|  |  |  |  |
| --- | --- | --- | --- |
| AQUITAINE |  |  |  |

Institut Polytechnique de Bordeaux – ENSEIRB-MATMECA  
Institut des Techniques d'Ingénieur de l'Industrie d'Aquitaine  
Formation d’ingénieur Réseaux et Systèmes d’Information

**Projet de Réservation Aérienne**

Présenté par

**BAYOU Mathieu**

**GARBAGE Maxime**

**Réseaux et Application Répartie**

Sommaire

Sommaire ii

Figures iii

Préambule 1

1. Présentation du projet 2

2. Architecture technique 3

2.1. Le choix de la technologie 3

2.2. Patterns de développement 3

2.2.1. Séparation en couche 3

2.2.2. Command/Query 4

2.3. Choix des standards de l’API 4

2.3.1. RESTful, de nombreux avantages 4

2.3.2. JSON, format objet 5

3. Fonctionnalité 7

3.1. Lister les vols 7

3.1.1. Diagramme de séquence 7

3.1.2. Documentation 7

3.2. Détail d’un vol 9

3.2.1. Diagramme de séquence 9

3.2.2. Documentation 9

3.3. Créer une réservation 12

3.3.1. Diagramme de séquence 12

3.3.2. Documentation 12

3.4. Ajout de siège à une réservation 13

3.4.1. Diagramme de séquence 13

3.4.2. Documentation 13

3.5. Mis à jour du statut d’une réservation 15

3.5.1. Diagramme de séquence 15

3.5.2. Documentation 15

3.6. Annule une réservation 16

3.6.1. Diagramme de séquence 16

3.6.2. Documentation 16

3.7. Détail d’une réservation 17

3.7.1. Diagramme de séquence 17

3.7.2. Documentation 17

4. Information complémentaire 19

4.1. Prérequis 19

4.2. POSTMAN 19

Figures

Figure 1 - Organigramme 3

Figure 2 - Les différents acteurs de Domicalis 4

Figure 3 - Ambulis application iOS 5

Figure 4 - Portail infirmier Ambulis 6

Figure 3 - Méthode Scrum 7

Figure 4 -Partenaires de Radhius 9

Figure 5 - Architecture standard des solutions Radhius 11

Figure 6 - Diagramme de Kiviat 15

Figure 9 - Description tâche deuxième année 18

Figure 10 - Logo Docker 19

Préambule

L’évolution de l’informatique a conduit à faire évoluer le type d’application le plus rependu. Quelques années auparavant le standard résidait dans l’instauration d’applications monopostes, dépendant de l’action d’un seul et unique utilisateur. Les technologies évoluant, la demande des utilisateurs s’est transformée pour aujourd’hui être orientée uniquement vers des applications « collaboratives » avec divers utilisateurs, des données accessibles depuis n’importe où et n’importe quand.

Afin de comprendre ce nouveau type d’application, nous avons réalisé un serveur applicatif de réservation aérienne. Ce document détaille les choix techniques que nous avons effectués pour réaliser ce projet en utilisant comme vous le verrez des technologies et des concepts d’aujourd’hui.

1. Présentation du projet

Le projet consiste à proposer un service de réservation aérienne. Il devra répondre aux problématiques du client et ainsi lui permettre de :

1. **Rechercher** des vols entre les aéroports de départ et d’arrivée, qui décollent à la date donnée (pour plus de simplicité, seuls les vols aller simple sont considérés). La réponse du service est une liste de dossiers contenant les informations suivantes pour chaque vol :
   * Le numéro de vol
   * Le nombre de places disponibles dans l’avion
   * Le prix
2. **Réserver** un billet pour le vol avec le numéro donné à la date donnée. Si la réservation est bien effectuée, le service doit renvoyer un identifiant (ID) de réservation. Un ID de réservation est composé d’une série unique de chiffres et de lettres générée lorsqu’un paiement est effectué pour une réservation.
3. **Annuler** la réservation avec l’ID donnée. En cas d’erreur, comme lors d’une tentative d’annulation d’une réservation d’un vol non existant ou d’une tentative d’annulation de réservation non existante, le service doit informer les clients avec des messages appropriés. L’annulation de la réservation entraine la restitution de la place au pool de places disponibles.
4. Architecture technique

# Le choix de la technologie

Pour la réalisation nous avions le choix entre deux technologies, Java et C, offrant tous les deux des approches très différentes du développement. Un choix a dû être fait pour ainsi répondre au mieux aux attentes.

Ayant tous deux travaillé dans le domaine des applications web, nous avons utilisé notre expérience pour réfléchir au meilleur choix. Nous avons constaté que les problématiques d’un projet à un autre sont régulièrement les mêmes :

* Être capable d’intercepter des requêtes
* Pouvoir lire et interpréter des formats de données (comme le JSON)
* Pouvoir renvoyer des données dans un certain format
* Gérer les appels concurrents
* Gérer la sécurité
* Gérer les performances

Depuis l’arrivée du web et l’explosion des applications réparties, ces problématiques n’ont que légèrement évolué. C’est pourquoi de nombreuses librairies existent afin de répondre à ces exigences. Afin de nous décider entre ces deux technologies, nous avons cherché à connaître lesquels de ces langages disposés du plus de support dans ce domaine, mais surtout nous nous sommes renseignés sur les utilisations de ces derniers et quelles sont leurs parts d’utilisation dans le monde en 2016. Il s’avère que depuis quelques années le C est en net recul face au Java qui se démarque largement. En effet en 2016 le JAVA a fait un bon 4,09% d’utilisation dans le monde alors que le C est en recul de 3,62% pour attendre respectivement 20,95% d’utilisation pour le JAVA et 13,22% pour le C. Compte tenu de ces dernières informations il nous a semblé évident que dans l’intérêt de nos compétences futures nous devions réaliser ce projet en JAVA.

# Patterns de développement

## Séparation en couche

Compte tenu de notre choix, nous avions la possibilité d’architecturer notre projet en utilisant des patterns de développement permettant une meilleure évolutivité et une bonne lisibilité. Dans un premier temps nous avons mis en place une séparation en couches permettant de ne pas mélanger les entités de la base de données, les traitements métiers et la couche de présentation (API).

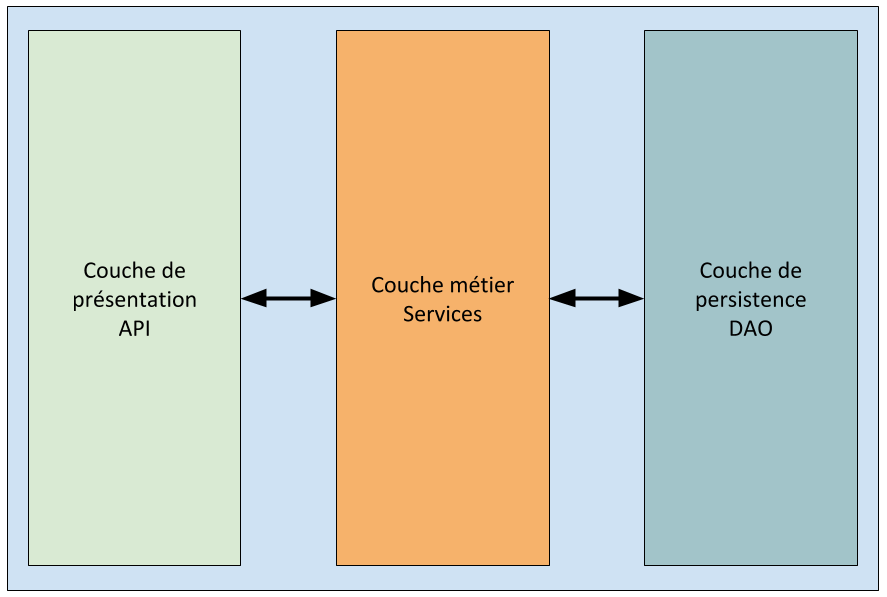


Figure - Schéma de séparation en couche

## Command/Query

Nous avons choisi l’utilisation d’un second pattern de développement qui consiste à diviser les traitements métier en deux parties :

* **Command** : Service allant altérer les données.
* **Query**: Service allant récupérer des données.

Cela aboutit à une meilleure structuration du projet et permet ainsi de ne pas se perdre dans les traitements métier.

# Choix des standards de l’API

## RESTful, de nombreux avantages

Nous avons opté pour l’utilisation du design d’API RESTful pour diverses raisons :

* **Intuitivité**: REST se rapproche au plus près des standards HTTP, il utilise les méthodes disponibles dans le standard pour effectuer toutes les actions de base telles que l’ajout, la suppression, la mise à jour ou encore la récupération de données.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| URL | POST  créer | GET  lire | PUT  Mis à jour | DELETE  suppression |
| /articles | Crée un nouvel article | Récupère la liste d’article | Met à jour tous les articles | Supprime tous les articles |
| /articles/12 | Erreur | Récupère le détail d’un article | Met à jour l’article s’il existe  Sinon erreur | Supprime l’article s’il existe  Sinon erreur |

Figure - Relation URL, méthode et action

* **Performance** : REST est « stateless », c’est-à-dire qu’entre deux appels aucun état du client n’est gardé en mémoire sur le serveur. Cela permet de maintenir les performances en ne surchargeant pas l’utilisation des ressources.
* **Économie** : REST n’impose pas de format d’échange, contrairement à SOAP qui oblige l’utilisation d’un format contraignant, lourd, et avec une faible quantité de « charge utile » en son sein. En effet les enveloppes SOAP disposent de balises superflues, mais nécessaires au standard afin d’être compréhensibles. REST quant à lui permet l’utilisation de formats tels que le JSON qui offre un très bon rapport entre la charge utile et le poids total des données envoyées.

## JSON, format objet

Depuis notre premier choix technique, le JAVA, nous avons emprunté le chemin de la programmation objet, pour rester cohérent avec nos précédents choix nous avons opté pour le format JSON qui permet une représentation objet des données aisément interprétable par l’homme. Le XML aurait pu être un concurrent, mais compte tenu de son architecture en balise, beaucoup d’éléments sont répétés alors qu’il n’apporte pas d’information supplémentaire lors d’un échange.

De plus, la part d’utilisation du format XML dans le monde est en perpétuelle baisse et ne laisse pas présager une grande expansion de l’intégration de ce dernier. Toujours dans un souci d’utilisation des technologies en accord avec l’évolution du monde informatique et son état actuel il ne nous a pas semblé approprié d’utiliser ce format.

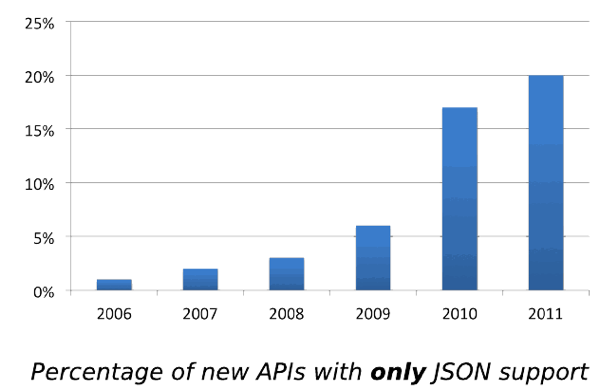


Figure - Pourcentage de nouvelles API ne supportant que le format JSON

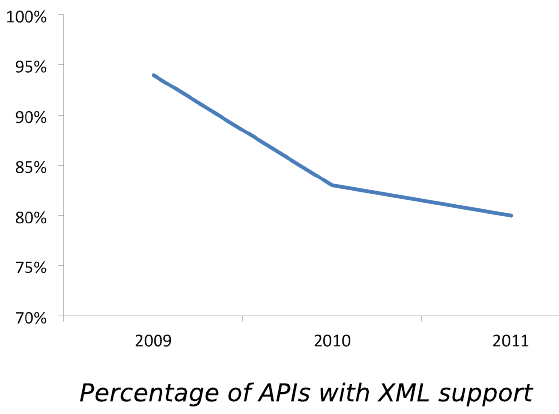
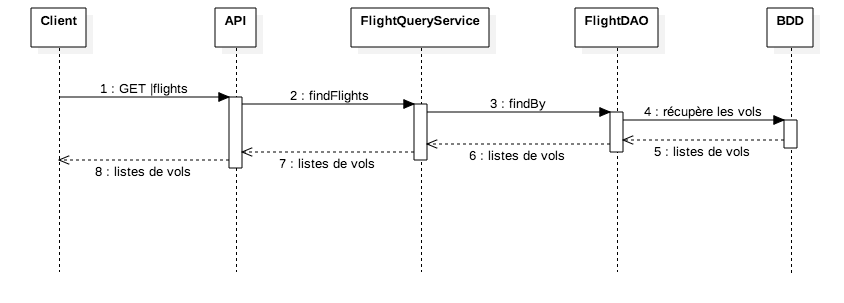


Figure - Pourcentage d'API avec un support XML

1. Fonctionnalité

# Lister les vols

## Diagramme de séquence



## Documentation

**GET /flights**

Retourne la liste des vols disponible sur le système

|  |  |
| --- | --- |
| Méthode HTTP | GET |
| Format réponse | JSON |
| Objet | Flight[] |

**Erreurs HTTP**

|  |  |
| --- | --- |
| Code | Description |
| 400 | Les paramètres de requête sont invalides |

**Query**

|  |  |  |  |
| --- | --- | --- | --- |
| Nom | Type | Obligatoire | Description |
| arrival | String | Non | Code de l’aéroport d’arrivé |
| departure | String | Non | Code de l’aéroport de départ |
| arrivalDate | String | Non | Date et heure d’arrivé au format ISO-8601 |
| departureDate | String | Non | Date et heure de départ au format ISO-8601 |
| flightDay | String | Non | Date du départ au format ISO-8601 |

**Exemple :**

http://localhost:9888/flights?arrival=BOD&departureDate= 2016-04-10T09:51:54Z

**Flight**

|  |  |  |  |
| --- | --- | --- | --- |
| Attribut | Type | Obligatoire | Description |
| id | int | Oui | Identifiant du vol dans le système |
| flightId | Int | Oui | Identifiant du vol dans iatadb |
| airLine | String | Oui | Code de la compagnie |
| arrivalAirportCode | String | Oui | Code de l’aéroport d’arrivé |
| departureAirportCode | String | Oui | Code de l’aéroport de départ |
| arrivalDate | String | Oui | Date et heure de l’arrivé (ISO-8601) |
| departureDate | String | Oui | Date et heure du départ (ISO-8601) |
| mseatCabinInformation | SeatCabinInformation | Oui | Information sur les siège disponible de la cabine de type « M » |
| yseatCabinInformation | SeatCabinInformation | Oui | Information sur les siège disponible de la cabine de type « Y » |
| jseatCabinInformation | SeatCabinInformation | Oui | Information sur les siège disponible de la cabine de type « J » |

**SeatCabinInformation**

|  |  |  |  |
| --- | --- | --- | --- |
| Attribut | Type | Obligatoire | Description |
| cabinClass | String | Oui | Classe de la cabine (Y,M,J) |
| Quantity | Int | Oui | Nombre de siège encore disponible |

**Exemple de réponse :**

[

{

"id": 162,

"flightId": 825,

"airLine": "IR",

"arrivalAirportCode": "IQT",

"departureAirportCode": "PCL",

"arrivalDate": "2016-05-07T09:40:00Z",

"departureDate": "2016-05-06T23:10:00Z",

"mseatCabinInformation": {

"cabinClass": "M",

"quantity": 9

},

"jseatCabinInformation": {

"cabinClass": "J",

"quantity": 32

},

"yseatCabinInformation": {

"cabinClass": "Y",

"quantity": 13

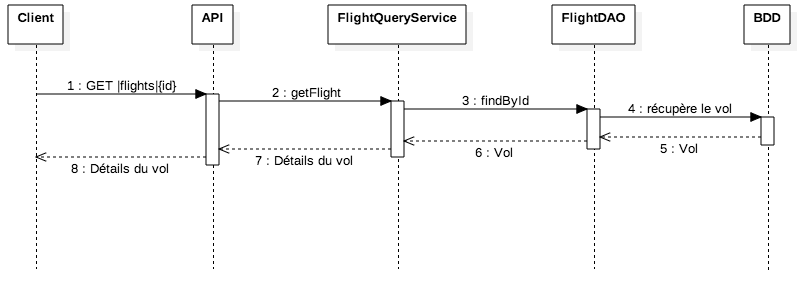
}

}

]

# Détail d’un vol

## Diagramme de séquence



## Documentation

**GET /flights/{flightid}**

Retourne les informations détaillées d’un vol disponible sur le système

|  |  |
| --- | --- |
| Méthode HTTP | GET |
| Format réponse | JSON |
| Objet | FlightDetail |

**Erreurs HTTP**

|  |  |
| --- | --- |
| Code | Description |
| 404 | Le vol n’existe pas |
| 400 | Les paramètres de requête sont invalides |

**Requête**

|  |  |  |
| --- | --- | --- |
| Nom | Type | Description |
| flightid | **Int** | **Identifiant dans le système** |

**Exemple :**

http://localhost:9888/flights/12

**FlightDetail**

|  |  |  |  |
| --- | --- | --- | --- |
| Attribut | Type | Obligatoire | Description |
| id | int | Oui | Identifiant du vol dans le système |
| flightId | Int | Oui | Identifiant du vol dans iatadb |
| airLine | String | Oui | Code de la compagnie |
| arrivalAirportCode | String | Oui | Code de l’aéroport d’arrivé |
| departureAirportCode | String | Oui | Code de l’aéroport de départ |
| arrivalDate | String | Oui | Date et heure de l’arrivé (ISO-8601) |
| departureDate | String | Oui | Date et heure du départ (ISO-8601) |
| mseatCabinInformation | SeatCabinInformation | Oui | Information sur les siège disponible de la cabine de type « M » |
| yseatCabinInformation | SeatCabinInformation | Oui | Information sur les siège disponible de la cabine de type « Y » |
| jseatCabinInformation | SeatCabinInformation | Oui | Information sur les siège disponible de la cabine de type « J » |

**SeatCabinInformation**

|  |  |  |  |
| --- | --- | --- | --- |
| Attribut | Type | Obligatoire | Description |
| cabinClass | String | Oui | Classe de la cabine (Y,M,J) |
| priceWithoutTaxes | Float | Oui | Prix d’un siège sans les taxes |
| taxesPrice | Float | Oui | Prix des taxes pour un siège |
| Quantity | Int | Oui | Nombre de siège encore disponible |

**Exemple de réponse :**

{

"id": 240,

"flightId": 533,

"airLine": "IR",

"arrivalAirportCode": "ALG",

"departureAirportCode": "CDG",

"arrivalDate": "2016-05-23T14:00:00Z",

"departureDate": "2016-05-23T04:30:00Z",

"mseatCabinInformation": {

"cabinClass": "M",

"quantity": 9,

"priceWithoutTaxes": 126.25,

"taxesPrice": 58.51

},

"jseatCabinInformation": {

"cabinClass": "J",

"quantity": 26,

"priceWithoutTaxes": 764.87,

"taxesPrice": 53.86

},

"yseatCabinInformation": {

"cabinClass": "Y",

"quantity": 0,

"priceWithoutTaxes": 0,

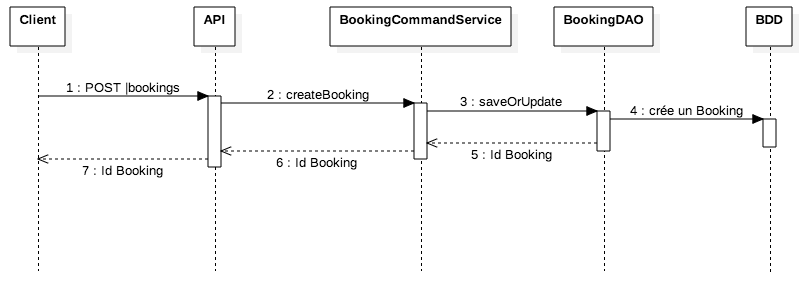
"taxesPrice": 0

}

}

# Créer une réservation

## Diagramme de séquence



## Documentation

**POST /bookings**

Crée une nouvelle réservation dans le système

|  |  |
| --- | --- |
| Méthode HTTP | POST |
| Format requête | JSON |
| Objet requête | SaveBooking |

**Erreurs HTTP**

|  |  |
| --- | --- |
| Code | Description |
| 400 | Les paramètres de requête sont invalides |

**Exemple :**

http://localhost:9888/bookings

**SaveBooking**

|  |  |  |  |
| --- | --- | --- | --- |
| Attribut | Type | Obligatoire | Description |
| flightId | Int | Oui | Identifiant du vol dans iatadb |
| cabinClass | String | Oui | Classe souhaitée pour le siège |
| quantity | int | Oui | Nombre de sièges à réserver |
| customer | Customer | Oui | Information sur le client |

**Customer**

|  |  |  |  |
| --- | --- | --- | --- |
| Attribut | Type | Obligatoire | Description |
| firstName | String | Oui | Prénom du client |
| lastName | String | Oui | Nom du client |

**Exemple de requête:**

{

"flightId": 240,

"cabinClass": "J",

"quantity": 2,

"customer":{

"firstName": "John",

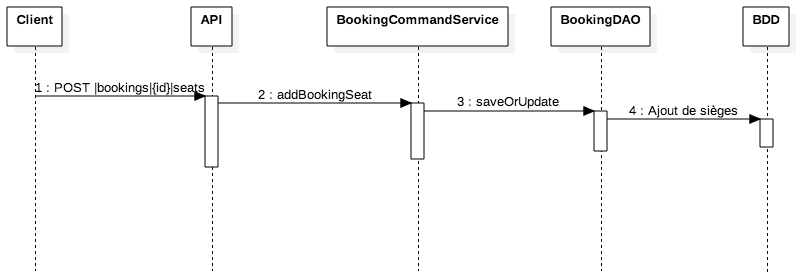
"lastName": "Smith"

}

}

# Ajout de siège à une réservation

## Diagramme de séquence



## Documentation

**POST /bookings/{bookingid}/seats**

Ajoute des sièges supplémentaires à la réservation **(la catégorie M,Y,J ne peut être changé)**

|  |  |
| --- | --- |
| Méthode HTTP | POST |
| Format requête | JSON |
| Objet requête | AddBookingSeat |

**Erreurs HTTP**

|  |  |
| --- | --- |
| Code | Description |
| 400 | Les paramètres de requête sont invalides |
| 404 | La réservation n’existe pas |

**Requête**

|  |  |  |
| --- | --- | --- |
| Nom | Type | Description |
| bookingid | **Int** | **Identifiant dans le système** |

**Exemple :**

http://localhost:9888/bookings/12/seats

**AddBookingSeat**

|  |  |  |  |
| --- | --- | --- | --- |
| Attribut | Type | Obligatoire | Description |
| quantity | int | Oui | Nombre de sièges supplémentaires à réserver |

**Exemple de requête:**

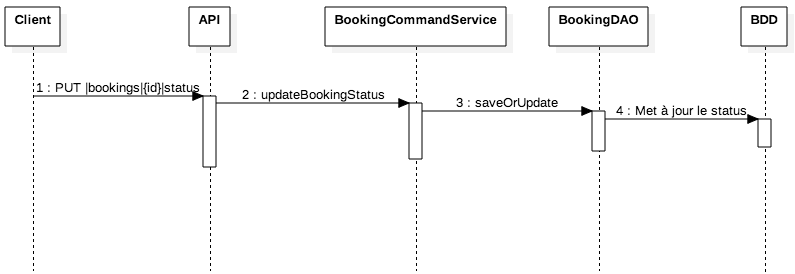
{

"quantity": 2

}

# Mis à jour du statut d’une réservation

## Diagramme de séquence



## Documentation

**PUT /bookings/{bookingid}/status**

Permet de mettre à jour le statut d’une réservation

|  |  |
| --- | --- |
| Méthode HTTP | POST |
| Format requête | JSON |
| Objet requête | UpdateBookingStatus |

**Erreurs HTTP**

|  |  |
| --- | --- |
| Code | Description |
| 400 | Les paramètres de requête sont invalides |
| 404 | La réservation n’existe pas |

**Requête**

|  |  |  |
| --- | --- | --- |
| Nom | Type | Description |
| bookingid | **Int** | **Identifiant dans le système** |

**Exemple :**

http://localhost:9888/bookings/12/status

**UpdateBookingStatus**

|  |  |  |  |
| --- | --- | --- | --- |
| Attribut | Type | Obligatoire | Description |
| status | String | Oui | Nouveau statut de la réservation (PENDING\_CONFIRMATION, OK) |

**Exemple de requête:**

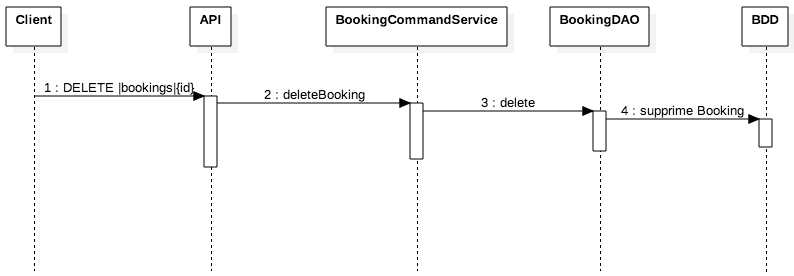
{

"status": "OK"

}

# Annule une réservation

## Diagramme de séquence



## Documentation

**DELETE /bookings/{bookingid}**

Supprime une réservation

|  |  |
| --- | --- |
| Méthode HTTP | POST |
| Format requête | JSON |

**Erreurs HTTP**

|  |  |
| --- | --- |
| Code | Description |
| 400 | Les paramètres de requête sont invalides |
| 404 | La réservation n’existe pas |

**Requête**

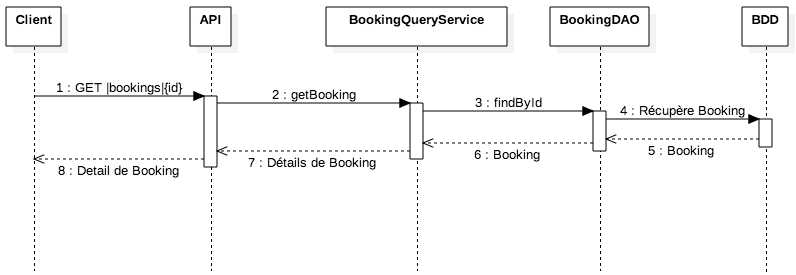
|  |  |  |
| --- | --- | --- |
| Nom | Type | Description |
| bookingid | **Int** | **Identifiant dans le système** |

**Exemple :**

http://localhost:9888/bookings/12

# Détail d’une réservation

## Diagramme de séquence



## Documentation

**GET /bookings/{bookingid}**

Retourne les informations détaillées d’une réservation

|  |  |
| --- | --- |
| Méthode HTTP | GET |
| Format réponse | JSON |
| Objet | Booking |

**Erreurs HTTP**

|  |  |
| --- | --- |
| Code | Description |
| 404 | La réservation n’existe pas |
| 400 | Les paramètres de requête sont invalides |

**Requête**

|  |  |  |
| --- | --- | --- |
| Nom | Type | Description |
| bookingid | **Int** | **Identifiant dans le système** |

**Exemple :**

http://localhost:9888/bookings/12

**Booking**

|  |  |  |  |
| --- | --- | --- | --- |
| Attribut | Type | Obligatoire | Description |
| id | int | Oui | Identifiant de la réservation dans le système |
| flightId | Int | Oui | Identifiant du vol dans le système |
| cabinClass | String | Oui | Classe de la cabine des siège (M,Y,J) |
| quantity | String | Oui | Quantité de sièges réservés |
| price | String | Oui | Prix total de la réservation |
| status | String | Oui | Statut de la réservation (PENDING\_CONFIRMATION, OK) |
| customer | Customer | Oui | Information du client |

**Customer**

|  |  |  |  |
| --- | --- | --- | --- |
| Attribut | Type | Obligatoire | Description |
| firstName | String | Oui | Prénom du client |
| lastName | String | Oui | Nom du client |

**Exemple de réponse :**

{

"id": 2,

"flightId": 240,

"cabinClass": "J",

"quantity": 2,

"prices": 1637.46,

"status": "PENDING\_CONFIRMATION",

"customer": {

"firstName": "John",

"lastName": "Smith"

}

}

1. Information complémentaire

# Prérequis

Nous avons fait le choix d’extraire les données du fichier iatadb.dat et de les sauvegarder dans une base données relationnelle. Il est donc nécessaire avant de commencer à se servir de l’application de faire appel à un service d’extraction de ces données pour remplir la base de données. Il s’agit d’une ressource en POST : /admin/prepare .

# POSTMAN

Nous avons préparé une liste de ressource préconfigurée pour l’outil POSTMAN, afin que vous puissiez facilement tester l’application. Vous trouverez l’outil à cette adresse : https://chrome.google.com/webstore/detail/postman/fhbjgbiflinjbdggehcddcbncdddomop .

Le fichier de configuration à intégrer à l’outil se trouve à la racine du projet, il se nomme : flightbooking.json.postman\_collection