

CRAM - My Booking Service

README - Spécification Générale

Ce document est un récapitulatif des travaux effectués sur le projet. Il sert aussi bien de SFD (Spécification Fonctionnelle Détaillé) que de SFT (Spécification Technique Détaillé) et à certain égard de DAT (Document d'Architecture Technique).

I - Introduction

A. Organisation

Pour répondre aux besoins du client, nous avons choisi de suivre une méthodologie Agile avec des sprints de 3 semaines. Les cérémonies retenues sont les daily un jour sur deux, un weekly chaque jeudi et un monthly tous les 3 vendredis. Les journées seront ponctuées de séances de peer-programming entre les profils concernés par la tâche en cours (entre DevOps, entre Devs ou entre DevOps et Devs).

Lors des monthly, nous ferons le point sur l'évolution du projet, le respect des deadlines et des objectifs, la bonne répartition des tâches et les nouveaux objectifs si besoin.

B. Profils

Pour la répartition des tâches nous avons fait en fonction des expertises de chacun en effet :

Alan est DevOps, il pourra apporter son expérience et ses connaissances sur la CI/CD et apporter du soutien sur le maintien de l'infrastructure.

Maxime a un profil multi-casquette (Dev/DevOps/Ops). Il se concentrera sur la configuration du cluster Kubernetes et d'Istio, ainsi que du déploiement des API sur le cluster. L'accent sera mis sur les stratégies de déploiement. Il accompagnera aussi les développeurs sur les bonnes pratiques de développement pour ce genre d'environnement.

Randy est développeur Full-Stack spécialisé dans les projets Symfony, il pourra apporter un jugement sur le développement des API et des tests.

Corentin aussi est développeur Full-Stack, il a l'habitude de travailler aussi bien sur le front-end que sur le back-end. Ainsi, il a su développer une réflexion pertinente sur le jugement des fonctionnalités à développer. Il pourra facilement détecter et remonter les anomalies et bugs d'où son rôle de lead test.

La répartition des tâches

Qui ?	Quoi ?	Comment ?
Maxime Places	Mise en place de l'infrastructure, environnement de dev, déploiement	Rôles Ansible, docker-compose, helm charts
Randy DERET	API Booking et Hotel + Base de donnée + accompagnement des tests	Mise en place de l'ORM et création des routes API. Mise en forme de l'architecture dans les API. Communication BDD
Corentin HOUDECEK	Tests Booking et Hotel + Accompagnement dev API	Mise en place des tests unitaires des API. Contrôle régulier des swagger. Implémentation des tests dans le docker-compose.
Alan Mecheraf	CI/CD + Doc	Intégration des pipelines et des jobs lors de l'intégration continu sur Gitlab

C. Technologies

Nous avons opté pour une solution d'orchestration de containers éprouvée par la communauté DevOps : Kubernetes.

Nous utiliserons Containerd en tant que Container Runtime et Calico en Container Networking Interface. Containerd est une solution plus économe en ressources et plus sécurisée que l'environnement d'exécution Docker. Calico est aussi une solution appropriée en terme de consommation de ressources et de performance (cf. benchmark ci-dessous).

Nous configurons les VMs mises à disposition grâce à un playbook Ansible, allant de la création du cluster jusqu'au déploiement de la stack Tick sur celui-ci.

Concernant la stack applicative, nous avons opté pour l'utilisation du langage Python pour sa syntaxe claire et lisible ainsi que pour sa grande communauté et documentation. Nous avons retenu le framework FastAPI pour sa simplicité d'intégration. Le stockage de la donnée est fait à l'aide de bases de données Postgresql.

Notre découpage en micro-services permet de gérer sur une API les réservations et utilisateurs, et sur l'autre les hôtels, chambres et prix de ces dernières.

Il sera possible de créer une troisième API contenant la logique métier sur le pattern du Backend For Frontend.

Nous avons installé sur notre cluster Kubernetes un IngressController NGINX. Celui-ci sera en mesure de créer des points d'entrée vers nos API en s'assurant de la bonne résolution des noms et des certificats TLS. L'ingress déployé avec l'API permettra de rediriger vers le service de l'appliquatif en question, et ce service enverra la requête aux pods.

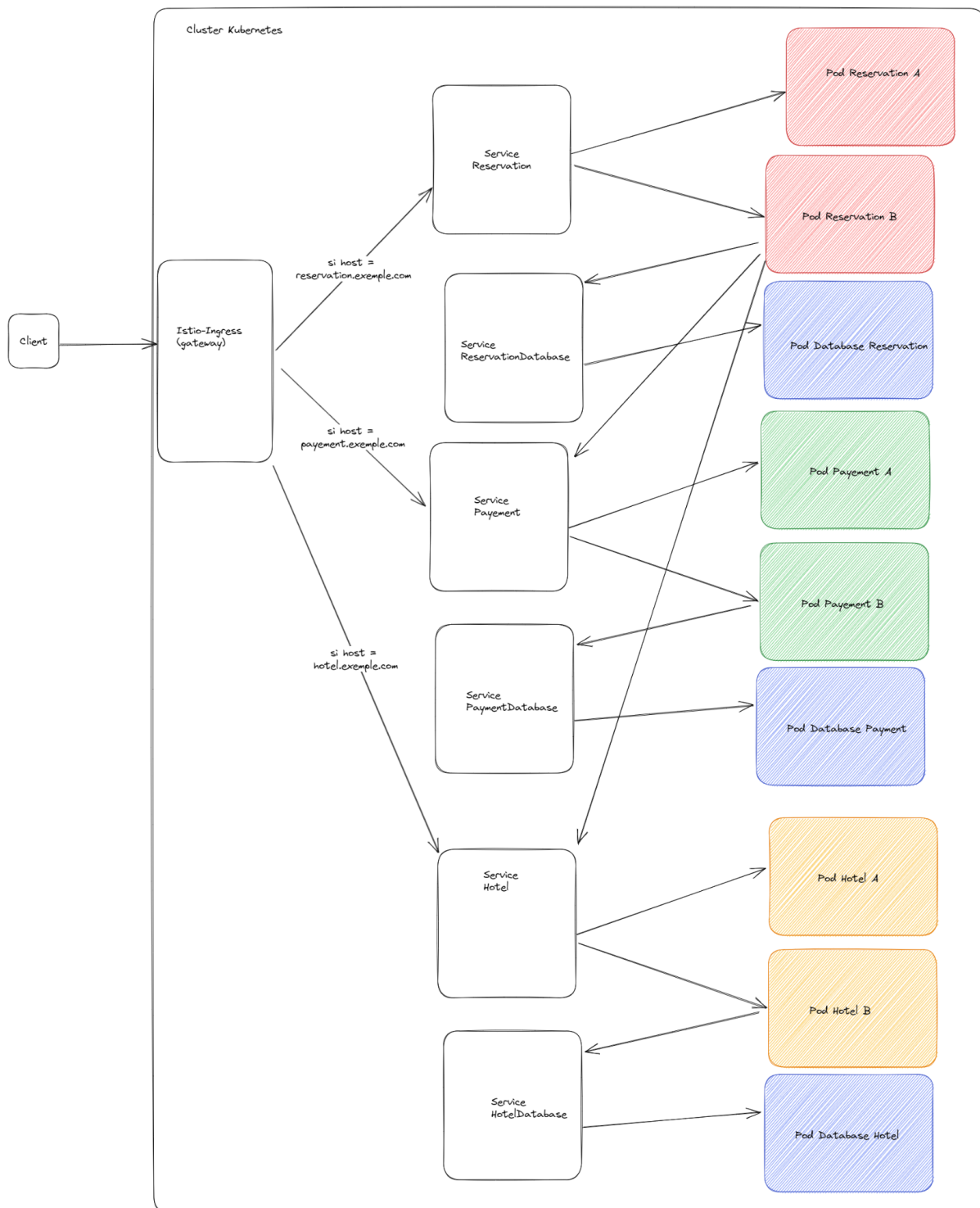
La communication entre notre API Backend for Frontend et les 2 premières API se fera en interne dans le cluster Kubernetes. Cependant il sera nécessaire d'exposer nos 3 API : que ce soit pour un back-office ou pour une application web destinée au client final.

II - Découpage en micro-services

Le découpage du besoin en microservices que nous avons réalisé est comme suit :

- Service de réservation et de paiement : ce microservice serait responsable de gérer les réservations et les annulations de réservations. Il s'occuperait de la vérification de la disponibilité des chambres, du calcul du prix de la réservation, de l'envoi de la confirmation de réservation par e-mail, et de la gestion des annulations de réservations.
- Service de gestion des hôtels : ce microservice serait responsable de gérer les informations sur les différents hôtels, tels que les informations sur les chambres disponibles, les tarifs, les services proposés, etc. Il fournira ces informations au service de réservation pour aider à la prise de décision sur la disponibilité des chambres et le calcul des prix.
- Service de paiement : ce microservice serait responsable de la gestion des paiements pour les réservations de chambres d'hôtel. Il s'occuperait de l'autorisation des paiements par carte de crédit, de la collecte des paiements et de la notification du service de réservation une fois le paiement accepté.

Ce schéma ci-dessous représente le fonctionnement simplifié d'un appel HTTP vers un applicatif hébergé sur un cluster Kubernetes utilisant Istio pour faire du service-meshing. Les pods applicatifs seront stateless et les bases de données seront stateful avec des volumes de stockage réservés.



Nous avons développé nos microservices en Fastapi afin d'avoir cette cohérence dans le choix de nos technologies

Hotel API :

La “Hotel API” est l’api servant à gérer les hôtels, leurs chambres et les catégories de ces dernières.

default	^
GET / Health Check	▼
GET /itemtest/{item_id} Read Main	▼
POST /itemtest/ Create Item	▼
GET /hotels/ Read Hotels	▼
POST /hotels/ Create Hotel	▼
GET /hotels/{hotel_id} Read Hotel	▼
DELETE /hotels/{hotel_id} Delete Hotel	▼
PUT /hotel/{hotel_id} Update Hotel	▼
GET /categories/ Read Categories	▼
POST /categories/ Create Category	▼
GET /categories/{category_id} Read Category	▼
PUT /categories/{category_id} Update Category	▼
DELETE /categories/{category_id} Delete Category	▼
POST /hotels/{hotel_id}/rooms/ Create Room For Hotel	▼
GET /rooms/ Read Rooms	▼
DELETE /rooms/{room_id} Delete Room	▼
PUT /hotels/{hotel_id}/rooms/{room_id} Update Room For Hotel	▼

Health check :

GET / Health Check	^
Parameters	
No parameters	
Try it out	
Responses	
Code	Description
200	Successful Response
Links	
No links	
Media type	
application/json	
Controls Accept header	
Example Value Schema	
"string"	

Read hotels :

Responses

Code	Description	Links
200	<p>Successful Response</p> <p>Media type application/json</p> <p>Controls Accept header</p> <p>Example Value Schema</p> <pre>[{ "name": "string", "id": 0, "address": "string", "is_active": true, "rooms": [{ "title": "string", "description": "string", "category_id": 0, "id": 0, "owner_id": 0, "category": { "name": "string", "description": "string", "id": 0 } }] }]</pre>	No links
422	<p>Validation Error</p> <p>Media type application/json</p> <p>Example Value Schema</p> <pre>{ "detail": [{ "loc": ["string", 0], "msg": "string", "type": "string" }] }</pre>	No links

Create hotel :

Request body required application/json

Example Value | Schema

```
{
  "name": "string",
  "address": "string"
}
```

Responses		
Code	Description	Links
200	Successful Response	No links
<div>Media type application/json</div> <div>Controls Accept header.</div> <div>Example Value Schema</div> <pre>{ "name": "string", "id": 0, "address": "string", "is_active": true, "rooms": [{ "title": "string", "description": "string", "category_id": 0, "id": 0, "owner_id": 0, "category": { "name": "string", "description": "string", "id": 0 } }]}</pre>		
422	Validation Error	No links
<div>Media type application/json</div> <div>Example Value Schema</div> <pre>{ "detail": [{ "loc": ["string", 0], "msg": "string", "type": "string" }]}</pre>		

Read hotel :

GET /hotels/{hotel_id} Read Hotel		
Parameters		
Try it out		
Name	Description	
hotel_id * required integer (path)	hotel_id	
Responses		
Code	Description	Links
200	Successful Response	No links
<div>Media type application/json</div> <div>Controls Accept header.</div> <div>Example Value Schema</div> <pre>{ "name": "string", "id": 0, "address": "string", "is_active": true, "rooms": [{ "title": "string", "description": "string", "category_id": 0, "id": 0, "owner_id": 0, "category": { "name": "string", "description": "string", "id": 0 } }]}</pre>		

Delete hotel :

DELETE

/hotels/{hotel_id} Delete Hotel

Parameters

Try it out

Name	Description
hotel_id <div>required</div>	
integer	hotel_id
(path)	

Responses

Code	Description	Links
200	Successful Response	No links

Media type

application/json

Controls Accept header.

Example Value

Schema

"string"

Update hotel :

PUT

/hotel/{hotel_id} Update Hotel

Parameters

Try it out

Name	Description
hotel_id <div>required</div>	
integer	hotel_id
(path)	

Request body

required

application/json

Example Value

Schema

{
 "name": "string",
 "address": "string"
}

Responses

Code	Description	Links
200	Successful Response	No links

Media type

application/json

Controls Accept header.

Example Value

Schema

{
 "name": "string",
 "id": 0,
 "address": "string",
 "is_active": true,
 "rooms": [
 {
 "title": "string",
 "description": "string",
 "category_id": 0,
 "id": 0,
 "owner_id": 0,
 "category": {
 "name": "string",
 "description": "string",
 "id": 0
 }
 }
]
}

Read categories :

GET

/categories/

Read Categories

Parameters

Try it out

Name	Description
skip integer (query)	Default value : 0
	<input type="text" value="0"/>
limit integer (query)	Default value : 100
	<input type="text" value="100"/>

Responses

Code	Description	Links
200	Successful Response	No links

Media type

application/json

Controls Accept header.

Example Value | Schema

```
[
  {
    "name": "string",
    "description": "string",
    "id": 0
  }
]
```

Create category :

POST

/categories/

Create Category

Parameters

Try it out

No parameters

Request body required

application/json

Example Value | Schema

```
{
  "name": "string",
  "description": "string"
}
```

Responses

Code	Description	Links
200	Successful Response	No links

Media type

application/json

Controls Accept header.

Example Value | Schema

```
{
  "name": "string",
  "description": "string",
  "id": 0
}
```

Read category :

GET

/categories/{category_id} Read Category

⌵

Parameters

Try it out

Name	Description
category_id * required integer (path)	<input type="text" value="category_id"/>

Responses

Code	Description	Links
200	Successful Response	No links

Media type

application/json

⌵

Controls Accept header.

Example Value

Schema

```
{
  "name": "string",
  "description": "string",
  "id": 0
}
```

Update category :

PUT

/categories/{category_id} Update Category

⌵

Parameters

Try it out

Name	Description
category_id * required integer (path)	<input type="text" value="category_id"/>

Request body

required

application/json

⌵

Example Value

Schema

```
{
  "name": "string",
  "description": "string"
}
```

Responses

Code	Description	Links
200	Successful Response	No links

Media type

application/json

⌵

Controls Accept header.

Example Value

Schema

```
{
  "name": "string",
  "description": "string",
  "id": 0
}
```

Delete category :

DELETE

/categories/{category_id} Delete Category

Parameters

Try it out

Name	Description
category_id * required integer (path)	<input type="text" value="category_id"/>

Responses

Code	Description	Links
200	Successful Response	No links

Media type

application/json

Controls Accept header.

Example Value

Schema

"string"

Create room for hotel :

POST

/hotels/{hotel_id}/rooms/ Create Room For Hotel

Parameters

Try it out

Name	Description
hotel_id * required integer (path)	<input type="text" value="hotel_id"/>

Request body

required

application/json

Example Value

Schema

{
 "title": "string",
 "description": "string",
 "category_id": 0
}

Responses

Code	Description	Links
200	Successful Response	No links

Media type

application/json

Controls Accept header.

Example Value

Schema

{
 "title": "string",
 "description": "string",
 "category_id": 0,
 "id": 0,
 "owner_id": 0,
 "category": {
 "name": "string",
 "description": "string",
 "id": 0
 }
}

Read rooms :

GET /rooms/ Read Rooms

Try it out

Parameters

Name	Description
skip integer (query)	Default value : 0
limit integer (query)	Default value : 100

Responses

Code	Description	Links
200	Successful Response	No links

Media type

application/json

Controls Accept header.

Example Value | Schema

```
[
  {
    "title": "string",
    "description": "string",
    "category_id": 0,
    "id": 0,
    "owner_id": 0,
    "category": {
      "name": "string",
      "description": "string",
      "id": 0
    }
  }
]
```

Delete room :

DELETE /rooms/{room_id} Delete Room

Try it out

Parameters

Name	Description
room_id * required integer (path)	room_id

Responses

Code	Description	Links
200	Successful Response	No links

Media type

application/json

Controls Accept header.

Example Value | Schema

```
"string"
```

Update room for hotel :

PUT

/hotels/{hotel_id}/rooms/{room_id} Update Room For Hotel

Try it out

Parameters

Name	Description
hotel_id <small>* required</small>	<input type="text" value="hotel_id"/>
<small>integer (path)</small>	
room_id <small>* required</small>	<input type="text" value="room_id"/>
<small>integer (path)</small>	

Request body required

application/json

Example Value | Schema

```
{
  "title": "string",
  "description": "string",
  "category_id": 0
}
```

Responses

Code	Description	Links
200	Successful Response	No links

Media type

application/json

Controls Accept header.

Example Value | Schema

```
{
  "title": "string",
  "description": "string",
  "category_id": 0,
  "id": 0,
  "owner_id": 0,
  "category": {
    "name": "string",
    "description": "string",
    "id": 0
  }
}
```

Booking API :

La “Booking API” est l’api servant de gestion d’utilisateur mais elle sert aussi à gérer toute la partie réservation, comprenant donc le paiement et les possibles services additionnels pouvant être demandés.

default

GET	/ Health Check	▼
GET	/users/ Read Users	▼
POST	/users/ Create User	▼
GET	/users/{user_id} Read User	▼
PUT	/users/{user_id} Update User	▼
DELETE	/users/{user_id} Delete User	▼
POST	/users/{user_id}/bookings/ Create Booking For User	▼
GET	/bookings/ Get Bookings	▼
GET	/users/{user_id}/bookings/{booking_id} Read Booking User	▼
PUT	/users/{user_id}/bookings/{booking_id} Update Booking For User	▼
DELETE	/users/{user_id}/bookings/{booking_id} Delete Booking For User	▼
GET	/additional_services/ Read Additional Services	▼
POST	/additional_services/ Create Additional Service	▼
GET	/additional_services/{additional_service_id} Read Additional Service	▼
PUT	/additional_services/{additional_service_id} Update Additional Service	▼
DELETE	/additional_services/{additional_service_id} Delete Additional Service	▼
GET	/bookings/{booking_id}/payment/ Read Payment	▼

Health check :

GET / Health Check

Parameters

Cancel

No parameters

ExecuteClear

Responses

Curl

curl -X 'GET' \
'http://localhost:8004/' \
-H 'accept: application/json'

Request URL

http://localhost:8004/

Server response

Code	Details
200	<div>Response body</div> <div>"Healthy"</div> <div>Download</div> <div>Response headers</div> <div>content-length: 9 content-type: application/json date: Thu, 01 Jun 2023 09:08:05 GMT server: uvicorn</div>

Read users :

GET

/users/ Read Users

Try it out

Parameters

No parameters

Responses

Code	Description	Links
200	Successful Response	No links

Media type

application/json

Controls Accept header.

Example Value | Schema

```
{
  "firstname": "string",
  "lastname": "string",
  "age": 0,
  "phone": 0,
  "email": "string",
  "id": 0,
  "hashed_password": "string",
  "bookings": [
    {
      "nights": 0,
      "numbers_people": 0,
      "users_name": "string",
      "id": 0,
      "user_id": 0,
      "reservation_number": 0,
      "additional_service": [
        {
          "name": "string",
          "price": 0,
          "id": 0
        }
      ],
      "payment": {
        "price": 0,
        "id": 0
      }
    }
  ]
}
```

Create user :

POST

/users/ Create User

Try it out

Parameters

No parameters

Request body required

application/json

Example Value | Schema

```
{
  "firstname": "string",
  "lastname": "string",
  "age": 0,
  "phone": 0,
  "email": "string",
  "hashed_password": "string"
}
```

Responses

Code	Description	Links
200	Successful Response	No links

Media type

application/json

Controls Accept header.

Example Value | Schema

```
{  "firstname": "string",  "lastname": "string",  "age": 0,  "phone": 0,  "email": "string",  "id": 0,  "hashed_password": "string",  "bookings": [    {      "nights": 0,      "numbers_people": 0,      "users_name": "string",      "id": 0,      "user_id": 0,      "reservation_number": 0,      "additional_service": [        {          "name": "string",          "price": 0,          "id": 0        }      ]    }  ],  "payment": {    "price": 0,
```

Read user :

GET /users/{user_id} Read User

Try it out

Parameters

Name	Description
user_id * required integer (path)	user_id

Responses

Code	Description	Links
200	Successful Response	No links

Media type

application/json

Controls Accept header.

Example Value | Schema

```
{  "firstname": "string",  "lastname": "string",  "age": 0,  "phone": 0,  "email": "string",  "id": 0,  "hashed_password": "string",  "bookings": [    {      "nights": 0,      "numbers_people": 0,      "users_name": "string",      "id": 0,      "user_id": 0,      "reservation_number": 0,      "additional_service": [        {          "name": "string",          "price": 0,          "id": 0        }      ]    }  ],  "payment": {    "price": 0,
```


Update user :

PUT

/users/{user_id} Update User

⌵

Parameters

Try it out

Name	Description
user_id * required integer (path)	<input type="text" value="user_id"/>

Request body

required

application/json

⌵

Example Value

Schema

```
{
  "firstname": "string",
  "lastname": "string",
  "age": 0,
  "phone": 0,
  "email": "string",
  "hashed_password": "string"
}
```

Responses

Code	Description	Links
200	Successful Response	No links

Media type

application/json

⌵

Controls

Accept header.

Example Value

Schema

```
{
  "firstname": "string",
  "lastname": "string",
  "age": 0,
  "phone": 0,
  "email": "string",
  "id": 0,
  "hashed_password": "string",
  "bookings": [
    {
      "nights": 0,
      "numbers_people": 0,
      "users_name": "string",
      "id": 0,
      "user_id": 0,
      "reservation_number": 0,
      "additional_service": [
        {
          "name": "string",
          "price": 0,
          "id": 0
        }
      ],
      "payment": {
        "price": 0,

```

Delete user :

DELETE

/users/{user_id} Delete User

⌵

Parameters

Try it out

Name	Description
user_id * required integer (path)	<input type="text" value="user_id"/>

Responses

Code	Description	Links
200	Successful Response	No links

Media type

application/json

⌵

Controls

Accept header.

Example Value

Schema

```
"string"
```

Create booking for user :

POST

/users/{user_id}/bookings/

Create Booking For User

Parameters

Try it out

Name	Description
user_id * required integer (path)	<input type="text" value="user_id"/>

Request body

required

application/json

Example Value

Schema

```
{
  "nights": 0,
  "numbers_people": 0,
  "users_name": "string",
  "additional_service_ids": [
    0
  ]
}
```

Responses

Code	Description	Links
200	Successful Response	No links

Media type

application/json

Controls Accept header.

Example Value

Schema

```
{
  "nights": 0,
  "numbers_people": 0,
  "users_name": "string",
  "id": 0,
  "user_id": 0,
  "reservation_number": 0,
  "additional_service": [
    {
      "name": "string",
      "price": 0,
      "id": 0
    }
  ],
  "payment": {
    "price": 0,
    "id": 0
  }
}
```

Read bookings :

GET

/bookings/

Get Bookings

Parameters

Try it out

No parameters

Responses

Code	Description	Links
200	Successful Response	No links

Media type

application/json

Controls Accept header.

Example Value

Schema

```
[
  {
    "nights": 0,
    "numbers_people": 0,
    "users_name": "string",
    "id": 0,
    "user_id": 0,
    "reservation_number": 0,
    "additional_service": [
      {
        "name": "string",
        "price": 0,
        "id": 0
      }
    ],
    "payment": {
      "price": 0,
      "id": 0
    }
  }
]
```

Read booking user :

GET

/users/{user_id}/bookings/{booking_id}

Read Booking User

Parameters

Try it out

Name	Description
user_id * required integer (path)	user_id
booking_id * required integer (path)	booking_id

Responses

Code	Description	Links
200	Successful Response	No links

Media type

application/json

Controls Accept header.

Example Value

Schema

```
{
  "nights": 0,
  "numbers_people": 0,
  "users_name": "string",
  "id": 0,
  "user_id": 0,
  "reservation_number": 0,
  "additional_service": [
    {
      "name": "string",
      "price": 0,
      "id": 0
    }
  ],
  "payment": {
    "price": 0,
    "id": 0
  }
}
```

Update booking for user :

PUT

/users/{user_id}/bookings/{booking_id} Update Booking For User

Parameters

Try it out

Name	Description
user_id * required integer (path)	<input type="text" value="user_id"/>
booking_id * required integer (path)	<input type="text" value="booking_id"/>

Request body

required

application/json

Example Value

Schema

```
{
  "nights": 0,
  "numbers_people": 0,
  "users_name": "string"
}
```

Responses

Code	Description	Links
200	Successful Response	No links

Media type

application/json

Controls Accept header.

Example Value

Schema

```
{
  "nights": 0,
  "numbers_people": 0,
  "users_name": "string",
  "id": 0,
  "user_id": 0,
  "reservation_number": 0,
  "additional_service": [
    {
      "name": "string",
      "price": 0,
      "id": 0
    }
  ],
  "payment": {
    "price": 0,
    "id": 0
  }
}
```

Delete booking for user :

DELETE

/users/{user_id}/bookings/{booking_id} Delete Booking For User

Parameters

Try it out

Name	Description
user_id * required integer (path)	<input type="text" value="user_id"/>
booking_id * required integer (path)	<input type="text" value="booking_id"/>

Responses

Code	Description	Links
200	Successful Response	No links

Media type

application/json

Controls Accept header.

Example Value

Schema

```
{
  "nights": 0,
  "numbers_people": 0,
  "users_name": "string",
  "id": 0,
  "user_id": 0,
  "reservation_number": 0,
  "additional_service": [
    {
      "name": "string",
      "price": 0,
      "id": 0
    }
  ],
  "payment": {
    "price": 0,
    "id": 0
  }
}
```

Read additional services :

GET

/additional_services/ Read Additional Services

Parameters

Try it out

No parameters

Responses

Code	Description	Links
200	Successful Response	No links

Media type

application/json

Controls Accept header.

Example Value

Schema

```
[
  {
    "name": "string",
    "price": 0,
    "id": 0
  }
]
```

Create additional service :

POST

/additional_services/

Create Additional Service

Parameters

Try it out

No parameters

Request body

required

application/json

Example Value

Schema

```
{
  "name": "string",
  "price": 0
}
```

Responses

Code	Description	Links
200	Successful Response	No links

Media type

application/json

Controls Accept header.

Example Value

Schema

```
{
  "name": "string",
  "price": 0,
  "id": 0
}
```

Read additional service :

GET

/additional_services/{additional_service_id}

Read Additional Service

Parameters

Try it out

Name	Description
additional_service_id <div>integer (path)</div>	additional_service_id

Responses

Code	Description	Links
200	Successful Response	No links

Media type

application/json

Controls Accept header.

Example Value

Schema

```
{
  "name": "string",
  "price": 0,
  "id": 0
}
```

Update additional services :

PUT

/additional_services/{additional_service_id}

Update Additional Service

Parameters

Try it out

Name

Description

additional_service_id * required

integer

(path)

additional_service_id

Request body

required

application/json

Example Value | Schema

{

"name": "string",

"price": 0

}

Responses

Code

Description

Links

200

Successful Response

No links

Media type

application/json

Controls Accept header.

Example Value | Schema

{

"name": "string",

"price": 0,

"id": 0

}

Delete additional services :

DELETE

/additional_services/{additional_service_id}

Delete Additional Service

Parameters

Try it out

Name

Description

additional_service_id * required

integer

(path)

additional_service_id

Responses

Code

Description

Links

200

Successful Response

No links

Media type

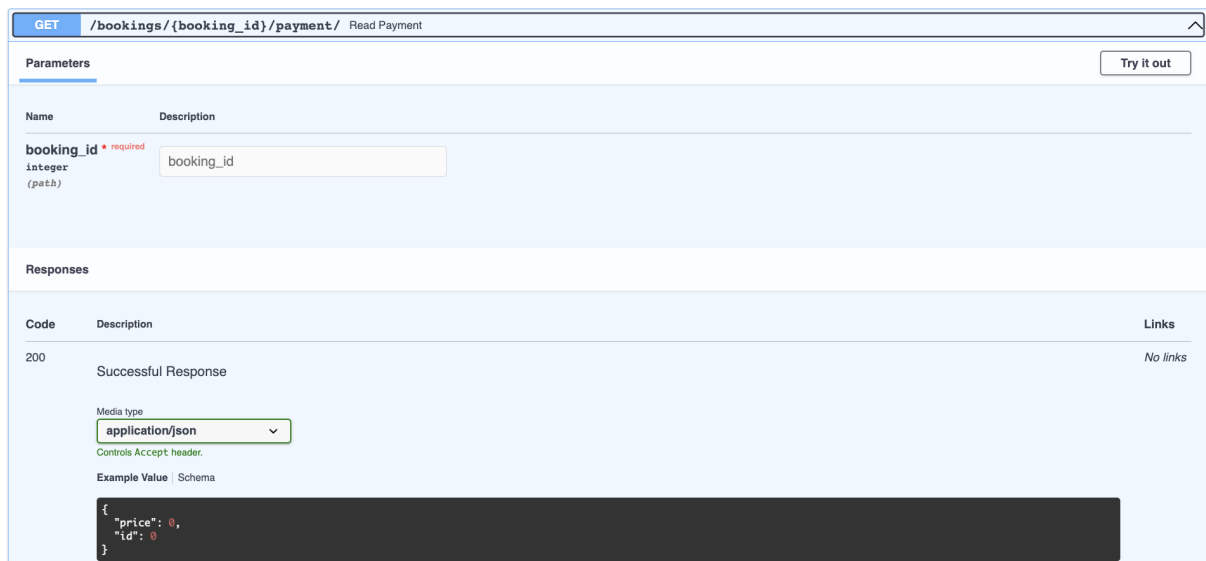
application/json

Controls Accept header.

Example Value | Schema

"string"

Read Payment :



The image shows a Swagger UI interface for a REST API endpoint. The top bar indicates a GET request to `/bookings/{booking_id}/payment/` with the title "Read Payment". Below this, the "Parameters" section lists a required path parameter `booking_id` of type integer. The "Responses" section shows a 200 status code for a "Successful Response" with a media type of `application/json`. An example JSON response is displayed in a dark box: `{ "price": 0, "id": 0 }`.

Name	Description
booking_id * required integer (path)	booking_id

Code	Description	Links
200	Successful Response	No links

Media type: `application/json`

Example Value: `{ "price": 0, "id": 0 }`

III - Fonctionnement et intégration du Git Flow

A. Environnements

Nous disposons actuellement d'un cluster Kubernetes. Idéalement, nous aurions un cluster par environnement. Afin de respecter les ressources attribuées, nous simulons un environnement de Production et de Pré-production en les isolant par namespace et par sous-domaine.

Par exemple :

- Production : `hotel-api.example.local`
- Préproduction : `hotel-api-preprod.example.local`

B. Donnée Mocker :

Pour mocker nos données, nous avons eu l'idée de seeder l'information lors des lancements de nos services .

Hotels:

Id	Name	Address	Is Active
----	------	---------	-----------

51	Bobby's Hotel	7 rue du voisin	true
52	Randy's Hotel	14 rue du four	true

Categories:

Id	Name	Description
51	Suite présidentielle	jusqu'à 5 personnes, tarif de 1000\$
52	Suite	jusqu'à 3 personnes, tarif de 720\$
53	Junior suite	jusqu'à 2 personnes, tarif de 500\$
54	Chambre de luxe	jusqu'à 2 personnes, tarif de 300\$
55	Chambre standard	jusqu'à 2 personnes, tarif de 150\$

Rooms:

Id	Title	Description	Hotel Id	Category Id
51	101	la 101	51	52
52	102	la 102	51	53
53	103	la 103	51	54
54	104	la 104	51	55
55	105	la 105	51	55
56	101	la 101	52	51
57	102	la 102	52	51

Additional Services:

Id	Name	Price
51	Place de la gare	25
52	Lit bebe	0
53	Pack romance	50
54	Petit dejeuner	30

C. Politique de branche

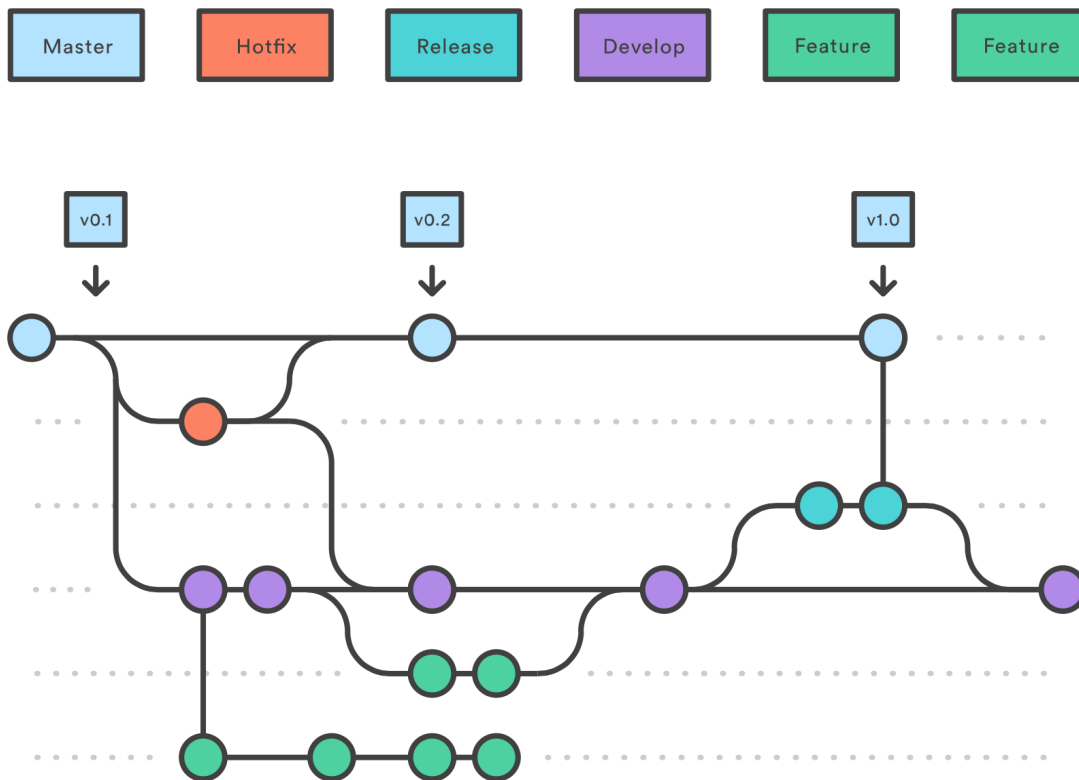
La mise en place d'une politique de branche pour empêcher de pousser des modifications sur la branche main est primordiale afin de s'assurer de la cohérence des modifications et de la qualité du code.

Pour pousser des modifications sur la branche de développement, il faudra passer par une Merge Request, qui déclenche le lancement des tests unitaires sur la branche à merger pour mettre en avant la couverture des tests et empêcher les régressions.

Les Merge Request auront par ailleurs l'obligation d'être approuvées par un pair afin d'éviter au maximum les oublis et mauvaises pratiques : Les développeurs et devOps auront des branches de feature respectives à leurs organisations. Concernant l'intégration des modifications dans la branche main via la branche develop, celle-ci nécessitera un déploiement en préprod avec tests et l'autorisation d'un devOps et d'un développeur.

Nous définirons des Tags sur les commits au fur et à mesure de l'ajout des features, cela nous permettra de pouvoir revenir sur une version plus facilement si nécessaire.

Au fil du développement du projet, nous demanderons aux devops et développeurs d'informer par l'intermédiaire des issues Git les informations importantes à signaler sur le nouveau commit : Détailler légèrement le code apporté, éventuellement alerter si un problème est présent, s'il nécessite d'être revu ou testé par quelqu'un d'autres plus en détail, ou tout simplement signaler que tout est OK. Les issues pourront aussi permettre de créer une nouvelle tâche au projet avec une description du besoin, si un développeur exige un environnement particulier de la part des DevOps par exemple.



D. Tests de CI

Pour la lisibilité et la cohérence de notre code entre les développeurs, nous mettrons une norme à notre code qui sera analysé via Pylint.

Pour finir nous devons mettre en œuvre, bien entendu, le SAST Gitlab afin de vérifier les possibles failles et vulnérabilités dans notre code ou les librairies utilisées. Il faut néanmoins savoir que le SAST est un outil permettant de trouver une faille de manière théorique.

E. Mise en production

Concernant la CI/CD, nous utiliserons des agents Gitlab hébergés sur notre infrastructure. La pipeline consistera à construire les images Docker, les publier sur un registre de conteneurs puis de les déployer avec des charts Helm sur notre cluster Kubernetes. Une fois les applications déployées, des tests d'intégrations seront lancés à l'aide d'un Runner Postman pour s'assurer de la disponibilité et du bon fonctionnement individuel de nos API. Enfin, une fois que les API seront toutes déployées, il faudra lancer des tests End-To-End. Ces derniers seront aussi lancés avec un Runner Postman.

Ce document est un récapitulatif des travaux effectués sur le projet. Il sert aussi bien de SFD (Spécification Fonctionnelle Détaillée) que de SFT (Spécification Technique Détaillée) et à certain égard de DAT (Document d'Architecture Technique).

