API Hôtellerie

Les Besoins

- Un système de réservation de chambre, commun à tous ces hôtels (ex : un client qui possèdent un site internet pour tous ces hôtels).
- Un CRUD pour les chambres par hôtels.
- Un CRUD sur la gestion des prix des chambres (selon les catégories de chambre (suite, standard, ...), des services additionnels et la politique des prix).
- Une confirmation par mail pour une réservation de chambre.
- Ajouter un nouveau hôtel avec ces mêmes fonctionnalités.

Contexte

Les chambres sont classées par catégories :

- Suite présidentielle (SR), jusqu'à 5 personnes, tarif de base 1000$
- Suite (S), jusqu'à 3 personnes, tarif de base 720$
- Junior Suite (JS), jusqu'à 2 personnes, tarif de base 500$
- Chambre de luxe (CD), jusqu'à deux personnes, tarif de base 300$
- Chambre standard (CS), jusqu'à deux personnes, tarif de base 150$

Des services additionnels peuvent être ajoutés lors d'une réservation :

- Place de garage (25$)
- Ajout d'un lit bébé (sans frais additionnels)
- Pack romance (50$), doit être réservé avec deux jours d'avance
- Petit déjeuner (30$)

Les réservations seront confirmées par mail.

Contexte

Hôtel 1 :

- 5 chambres : 1 Suite, 1 Junior Suite, 1 Chambre de luxe, 2 Chambres standards
- 3 places de garages sont disponibles, une place coûte 25$ par nuit
- 2 lits bébé sont disponibles (sans supplément)

Hôtel 2 :

- 2 suites présidentielles
- 2 places de garage
- 2 lits bébé sont disponibles

Politique de prix :

- Le prix d'une chambre peut être modulé selon la période :
- Pour les nuits de vendredi et samedi le prix des chambres est majoré de 15%
- Les nuits du mercredi et jeudi sont minoré de 10%
- Si une seule personne occupe la chambre le prix est minoré de 5%

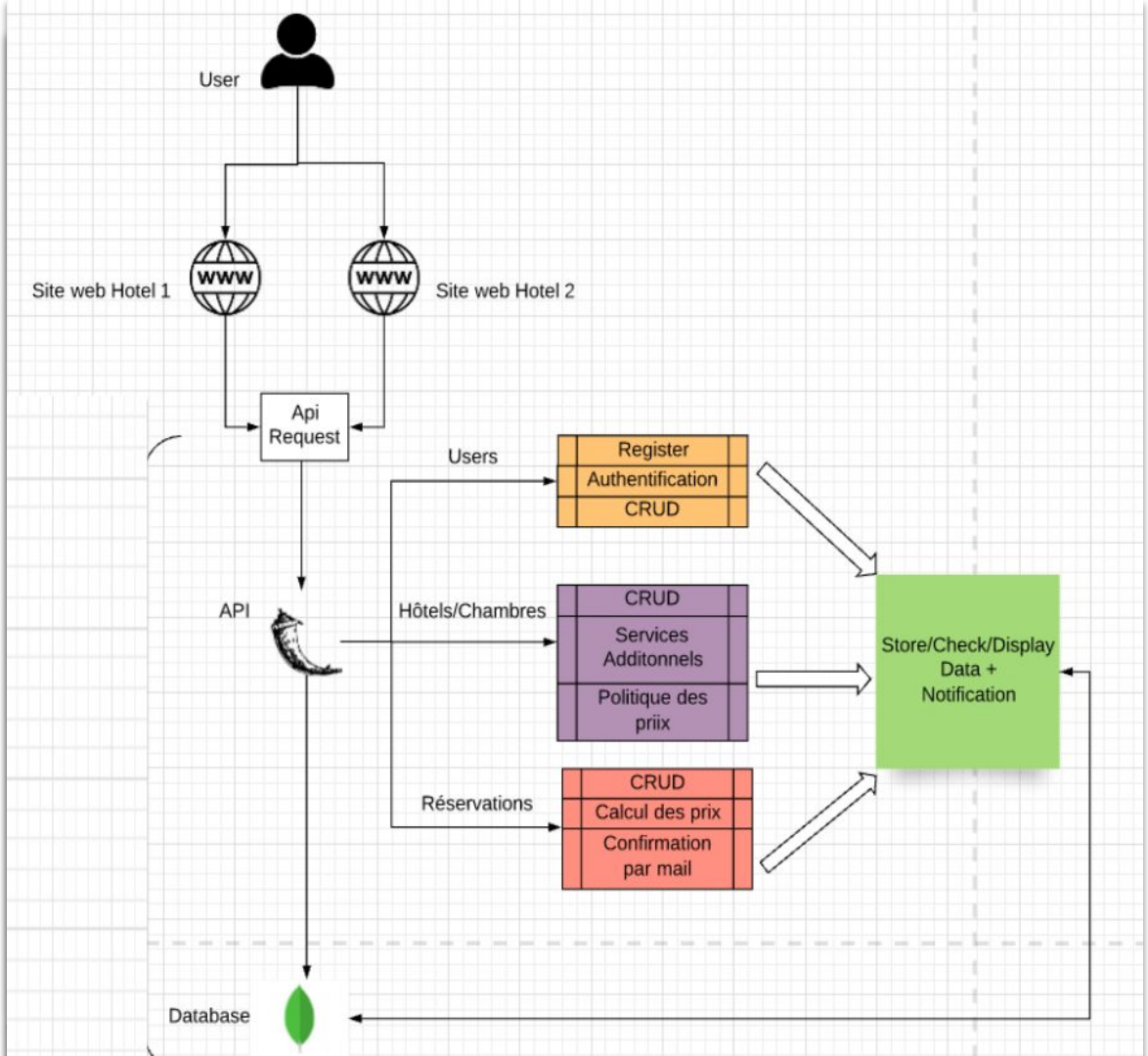
Technologies utilisées (1/2)

- MongoDB :
  - NoSQL (non relationnel)
  - Données semi-structurées (format JSON ou XML)
  - Requêtage rapide et plus complet
  - Flexibilité
  - Scalabilité
- Framework Flask (Python 3) :
  - API et Microservices
  - Simple d'utilisation
  - Implémentation simple des routages d'URL
  - Packages:
    flask_pymongo, flask_mail,
    flask_jwt_extended & flask_bcrypt,
    Blueprint , pytest (Test Unitaire) et
    logging (Log Applicatif)

Technologies utilisées (2/2)

- Swagger :
  - API MOCK
  - Documentation API
- Postman :
  - Test Fonctionnel
  - Newman

Schéma Architecture Applicative



Les Microservices

Microservices	Fonctions
Utilisateurs	CRUD et Authentification Utilisateurs Affichage des utilisateurs
Hôtels/Chambres	CRUD Hôtels CRUD Chambres avec la gestion des prix, les catégories et des services Affichage des Hôtels et Chambres avec les services et les prix
Réservations	CRUD Réservations qui permettra également de calculer les prix de la réservation en fonction des services additionnels Affichage des Réservations

Pytest (Test Unitaire) & Newman(Test Fonctionnel)

```
(venv) thiva@debian:~/Documents/API/modules$ python3 -m pytest -v tests/
============================= test session starts ==============================
platform linux -- Python 3.5.3, pytest-6.0.0, py-1.9.0, pluggy-0.13.1 -- /home/thiva/Documents/API/venv/bin/python3
cachedir: .pytest_cache
rootdir: /home/thiva/Documents/API/modules
plugins: mongodb-2.2.0, Faker-4.1.1
collected 32 items

tests/test_api.py::test_register_users PASSED                            [  3%]
tests/test_api.py::test_register_bad_request_users PASSED                [  6%]
tests/test_api.py::test_register_email_exist_users PASSED                [  9%]
tests/test_api.py::test_auth_user PASSED                                 [ 12%]
tests/test_api.py::test_auth_bad_request_user PASSED                     [ 15%]
tests/test_api.py::test_auth_user_invalid_user PASSED                    [ 18%]
tests/test_api.py::test_get_unauthorized_user PASSED                     [ 21%]
tests/test_api.py::test_get_user PASSED                                  [ 25%]
tests/test_api.py::test_put_bad_request_user PASSED                      [ 28%]
tests/test_api.py::test_post_hotel PASSED                                [ 31%]
tests/test_api.py::test_post_exist_hotel PASSED                          [ 34%]
tests/test_api.py::test_post_element_hotel PASSED                        [ 37%]
tests/test_api.py::test_put_element_hotel PASSED                         [ 40%]
tests/test_api.py::test_put_hotel PASSED                                 [ 43%]
tests/test_api.py::test_delete_element_hotel PASSED                      [ 46%]
tests/test_api.py::test_get_unauthorized_hotel PASSED                    [ 50%]
tests/test_api.py::test_get_hotel PASSED                                 [ 53%]
tests/test_api.py::test_delete_unauthorized_hotel PASSED                 [ 56%]
tests/test_api.py::test_logout_user PASSED                               [ 59%]
tests/test_api.py::test_register_client_users PASSED                     [ 62%]
tests/test_api.py::test_auth_client_user PASSED                          [ 65%]
tests/test_api.py::test_param_not_exit_reservation PASSED                [ 68%]
tests/test_api.py::test_get_reservation PASSED                           [ 71%]
tests/test_api.py::test_post_reservation PASSED                          [ 75%]
tests/test_api.py::test_post_not_reservation PASSED                      [ 78%]
tests/test_api.py::test_get_user_reservation PASSED                      [ 81%]
tests/test_api.py::test_delete_not_exit_reservation PASSED               [ 84%]
tests/test_api.py::test_delete_reservation PASSED                        [ 87%]
tests/test_api.py::test_delete_user PASSED                               [ 90%]
tests/test_api.py::test_auth1_user PASSED                                [ 93%]
tests/test_api.py::test_delete_hotel PASSED                              [ 96%]
tests/test_api.py::test_put_user PASSED                                  [100%]

============================== 32 passed in 4.10s ==============================
```

```
→ test_delete_not_exit_reservation
  DELETE http://localhost:8881/reservation/?Nom=Hotel 1 [409 CONFLICT, 212B, 9ms]
  ✓  Status test

→ test_delete_reservation
  DELETE http://localhost:8881/reservation/?Nom=Hotel 1 [200 OK, 201B, 18ms]
  ✓  Status test

→ test_delete_user
  DELETE http://localhost:8881/user/ [200 OK, 209B, 9ms]
  ✓  Status test

→ test_auth1_user
  POST http://localhost:8881/user/auth [200 OK, 588B, 242ms]
  ✓  Status test

→ test_delete_hotel
  DELETE http://localhost:8881/hotel/ [200 OK, 194B, 9ms]
  ✓  Status test

→ test_put_user
  PUT http://localhost:8881/user/ [200 OK, 210B, 242ms]
  ✓  Status test

→ test_auth2_user
  POST http://localhost:8881/user/auth [200 OK, 592B, 245ms]
  ✓  Status test

→ test_delete2_user
  DELETE http://localhost:8881/user/ [200 OK, 209B, 10ms]
  ✓  Status test
```

	executed	failed
iterations	1	0
requests	34	0
test-scripts	34	0
prerequest-scripts	1	0
assertions	34	0
total run duration: 4.1s		
total data received: 32.33KB (approx)		
average response time: 105ms [min: 5ms, max: 1182ms, s.d.: 219ms]		

```
thiva@debian:~/Documents/API/test_fontionnel$
```