'price_by_night':
300, 'name': 'My little house', 'id': '76b65327-9e94-4632-b688-aaa22ab8a124', 'max_guest': 1
0, 'number_rooms': 4, 'updated_at': datetime.datetime(2017, 11, 10, 4, 41, 7, 843774), 'crea
ted_at': datetime.datetime(2017, 11, 10, 4, 41, 7, 843747)}]
(hbnb)
guillaume@ubuntu:~/AirBnB_v2$
```

Repo:

- GitHub repository: `AirBnB_clone_v2`
- File: `console.py, models/, tests/`

Done! | Check your code | Get a sandbox | QA Review

3. MySQL setup development

mandatory

Score: 65.0% (*Checks completed: 100.0%*)

Write a script that prepares a MySQL server for the project:

- A database `hbnb_dev_db`
- A new user `hbnb_dev` (in `localhost`)
- The password of `hbnb_dev` should be set to `hbnb_dev_pwd`
- `hbnb_dev` should have all privileges on the database `hbnb_dev_db` (and **only this database**)
- `hbnb_dev` should have `SELECT` privilege on the database `performance_schema` (and **only this database**)
- If the database `hbnb_dev_db` or the user `hbnb_dev` already exists, your script should not fail

```
guillaume@ubuntu:~/AirBnB_v2$ cat setup_mysql_dev.sql | mysql -hlocalhost -uroot -p
Enter password:
guillaume@ubuntu:~/AirBnB_v2$ echo "SHOW DATABASES;" | mysql -uhbnb_dev -p | grep hbnb_dev_d
b
Enter password:
hbnb_dev_db
guillaume@ubuntu:~/AirBnB_v2$ echo "SHOW GRANTS FOR 'hbnb_dev'@'localhost';" | mysql -uroot
-p
Enter password:
Grants for hbnb_dev@localhost
GRANT USAGE ON *.* TO 'hbnb_dev'@'localhost'
GRANT SELECT ON `performance_schema`.* TO 'hbnb_dev'@'localhost'
GRANT ALL PRIVILEGES ON `hbnb_dev_db`.* TO 'hbnb_dev'@'localhost'
guillaume@ubuntu:~/AirBnB_v2$
```

Repo:

- GitHub repository: `AirBnB_clone_v2`
- File: `setup_mysql_dev.sql`

Done! | Check your code | Get a sandbox | QA Review

4. MySQL setup test

mandatory

Score: 65.0% (*Checks completed: 100.0%*)

Write a script that prepares a MySQL server for the project:

- A database `hbnb_test_db`
- A new user `hbnb_test` (in `localhost`)
- The password of `hbnb_test` should be set to `hbnb_test_pwd`
- `hbnb_test` should have all privileges on the database `hbnb_test_db` (and **only this database**)
- `hbnb_test` should have `SELECT` privilege on the database `performance_schema` (and **only this database**)
- If the database `hbnb_test_db` or the user `hbnb_test` already exists, your script should not fail

```
guillaume@ubuntu:~/AirBnB_v2$ cat setup_mysql_test.sql | mysql -hlocalhost -uroot -p
Enter password:
guillaume@ubuntu:~/AirBnB_v2$ echo "SHOW DATABASES;" | mysql -uhbnb_test -p | grep hbnb_test
_db
Enter password:
hbnb_test_db
guillaume@ubuntu:~/AirBnB_v2$ echo "SHOW GRANTS FOR 'hbnb_test'@'localhost';" | mysql -uroot
-p
Enter password:
Grants for hbnb_test@localhost
GRANT USAGE ON *.* TO 'hbnb_test'@'localhost'
GRANT SELECT ON `performance_schema`.* TO 'hbnb_test'@'localhost'
GRANT ALL PRIVILEGES ON `hbnb_test_db`.* TO 'hbnb_test'@'localhost'
guillaume@ubuntu:~/AirBnB_v2$
```

Repo:

- GitHub repository: `AirBnB_clone_v2`
- File: `setup_mysql_test.sql`

Done! | Check your code | Get a sandbox | QA Review

5. Delete object

mandatory

Score: 65.0% (*Checks completed: 100.0%*)

Update `FileStorage` : (`models/engine/file_storage.py`)

- Add a new public instance method: `def delete(self, obj=None):` to delete `obj` from `__objects` if it's inside - if `obj` is equal to `None` , the method should not do anything
- Update the prototype of `def all(self)` to `def all(self, cls=None)` - that returns the list of objects of one type of class. Example below with `State` - it's an optional filtering

7)

```
guillaume@ubuntu:~/AirBnB_v2$ cat main_delete.py
#!/usr/bin/python3
""" Test delete feature
"""
from models.engine.file_storage import FileStorage
from models.state import State

fs = FileStorage()

# All States
all_states = fs.all(State)
print("All States: {}".format(len(all_states.keys())))
for state_key in all_states.keys():
    print(all_states[state_key])

# Create a new State
new_state = State()
new_state.name = "California"
fs.new(new_state)
fs.save()
print("New State: {}".format(new_state))

# All States
all_states = fs.all(State)
print("All States: {}".format(len(all_states.keys())))
for state_key in all_states.keys():
    print(all_states[state_key])

# Create another State
another_state = State()
another_state.name = "Nevada"
fs.new(another_state)
fs.save()
print("Another State: {}".format(another_state))

# All States
all_states = fs.all(State)
print("All States: {}".format(len(all_states.keys())))
for state_key in all_states.keys():
    print(all_states[state_key])

# Delete the new State
fs.delete(new_state)

# All States
all_states = fs.all(State)
print("All States: {}".format(len(all_states.keys())))
for state_key in all_states.keys():
    print(all_states[state_key])

guillaume@ubuntu:~/AirBnB_v2$ ./main_delete.py
All States: 0
```

```
New State: [State] (b0026fc6-116f-4d1a-a9cb-6bb9b299f1ce) {'name': 'California', 'created_a
t': datetime.datetime(2017, 11, 10, 1, 13, 32, 561137), 'id': 'b0026fc6-116f-4d1a-a9cb-6bb9b
299f1ce'}
All States: 1
[State] (b0026fc6-116f-4d1a-a9cb-6bb9b299f1ce) {'name': 'California', 'created_at': datetim
e.datetime(2017, 11, 10, 1, 13, 32, 561137), 'id': 'b0026fc6-116f-4d1a-a9cb-6bb9b299f1ce'}
Another State: [State] (37705d25-8903-4318-9303-6d6d336a22c1) {'name': 'Nevada', 'created_a
t': datetime.datetime(2017, 11, 10, 1, 13, 34, 619133), 'id': '37705d25-8903-4318-9303-6d6d3
36a22c1'}
All States: 2
[State] (b0026fc6-116f-4d1a-a9cb-6bb9b299f1ce) {'name': 'California', 'created_at': datetim
e.datetime(2017, 11, 10, 1, 13, 32, 561137), 'id': 'b0026fc6-116f-4d1a-a9cb-6bb9b299f1ce'}
[State] (37705d25-8903-4318-9303-6d6d336a22c1) {'name': 'Nevada', 'created_at': datetime.dat
etime(2017, 11, 10, 1, 13, 34, 619133), 'id': '37705d25-8903-4318-9303-6d6d336a22c1'}
All States: 1
[State] (37705d25-8903-4318-9303-6d6d336a22c1) {'name': 'Nevada', 'created_at': datetime.dat
etime(2017, 11, 10, 1, 13, 34, 619133), 'id': '37705d25-8903-4318-9303-6d6d336a22c1'}
guillaume@ubuntu:~/AirBnB_v2$
```

Repo:

- GitHub repository: `AirBnB_clone_v2`
- File: `models/engine/file_storage.py`

Done! | Check your code | Get a sandbox | QA Review

6. DBStorage - States and Cities

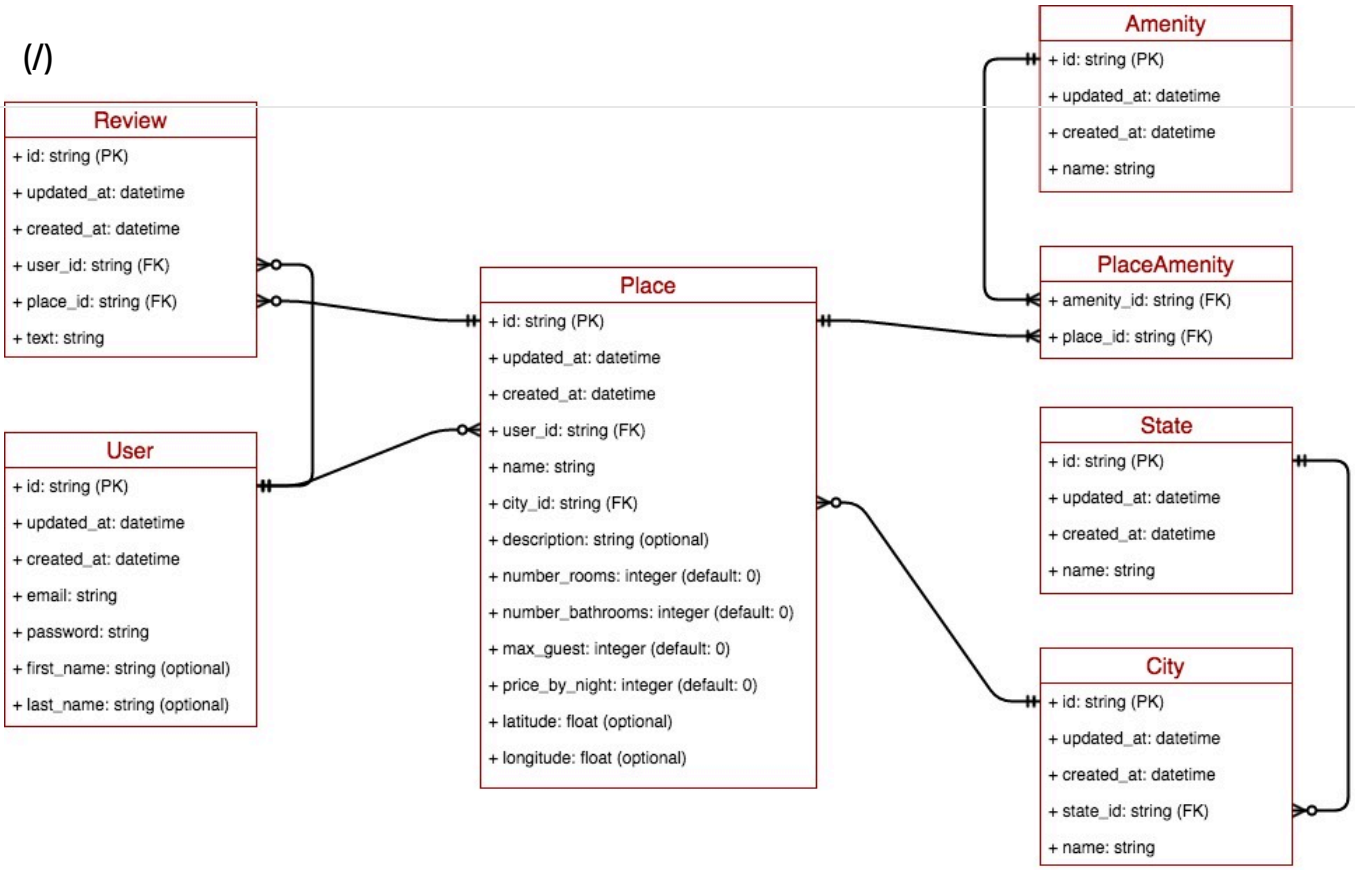
mandatory

Score: 65.0% (*Checks completed: 100.0%*)

SQLAlchemy will be your best friend!

It's time to change your storage engine and use `SQLAlchemy`

(/)



In the following steps, you will make multiple changes:

- the biggest one is the transition between `FileStorage` and `DBStorage` : In the industry, you will never find a system who can work with both in the same time - but you will find a lot of services who can manage multiple storage systems. (for example, logs service: in memory, in disk, in database, in ElasticSearch etc…) - The main concept behind is the **abstraction**: Make your code running without knowing how it's stored.
- add attributes for SQLAlchemy: they will be class attributes, like previously, with a "weird" value. Don't worry, these values are for description and mapping to the database. If you change one of these values, or add/remove one attribute of the a model, you will have to delete the database and recreate it in SQL. (Yes it's not optimal, but for development purposes, it's ok. In production, we will add "migration mechanism" - for the moment, don't spend time on it.)

Please follow all these steps:

Update `BaseModel` : (`models/base_model.py`)

- Create `Base = declarative_base()` before the class definition of `BaseModel`
- **Note! BaseModel does /not/ inherit from Base. All other classes will inherit from BaseModel to get common values (id, `created_at` , `updated_at`), where inheriting from Base will actually cause SQLAlchemy to attempt to map it to a table.**
- Add or replace in the class `BaseModel` :
  - class attribute `id`
    - represents a column containing a unique string (60 characters)
    - can't be null
    - primary key
  - class attribute `created_at`
    - represents a column containing a datetime

        - can't be null
        - default value is the current datetime (use `datetime.utcnow()`)
    - class attribute `updated_at`
        - represents a column containing a datetime
        - can't be null
        - default value is the current datetime (use `datetime.utcnow()`)
- Move the `models.storage.new(self)` from `def __init__(self, *args, **kwargs):` to `def save(self):` and call it just before `models.storage.save()`
- In `def __init__(self, *args, **kwargs):` , manage `kwargs` to create instance attribute from this dictionary. Ex: `kwargs={ 'name': "California" } => self.name = "California"` if it's not already the case
- Update the `to_dict()` method of the class `BaseModel` :
    - remove the key `_sa_instance_state` from the dictionary returned by this method **only if this key exists**
- Add a new public instance method: `def delete(self):` to delete the current instance from the storage (`models.storage`) by calling the method `delete`

Update `City` : (`models/city.py`)

- `City` inherits from `BaseModel` and `Base` (respect the order)
- Add or replace in the class `City` :
    - class attribute `__tablename__` -
        - represents the table name, `cities`
    - class attribute `name`
        - represents a column containing a string (128 characters)
        - can't be null
    - class attribute `state_id`
        - represents a column containing a string (60 characters)
        - can't be null
        - is a foreign key to `states.id`

Update `State` : (`models/state.py`)

- `State` inherits from `BaseModel` and `Base` (respect the order)
- Add or replace in the class `State` :
    - class attribute `__tablename__`
        - represents the table name, `states`
    - class attribute `name`
        - represents a column containing a string (128 characters)
        - can't be null
    - for `DBStorage` : class attribute `cities` must represent a relationship with the class `City` . If the `State` object is deleted, all linked `City` objects must be automatically deleted. Also, the reference from a `City` object to his `State` should be named `state`
    - for `FileStorage` : getter attribute `cities` that returns the list of `City` instances with `state_id` equals to the current `State.id` => It will be the `FileStorage` relationship between `State` and `City`

New engine `DBStorage` : (`models/engine/db_storage.py`)

- Private class attributes:
    - `__engine` : set to `None`
    - `__session` : set to `None`

- Public instance methods:
  - `__init__(self)`:
    - create the engine (`self.__engine`)
    - the engine must be linked to the MySQL database and user created before (`hbnb_dev` and `hbnb_dev_db`):
      - dialect: `mysql`
      - driver: `mysqldb`
    - all of the following values must be retrieved via environment variables:
      - MySQL user: `HBNB_MYSQL_USER`
      - MySQL password: `HBNB_MYSQL_PWD`
      - MySQL host: `HBNB_MYSQL_HOST` (here = `localhost`)
      - MySQL database: `HBNB_MYSQL_DB`
    - don't forget the option `pool_pre_ping=True` when you call `create_engine`
    - drop all tables if the environment variable `HBNB_ENV` is equal to `test`
  - `all(self, cls=None)` :
    - query on the current database session (`self.__session`) all objects depending of the class name (argument `cls`)
    - if `cls=None` , query all types of objects (`User` , `State` , `City` , `Amenity` , `Place` and `Review`)
    - this method must return a dictionary: (like `FileStorage`)
      - key = `<class-name>.<object-id>`
      - value = object
  - `new(self, obj)` : add the object to the current database session (`self.__session`)
  - `save(self)` : commit all changes of the current database session (`self.__session`)
  - `delete(self, obj=None)` : delete from the current database session `obj` if not `None`
  - `reload(self)` :
    - create all tables in the database (feature of SQLAlchemy) (WARNING: all classes who inherit from `Base` must be imported before calling `Base.metadata.create_all(engine)`)
    - create the current database session (`self.__session`) from the engine (`self.__engine`) by using a sessionmaker (/rltoken/FhKkGICnmM0DN4TrIfJw3g) - the option `expire_on_commit` must be set to `False` ; and scoped_session (/rltoken/kSil6Cs0L7bWTXNJTrk9yw) - to make sure your Session is thread-safe

Update `__init__.py` : (`models/__init__.py`)

- Add a conditional depending of the value of the environment variable `HBNB_TYPE_STORAGE` :
  - If equal to `db` :
    - Import `DBStorage` class in this file
    - Create an instance of `DBStorage` and store it in the variable `storage` (the line `storage.reload()` should be executed after this instantiation)
  - Else:
    - Import `FileStorage` class in this file
    - Create an instance of `FileStorage` and store it in the variable `storage` (the line `storage.reload()` should be executed after this instantiation)
- This "switch" will allow you to change storage type directly by using an environment variable (example below)

State creation:

(7)

```
guillaume@ubuntu:~/AirBnB_v2$ echo 'create State name="California"' | HBNB_MYSQL_USER=hbnb_d
ev HBNB_MYSQL_PWD=hbnb_dev_pwd HBNB_MYSQL_HOST=localhost HBNB_MYSQL_DB=hbnb_dev_db HBNB_TYPE
_STORAGE=db ./console.py
(hbnb) 95a5abab-aa65-4861-9bc6-1da4a36069aa
(hbnb)
guillaume@ubuntu:~/AirBnB_v2$
guillaume@ubuntu:~/AirBnB_v2$ echo 'all State' | HBNB_MYSQL_USER=hbnb_dev HBNB_MYSQL_PWD=hbn
b_dev_pwd HBNB_MYSQL_HOST=localhost HBNB_MYSQL_DB=hbnb_dev_db HBNB_TYPE_STORAGE=db ./consol
e.py
(hbnb) [[State] (95a5abab-aa65-4861-9bc6-1da4a36069aa) {'name': 'California', 'id': '95a5aba
b-aa65-4861-9bc6-1da4a36069aa', 'updated_at': datetime.datetime(2017, 11, 10, 0, 49, 54), 'c
reated_at': datetime.datetime(2017, 11, 10, 0, 49, 54)}]
(hbnb)
guillaume@ubuntu:~/AirBnB_v2$
guillaume@ubuntu:~/AirBnB_v2$ echo 'SELECT * FROM states\G' | mysql -uhbnb_dev -p hbnb_dev_d
b
Enter password:
*************************** 1. row ***************************
        id: 95a5abab-aa65-4861-9bc6-1da4a36069aa
created_at: 2017-11-10 00:49:54
updated_at: 2017-11-10 00:49:54
      name: California
guillaume@ubuntu:~/AirBnB_v2$
```

City creation:

```
guillaume@ubuntu:~/AirBnB_v2$ echo 'create City state_id="95a5abab-aa65-4861-9bc6-1da4a36069
aa" name="San_Francisco"' | HBNB_MYSQL_USER=hbnb_dev HBNB_MYSQL_PWD=hbnb_dev_pwd HBNB_MYSQL_
HOST=localhost HBNB_MYSQL_DB=hbnb_dev_db HBNB_TYPE_STORAGE=db ./console.py
(hbnb) 4b457e66-c7c8-4f63-910f-fd91c3b7140b
(hbnb)
guillaume@ubuntu:~/AirBnB_v2$
guillaume@ubuntu:~/AirBnB_v2$ echo 'all City' | HBNB_MYSQL_USER=hbnb_dev HBNB_MYSQL_PWD=hbnb
_dev_pwd HBNB_MYSQL_HOST=localhost HBNB_MYSQL_DB=hbnb_dev_db HBNB_TYPE_STORAGE=db ./console.
py
(hbnb) [[City] (4b457e66-c7c8-4f63-910f-fd91c3b7140b) {'id': '4b457e66-c7c8-4f63-910f-fd91c3
b7140b', 'updated_at': datetime.datetime(2017, 11, 10, 0, 52, 53), 'state_id': '95a5abab-aa6
5-4861-9bc6-1da4a36069aa', 'name': 'San Francisco', 'created_at': datetime.datetime(2017, 1
1, 10, 0, 52, 53)]
(hbnb)
guillaume@ubuntu:~/AirBnB_v2$
```

(7)

```
guillaume@ubuntu:~/AirBnB_v2$ echo 'create City state_id="95a5abab-aa65-4861-9bc6-1da4a36069
aa" name="San_Jose"' | HBNB_MYSQL_USER=hbnb_dev HBNB_MYSQL_PWD=hbnb_dev_pwd HBNB_MYSQL_HOST=
localhost HBNB_MYSQL_DB=hbnb_dev_db HBNB_TYPE_STORAGE=db ./console.py
(hbnb) a7db3cdc-30e0-4d80-ad8c-679fe45343ba
(hbnb)
guillaume@ubuntu:~/AirBnB_v2$
guillaume@ubuntu:~/AirBnB_v2$ echo 'SELECT * FROM cities\G' | mysql -uhbnb_dev -p hbnb_dev_d
b
Enter password:
*************************** 1. row ***************************
        id: 4b457e66-c7c8-4f63-910f-fd91c3b7140b
created_at: 2017-11-10 00:52:53
updated_at: 2017-11-10 00:52:53
      name: San Francisco
  state_id: 95a5abab-aa65-4861-9bc6-1da4a36069aa
*************************** 2. row ***************************
        id: a7db3cdc-30e0-4d80-ad8c-679fe45343ba
created_at: 2017-11-10 00:53:19
updated_at: 2017-11-10 00:53:19
      name: San Jose
  state_id: 95a5abab-aa65-4861-9bc6-1da4a36069aa
guillaume@ubuntu:~/AirBnB_v2$
```

Repo:

- GitHub repository: `AirBnB_clone_v2`
- File: `models/base_model.py, models/city.py, models/state.py, models/engine/db_storage.py, models/__init__.py`

☑ Done! | Check your code | QA Review

7. DBStorage - User

mandatory

Score: 65.0% (*Checks completed: 100.0%*)

Update `User` : (`models/user.py`)

- `User` inherits from `BaseModel` and `Base` (respect the order)
- Add or replace in the class `User` :
  - class attribute `__tablename__`
    - represents the table name, `users`
  - class attribute `email`
    - represents a column containing a string (128 characters)
    - can't be null
  - class attribute `password`
    - represents a column containing a string (128 characters)
    - can't be null

(/)

- class attribute `first_name`
  - represents a column containing a string (128 characters)
  - can be null
- class attribute `last_name`
  - represents a column containing a string (128 characters)
  - can be null

```
guillaume@ubuntu:~/AirBnB_v2$ echo 'create User email="gui@hbtn.io" password="guipwd" first_
name="Guillaume" last_name="Snow"' | HBNB_MYSQL_USER=hbnb_dev HBNB_MYSQL_PWD=hbnb_dev_pwd HB
NB_MYSQL_HOST=localhost HBNB_MYSQL_DB=hbnb_dev_db HBNB_TYPE_STORAGE=db ./console.py
(hbnb) 4f3f4b42-a4c3-4c20-a492-efff10d00c0b
(hbnb)
guillaume@ubuntu:~/AirBnB_v2$
guillaume@ubuntu:~/AirBnB_v2$ echo 'all User' | HBNB_MYSQL_USER=hbnb_dev HBNB_MYSQL_PWD=hbnb
_dev_pwd HBNB_MYSQL_HOST=localhost HBNB_MYSQL_DB=hbnb_dev_db HBNB_TYPE_STORAGE=db ./console.
py
(hbnb) [[User] (4f3f4b42-a4c3-4c20-a492-efff10d00c0b) {'updated_at': datetime.datetime(2017,
11, 10, 1, 17, 26), 'id': '4f3f4b42-a4c3-4c20-a492-efff10d00c0b', 'last_name': 'Snow', 'firs
t_name': 'Guillaume', 'email': 'gui@hbtn.io', 'created_at': datetime.datetime(2017, 11, 10,
1, 17, 26), 'password': 'f4ce007d8e84e0910fbdd7a06fa1692d'}]
(hbnb)
guillaume@ubuntu:~/AirBnB_v2$
guillaume@ubuntu:~/AirBnB_v2$ echo 'SELECT * FROM users\G' | mysql -uhbnb_dev -p hbnb_dev_db
Enter password:
*************************** 1. row ***************************
        id: 4f3f4b42-a4c3-4c20-a492-efff10d00c0b
created_at: 2017-11-10 01:17:26
updated_at: 2017-11-10 01:17:26
     email: gui@hbtn.io
  password: guipwd
first_name: Guillaume
 last_name: Snow
guillaume@ubuntu:~/AirBnB_v2$
```

Repo:

- GitHub repository: `AirBnB_clone_v2`
- File: `models/user.py`

☑ Done! | Check your code | QA Review

8. DBStorage - Place

mandatory

Score: 65.0% (*Checks completed: 100.0%*)

Update `Place` : (`models/place.py`)

- `Place` inherits from `BaseModel` and `Base` (respect the order)
- Add or replace in the class `Place` :
  - class attribute `__tablename__`
    - represents the table name, `places`
  - class attribute `city_id`
    - represents a column containing a string (60 characters)
    - can't be null
    - is a foreign key to `cities.id`
  - class attribute `user_id`
    - represents a column containing a string (60 characters)
    - can't be null
    - is a foreign key to `users.id`
  - class attribute `name`
    - represents a column containing a string (128 characters)
    - can't be null
  - class attribute `description`
    - represents a column containing a string (1024 characters)
    - can be null
  - class attribute `number_rooms`
    - represents a column containing an integer
    - can't be null
    - default value: `0`
  - class attribute `number_bathrooms`
    - represents a column containing an integer
    - can't be null
    - default value: `0`
  - class attribute `max_guest`
    - represents a column containing an integer
    - can't be null
    - default value: `0`
  - class attribute `price_by_night`
    - represents a column containing an integer
    - can't be null
    - default value: `0`
  - class attribute `latitude`
    - represents a column containing a float
    - can be null
  - class attribute `longitude`
    - represents a column containing a float
    - can be null

Update `User` : (`models/user.py`)

- Add or replace in the class `User` :
  - class attribute `places` must represent a relationship with the class `Place` . If the `User` object is deleted, all linked `Place` objects must be automatically deleted. Also, the reference from a `Place` object to his `User` should be named `user`

Update `City` : (`models/city.py`)

- Add or replace in the class `City` :

(/)

- class attribute `places` must represent a relationship with the class `Place`. If the `City` object is deleted, all linked `Place` objects must be automatically deleted. Also, the reference from a `Place` object to his `City` should be named `cities`

```
guillaume@ubuntu:~/AirBnB_v2$ echo 'create Place city_id="4b457e66-c7c8-4f63-910f-fd91c3b714
0b" user_id="4f3f4b42-a4c3-4c20-a492-efff10d00c0b" name="Lovely_place" number_rooms=3 number
_bathrooms=1 max_guest=6 price_by_night=120 latitude=37.773972 longitude=-122.431297' | HBNB
_MYSQL_USER=hbnb_dev HBNB_MYSQL_PWD=hbnb_dev_pwd HBNB_MYSQL_HOST=localhost HBNB_MYSQL_DB=hbn
b_dev_db HBNB_TYPE_STORAGE=db ./console.py
(hbnb) ed72aa02-3286-4891-acbc-9d9fc80a1103
(hbnb)
guillaume@ubuntu:~/AirBnB_v2$
guillaume@ubuntu:~/AirBnB_v2$ echo 'all Place' | HBNB_MYSQL_USER=hbnb_dev HBNB_MYSQL_PWD=hbn
b_dev_pwd HBNB_MYSQL_HOST=localhost HBNB_MYSQL_DB=hbnb_dev_db HBNB_TYPE_STORAGE=db ./consol
e.py
(hbnb) [[Place] (ed72aa02-3286-4891-acbc-9d9fc80a1103) {'latitude': 37.774, 'city_id': '4b45
7e66-c7c8-4f63-910f-fd91c3b7140b', 'price_by_night': 120, 'id': 'ed72aa02-3286-4891-acbc-9d9
fc80a1103', 'user_id': '4f3f4b42-a4c3-4c20-a492-efff10d00c0b', 'max_guest': 6, 'created_at':
datetime.datetime(2017, 11, 10, 1, 22, 30), 'description': None, 'number_rooms': 3, 'longitu
de': -122.431, 'number_bathrooms': 1, 'name': '"Lovely place', 'updated_at': datetime.dateti
me(2017, 11, 10, 1, 22, 30)}]
(hbnb)
guillaume@ubuntu:~/AirBnB_v2$
guillaume@ubuntu:~/AirBnB_v2$ echo 'SELECT * FROM places\G' | mysql -uhbnb_dev -p hbnb_dev_d
b
Enter password:
*************************** 1. row ***************************
              id: ed72aa02-3286-4891-acbc-9d9fc80a1103
      created_at: 2017-11-10 01:22:30
      updated_at: 2017-11-10 01:22:30
         city_id: 4b457e66-c7c8-4f63-910f-fd91c3b7140b
         user_id: 4f3f4b42-a4c3-4c20-a492-efff10d00c0b
            name: "Lovely place"
     description: NULL
    number_rooms: 3
number_bathrooms: 1
       max_guest: 6
  price_by_night: 120
        latitude: 37.774
       longitude: -122.431
guillaume@ubuntu:~/AirBnB_v2$
```

Repo:

- GitHub repository: `AirBnB_clone_v2`
- File: `models/place.py, models/user.py, models/city.py`

Done! | Check your code | QA Review

9. DBStorage - Review

mandatory

Score: 65.0% (*Checks completed: 100.0%*)

Update `Review` : (`models/review.py`)

- `Review` inherits from `BaseModel` and `Base` (respect the order)
- Add or replace in the class `Review` :
  - class attribute `__tablename__`
    - represents the table name, `reviews`
  - class attribute `text`
    - represents a column containing a string (1024 characters)
    - can't be null
  - class attribute `place_id`
    - represents a column containing a string (60 characters)
    - can't be null
    - is a foreign key to `places.id`
  - class attribute `user_id`
    - represents a column containing a string (60 characters)
    - can't be null
    - is a foreign key to `users.id`

Update `User` : (`models/user.py`)

- Add or replace in the class `User` :
  - class attribute `reviews` must represent a relationship with the class `Review` . If the `User` object is deleted, all linked `Review` objects must be automatically deleted. Also, the reference from a `Review` object to his `User` should be named `user`

Update `Place` : (`models/place.py`)

- for `DBStorage` : class attribute `reviews` must represent a relationship with the class `Review` . If the `Place` object is deleted, all linked `Review` objects must be automatically deleted. Also, the reference from a `Review` object to his `Place` should be named `place`
- for `FileStorage` : getter attribute `reviews` that returns the list of `Review` instances with `place_id` equals to the current `Place.id` => It will be the `FileStorage` relationship between `Place` and `Review`

(7)

```
guillaume@ubuntu:~/AirBnB_v2$
guillaume@ubuntu:~/AirBnB_v2$ echo 'create User email="bob@hbtn.io" password="bobpwd" first_
name="Bob" last_name="Dylan"' | HBNB_MYSQL_USER=hbnb_dev HBNB_MYSQL_PWD=hbnb_dev_pwd HBNB_MY
SQL_HOST=localhost HBNB_MYSQL_DB=hbnb_dev_db HBNB_TYPE_STORAGE=db ./console.py
(hbnb) d93638d9-8233-4124-8f4e-17786592908b
(hbnb)
guillaume@ubuntu:~/AirBnB_v2$
guillaume@ubuntu:~/AirBnB_v2$ echo 'create Review place_id="ed72aa02-3286-4891-acbc-9d9fc80a
1103" user_id="d93638d9-8233-4124-8f4e-17786592908b" text="Amazing_place,_huge_kitchen"' | H
BNB_MYSQL_USER=hbnb_dev HBNB_MYSQL_PWD=hbnb_dev_pwd HBNB_MYSQL_HOST=localhost HBNB_MYSQL_DB=
hbnb_dev_db HBNB_TYPE_STORAGE=db ./console.py
(hbnb) a2d163d3-1982-48ab-a06b-9dc71e68a791
(hbnb)
guillaume@ubuntu:~/AirBnB_v2$
guillaume@ubuntu:~/AirBnB_v2$ echo 'all Review' | HBNB_MYSQL_USER=hbnb_dev HBNB_MYSQL_PWD=hb
nb_dev_pwd HBNB_MYSQL_HOST=localhost HBNB_MYSQL_DB=hbnb_dev_db HBNB_TYPE_STORAGE=db ./consol
e.py
(hbnb) [[Review] (f2616ff2-f723-4d67-85dc-f050a38e0f2f) {'text': 'Amazing place, huge kitche
n', 'place_id': 'ed72aa02-3286-4891-acbc-9d9fc80a1103', 'id': 'f2616ff2-f723-4d67-85dc-f050a
38e0f2f', 'updated_at': datetime.datetime(2017, 11, 10, 4, 6, 25), 'created_at': datetime.da
tetime(2017, 11, 10, 4, 6, 25), 'user_id': 'd93638d9-8233-4124-8f4e-17786592908b'}]
(hbnb)
guillaume@ubuntu:~/AirBnB_v2$
guillaume@ubuntu:~/AirBnB_v2$ echo 'SELECT * FROM reviews\G' | mysql -uhbnb_dev -p hbnb_dev_
db
Enter password:
*************************** 1. row ***************************
        id: f2616ff2-f723-4d67-85dc-f050a38e0f2f
created_at: 2017-11-10 04:06:25
updated_at: 2017-11-10 04:06:25
      text: Amazing place, huge kitchen
  place_id: ed72aa02-3286-4891-acbc-9d9fc80a1103
   user_id: d93638d9-8233-4124-8f4e-17786592908b
guillaume@ubuntu:~/AirBnB_v2$
```

Repo:

- GitHub repository: `AirBnB_clone_v2`
- File: `models/review.py, models/user.py, models/place.py`

☑ Done! | Check your code | QA Review

10. DBStorage - Amenity... and BOOM!

mandatory

Score: 65.0% (*Checks completed: 100.0%*)

Update `Amenity` : (`models/amenity.py`)

- `Amenity` inherits from `BaseModel` and `Base` (respect the order)
- Add or replace in the class `Amenity` :
  - class attribute `__tablename__`
    - represents the table name, `amenities`
  - class attribute `name`
    - represents a column containing a string (128 characters)
    - can't be null
  - class attribute `place_amenities` must represent a relationship Many-To-Many (/rltoken/LiU5umFamh-YbwWkgd8kNw) between the class `Place` and `Amenity` . Please see below more detail: `place_amenity` in the `Place` update

Update `Place` : (`models/place.py`)

- Add an instance of SQLAlchemy Table (/rltoken/Pngd-iHVu-kUMyq5VaC9PQ) called `place_amenity` for creating the relationship Many-To-Many (/rltoken/LiU5umFamh-YbwWkgd8kNw) between `Place` and `Amenity` :
  - table name `place_amenity`
  - `metadata = Base.metadata`
  - 2 columns:
    - `place_id` , a string of 60 characters foreign key of `places.id` , primary key in the table and never null
    - `amenity_id` , a string of 60 characters foreign key of `amenities.id` , primary key in the table and never null
- Update `Place` class:
  - for `DBStorage` : class attribute `amenities` must represent a relationship with the class `Amenity` but also as `secondary` to `place_amenity` with option `viewonly=False` (`place_amenity` has been define previously)
  - for `FileStorage` :
    - Getter attribute `amenities` that returns the list of `Amenity` instances based on the attribute `amenity_ids` that contains all `Amenity.id` linked to the `Place`
    - Setter attribute `amenities` that handles `append` method for adding an `Amenity.id` to the attribute `amenity_ids` . This method should accept only `Amenity` object, otherwise, do nothing.

What's a `Many-to-Many` relationship?

In our system, we don't want to duplicate amenities (for example, having 10000 time the amenity `Wifi`), so they will be unique. But, at least 2 places can have the same amenity (like `Wifi` for example). We are in the case of:

- an amenity can be linked to multiple places
- a place can have multiple amenities

`= Many-To-Many`

To make this link working, we will create a third table called `place_amenity` that will create these links.

And you are good, you have a new engine!

(7)

```
guillaume@ubuntu:~/AirBnB_v2$ cat main_place_amenities.py
#!/usr/bin/python3
""" Test link Many-To-Many Place <> Amenity
"""
from models import *

# creation of a State
state = State(name="California")
state.save()

# creation of a City
city = City(state_id=state.id, name="San Francisco")
city.save()

# creation of a User
user = User(email="john@snow.com", password="johnpwd")
user.save()

# creation of 2 Places
place_1 = Place(user_id=user.id, city_id=city.id, name="House 1")
place_1.save()
place_2 = Place(user_id=user.id, city_id=city.id, name="House 2")
place_2.save()

# creation of 3 various Amenity
amenity_1 = Amenity(name="Wifi")
amenity_1.save()
amenity_2 = Amenity(name="Cable")
amenity_2.save()
amenity_3 = Amenity(name="Oven")
amenity_3.save()

# link place_1 with 2 amenities
place_1.amenities.append(amenity_1)
place_1.amenities.append(amenity_2)

# link place_2 with 3 amenities
place_2.amenities.append(amenity_1)
place_2.amenities.append(amenity_2)
place_2.amenities.append(amenity_3)

storage.save()

print("OK")

guillaume@ubuntu:~/AirBnB_v2$
guillaume@ubuntu:~/AirBnB_v2$ HBNB_MYSQL_USER=hbnb_dev HBNB_MYSQL_PWD=hbnb_dev_pwd HBNB_MYSQ
L_HOST=localhost HBNB_MYSQL_DB=hbnb_dev_db HBNB_TYPE_STORAGE=db ./main_place_amenities.py
OK
guillaume@ubuntu:~/AirBnB_v2$
guillaume@ubuntu:~/AirBnB_v2$ echo 'SELECT * FROM amenities\G' | mysql -uhbnb_dev -p hbnb_de
v_db
```

```
Enter password:
*************************** 1. row ***************************
        id: 47321eb8-152a-46df-969a-440aa67a6d59
created_at: 2017-11-10 04:22:02
updated_at: 2017-11-10 04:22:02
      name: Cable
*************************** 2. row ***************************
        id: 4a307e7f-68f9-438f-81c0-8325898dda2a
created_at: 2017-11-10 04:22:02
updated_at: 2017-11-10 04:22:02
      name: Oven
*************************** 3. row ***************************
        id: b80aec52-d0c9-420a-8471-3254572954b6
created_at: 2017-11-10 04:22:02
updated_at: 2017-11-10 04:22:02
      name: Wifi
guillaume@ubuntu:~/AirBnB_v2$
guillaume@ubuntu:~/AirBnB_v2$ echo 'SELECT * FROM places\G' | mysql -uhbnb_dev -p hbnb_dev_d
b
Enter password:
*************************** 1. row ***************************
              id: 497e3867-d6e9-4401-9c7c-9687c18d2ac7
      created_at: 2017-11-10 04:22:02
      updated_at: 2017-11-10 04:22:02
         city_id: 9d60df6e-31f7-430c-8162-69e89f4a17aa
         user_id: 9b37bd51-6aef-485f-bf10-c7ab83fea2e9
            name: House 1
     description: NULL
    number_rooms: 0
number_bathrooms: 0
       max_guest: 0
  price_by_night: 0
        latitude: NULL
       longitude: NULL
*************************** 2. row ***************************
              id: db549ae1-4500-4d0c-9b50-4b4978ed229e
      created_at: 2017-11-10 04:22:02
      updated_at: 2017-11-10 04:22:02
         city_id: 9d60df6e-31f7-430c-8162-69e89f4a17aa
         user_id: 9b37bd51-6aef-485f-bf10-c7ab83fea2e9
            name: House 2
     description: NULL
    number_rooms: 0
number_bathrooms: 0
       max_guest: 0
  price_by_night: 0
        latitude: NULL
       longitude: NULL
guillaume@ubuntu:~/AirBnB_v2$
guillaume@ubuntu:~/AirBnB_v2$ echo 'SELECT * FROM place_amenity\G' | mysql -uhbnb_dev -p hbn
b_dev_db
Enter password:
```

```
*************************** 1. row ***************************
(/)place_id: 497e3867-d6e9-4401-9c7c-9687c18d2ac7
amenity_id: 47321eb8-152a-46df-969a-440aa67a6d59
*************************** 2. row ***************************
  place_id: db549ae1-4500-4d0c-9b50-4b4978ed229e
amenity_id: 47321eb8-152a-46df-969a-440aa67a6d59
*************************** 3. row ***************************
  place_id: db549ae1-4500-4d0c-9b50-4b4978ed229e
amenity_id: 4a307e7f-68f9-438f-81c0-8325898dda2a
*************************** 4. row ***************************
  place_id: 497e3867-d6e9-4401-9c7c-9687c18d2ac7
amenity_id: b80aec52-d0c9-420a-8471-3254572954b6
*************************** 5. row ***************************
  place_id: db549ae1-4500-4d0c-9b50-4b4978ed229e
amenity_id: b80aec52-d0c9-420a-8471-3254572954b6
guillaume@ubuntu:~/AirBnB_v2$
```

Repo:

- GitHub repository: `AirBnB_clone_v2`
- File: `models/amenity.py, models/place.py`

Done! | Check your code | QA Review