

Rapport API

Gestion de demandes de crédit

Rémy BANTON

02/02/2018

Contenu

Introduction.....	2
1. Réponse aux besoins de conception	3
2. Règles de gestion.....	4
3. Réalisation	5
4. Sécurité.....	7
5. Cas d'utilisation	9
Prérequis	9
Token	9
Erreurs	10
Visualisation des demandes	10
Visualisation des actions	11
Ajout d'une demande.....	12
Visualisation des détails d'une demande par un externe	13
Visualisation des détails d'une demande par un interne.....	13
Modification d'une demande par un externe	14
Ajout d'une action	15
Clôturer une demande	17
Modification d'une action	18
Conclusion	20

Introduction

L'objectif de ce projet était de mettre en place une API RESTful afin de permettre à un organisme bancaire de gérer ses demandes de crédit. Des contraintes et spécifications étaient spécifiées telles que la gestion des états des demandes lors du cheminement du processus de demande de crédit ou encore l'aspect sécuritaire à travers l'authentification par token. Nous allons détailler la réponse qui a été donnée aux besoins de conception, puis la réalisation par le biais des choix techniques opérés et enfin la sécurité au travers de OAuth et des JWT (Json Web Token).

1. Réponse aux besoins de conception

Afin d'apporter une réponse il convient de replacer le cadre de la logique métier spécifiée dans le sujet. Les clients déposent des demandes et à chaque demande est associée des actions. Ces actions sont prises par des opérateurs de l'organisme bancaire. A chaque action prise le statut de la demande est modifié. Ainsi cela permet d'avoir un suivi du traitement d'une demande puisque chaque action effectuée sur une demande est répertoriée. Les états d'une demande suivent cette nomenclature :

- **Dépôt** – Etat initial après qu'un client est déposé sa demande.
- **Début** – Etat une fois qu'un employé de la banque a validé.
- **Étude** – La demande est à l'étude
- **Décision** – la demande a été étudiée, la décision est en attente de validation
 - **Acceptation** la demande est acceptée, et le demandeur est notifié
 - **Rejet** la demande est rejetée, et le demandeur est notifié
- **Fin** - Etat final de la demande, quelle que soit la décision.

Cette solution respecte entièrement les principes GET, POST, PUT. La création des éléments est réalisée avec la méthode POST et les modifications via la méthode PUT.

En suivant cette logique, les liens suivants ont été établis afin de répondre aux besoins de conception :

Méthode	URI	Objectif	Body	Retour
GET	/demandes	Liste des demandes avec les liens HATEOAS des actions associés.		200 (OK)
GET	/demandes/{id}	Détails d'une demande		200 (OK)
POST	/demandes	Sauvegarde d'une nouvelle demande	<pre>{ "nom": "Durand", "prenom": "Paul", "adresse": "Paris", "revenus": 25, "etatcourantdemande": "Debut", "montantcreditdemande": 25, "dureecredit": 6, "datenaissance": "10-09-1987" }</pre>	201 (Created)

PUT	/demandes/{id}	Modification d'une demande	{ "nom": "Dupont", "prenom": "Jean-Michel", "adresse": "Paris", "revenus": 25, "datenaissance": "10-09-1992", "dureecredit": 6, "montantcreditdemande": 25 }	204 (No Content)
DELETE	/demandes/{id}	Clôture d'une demande		204 (No Content)
GET	/demandes/{id}/actions	Liste des actions d'une demande		200 (OK)
POST	/demandes/{id}/actions	Ajout d'une action pour une demande	{ "nom": "Acceptation", "personnecharge": "Moi" }	201 (Created)
GET	/demandes/{id}/actions/{idAction}	Retourne une action spécifique pour une demande donnée		200 (OK)
PUT	/demandes/{id}/actions/{idAction}	Modifie une action spécifique pour une demande donnée	{ "personnecharge": "Moi" }	204 (No Content)

La consultation d'une demande génère des liens HATEOAS permettant de visualiser les actions liées à cette demande ainsi qu'un lien sur la demande elle-même pour effectuer des modifications.

```
{
  "nom": "Durand",
  "prenom": "Paul",
  "adresse": "Paris",
  "revenus": 25,
  "token": "eyJhbGciOiJIUzUxMiJ9.eyJzdWIiOiJWZW90b3IiLCJleHAiOiJlMjI1MjYyNTF9.CXZw8T3VQUqI3FwpxCjkjresT_WhWZB5n3SGOUqTA5WU7NMUB0K02a3pgFE0Zk4CwAb_sR2Z9abpw3b8pxzng",
  "montantcreditdemande": 25,
  "etatcourantdemande": "Etude",
  "dureecredit": 6,
  "datenaissance": "10-09-1987",
  "_links": {
    "self": {
      "href": "http://localhost:8082/demandes/f9eb4f8b-60b9-4c01-9ac7-d9aa893cb1b6"
    },
    "Historique actions": [
      {
        "href": "http://localhost:8082/actions/58"
      },
      {
        "href": "http://localhost:8082/actions/57"
      }
    ],
    "Collection": {
      "href": "http://localhost:8082/demandes(?status)",
      "templated": true
    }
  }
}
```

La gestion de la sécurité via le service OAuth sera évoquée dans un autre point et nous allons concentrer notre propos sur les règles de gestion.

2. Règles de gestion

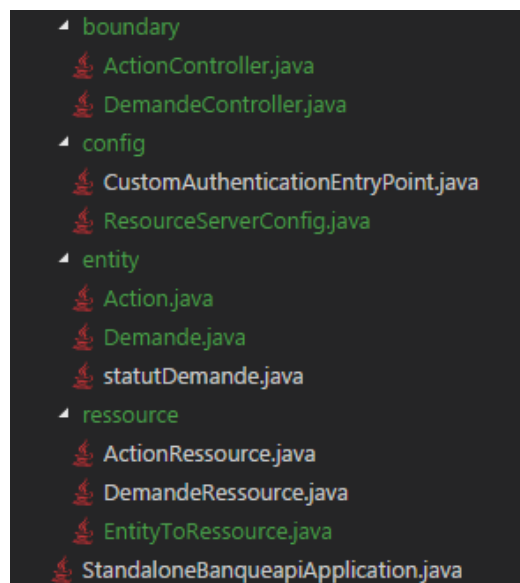
Toutes les règles de gestion ci-dessous se basent par rapport à l'échelle des étapes d'une demande évoquée dans la première partie.

- Une demande peut être tant que son statut n'a pas atteint l'étape Etude
- Une demande en état Fin ne peut plus recevoir aucune modification ni aucune action, mais peut être consultée. Cette étape peut être atteinte en cas de rejet/acceptation de la demande ou encore en cas de clôture via une requête DELETE.
- L'accès pour les internes à l'organisme doit se faire via un access token OAuth
- L'accès pour les clients externes se fait via un JWT propre à chaque demande
- La mise à jour d'une action est limitée à la modification de la personne en charge, afin de ne pas perturber la cohérence du workflow d'une demande.
- Une action ne peut être modifiée que si celle-ci est en cours (Dernière action enregistrée pour la demande)

3. Réalisation

Pour que l'**API** soit fonctionnelle **deux services** ont été mis en place. L'**un qui gère la logique métier** et que l'on va détailler maintenant et l'**autre** qui se charge de la **partie authentification et sécurité**.

Comme énoncé dans la partie précédente ce projet se compose de deux entités principales Demande et Action. Afin de répondre de réaliser la réponse aux besoins de conception présentée l'implémentation suivante a été choisie :



Décomposons un peu la logique pour mieux comprendre :

- Le package **boundary** correspond aux controllers pour les actions et les demandes. Ainsi pour chaque requete, on définit les opérations à effectuer puis le code d'erreur renvoyé à la fin du traitement.
- Le package **config** correspond à la configuration de connexion avec le service d'authentification. On spécifie notamment dans celui-ci les paths qui nécessitent une authentification OAuth pour les internes et un JWT pour les clients externes. Le moyen utilisé pour différencier les deux types de profils d'utilisateur a été de définir deux paths différents afin d'affiner les autorisations en fonction du path qui appelle. Ainsi cela donne pour la modification d'une demande :
 - Interne : /demandes/0904857e-992b-47f6-bc38-e6f5cbb62505 avec un token OAuth
 - Externe : /demandes/externe/0904857e-992b-47f6-bc38-e6f5cbb62505 avec le token que le client a reçu lorsqu'il a déposé sa demande dans le header.
- La package **entity** correspond aux entités exigées par le sujet Demande et Action mais aussi une **Enum statutDemande** qui référence les statuts possible d'une demande. Les entités se décomposent de la manière suivante :
 - Demande
 - id de la demande (génééré par l'API, en respectant UUID)
 - nom
 - prénom
 - adresse
 - date de naissance
 - revenus sur les 3 dernières années
 - montant du crédit demandé
 - durée du crédit
 - état courant de la demande
 - liste des actions associées à la demande
 - Action
 - id de l'action
 - nom de l'action
 - personne en charge de l'action
 - état de l'action (en cours/terminée)
 - date d'exécution de l'action
 - statutDemande

```

Depot(1),
Debut(2),
Etude(3),
Decision(4),
Acceptation(5),
Rejet(6),
Fin(7);
private Integer numero;
statutDemande(Integer numero){
    this.numero = numero;
}
public Integer getNumero(){
    return this.numero;
}

```

Le numéro est utilisé afin de faciliter les vérifications concernant le respect des étapes du processus de la demande. Ainsi on ne peut passer de l'étape 2 à l'étape 5 par exemple. Il y a de cette manière une gradation, une échelle.

- Le package ressource correspond à l'accès aux ressource et aux traitements (mise à jour, insertion) effectués. Ceci est réalisé dans la classe EntityToRessource afin de ne pas encombrer le controller et garder de la visibilité.

Mon idée de départ pour la réalisation a été de passer par deux Micro-Service Action et Demande communiquant entre eux avec Feign et Eureka comme vu en TD. J'ai réussi à mettre en place cette communication cependant suite à votre remarque concernant l'absence de contrainte pour la séparation des services j'ai préféré n'en faire qu'un seul.

4. Sécurité

Orientons notre explication vers le service OAuth et plus largement ses interactions avec le service banqueapi.

Afin de gérer l'aspect sécurité au niveau de l'application, deux manières d'accéder aux fonctionnalités ont été déterminé :

- **Interne** qui correspond aux employés de l'organisme. Ils récupèrent un jeton d'authentification Oauth et une fois authentifié ils ont accès à l'entièreté des fonctionnalités de gestion des demandes et des actions.

- **Externe** qui correspond aux clients. Une fois leurs identités vérifiées par un procédé cité ci-dessous ils ont la possibilité de visualiser les demandes et les actions qui leur sont reliés par le biais d'un token généré lors de la création de leurs demandes. Ils ont par conséquent restreint aux fonctionnalités de l'API par rapport aux internes.

Rappelons rapidement la structure des liens pour effectuer cette différenciation :

- Interne : /demandes/id
- Externe : /demandes/externe/id

Au niveau des entry points, une règle spécifie l'autorisation pour les liens externes :

```
.antMatchers("/demandes/externe/{\\.*}").permitAll()
```

La génération du token est actuellement réalisée en utilisant un utilisateur « Victor » et un mot de passe « Hugo » avec un secret « thesecret », cela permet de simplifier l'utilisation de l'API pour notre cas. Ci-dessous l'exemple du header de la requête d'appel au service OAuth à l'adresse localhost:9191/uaa/oauth/token :

<input checked="" type="checkbox"/>	password	Hugo
<input checked="" type="checkbox"/>	grant_type	password
<input checked="" type="checkbox"/>	scope	read
<input checked="" type="checkbox"/>	client_id	html5
<input checked="" type="checkbox"/>	client_secret	thesecret
<input checked="" type="checkbox"/>	scope	write
<input checked="" type="checkbox"/>	username	Victor

Afin de pleinement bénéficier du potentiel d'un tel système d'authentification il aurait été intéressant de lier l'utilisateur à sa demande et le contrôler par la suite. Il aurait été possible d'utiliser le nom d'utilisateur contenu dans le body pour générer le token et ainsi faire des contrôles plus poussés avec des secrets différents. Ainsi plusieurs utilisateurs pourraient s'authentifier et utiliser l'application. La vérification s'effectuerait au niveau de la partie utilisateur du token déchiffré et non un check direct du header.

Une remarque est à faire concernant la visibilité des paramètres de la requête de demande de token notamment sur Postman. En effet OAuth impose l'utilisation de HTTPS pour les échanges entre le client et le serveur d'autorisation du fait des données sensibles qui transitent entre les deux (jeton d'accès, éventuellement des identifiants et des mots de passe).

5. Cas d'utilisation

Afin de rendre la compréhension du workflow au sein de mon API plus aisée, un scénario d'utilisation va être exposé. Cela va permettre de traiter toutes les étapes du cycle de traitement d'une demande, de la réception à son achèvement, en prenant soin de montrer les comportements en cas de non-respect d'une des règles de gestion. Cette démonstration s'effectue via l'outil Postman. **Un fichier de bookmarks est fourni dans l'archive de rendu afin de faciliter les tests.**

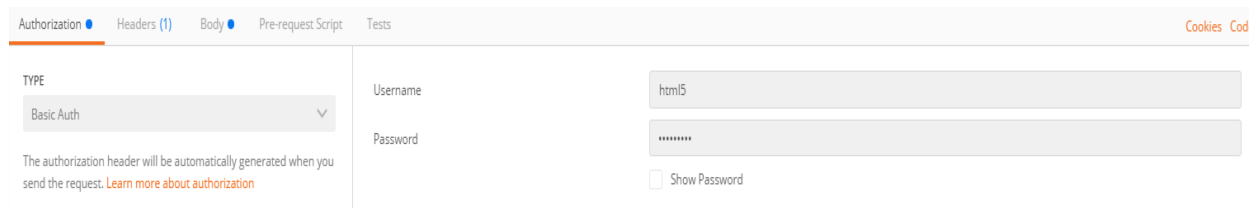
Prérequis

Pour utiliser cette API il est nécessaire de démarrer les services oauth-service et standalone-banqueapi. Pour cela il suffit de taper les commandes dans les répertoires respectifs :

- `mvn -DskipTests clean install`
- `java -jar target/oauth-service-0.0.1-SNAPSHOT.jar`
- `java -jar target/standalone-banqueapi-0.0.1-SNAPSHOT.jar`

Token

Avant de procéder à une requête il convient de demander un token au service OAuth qui est démarré sur le port 9191. Pour cela on effectue une requête POST et on remplit le body comme ci-dessous :



The screenshot shows the Postman interface with the 'Authorization' tab selected. On the left, under 'TYPE', 'Basic Auth' is chosen. Below it, a note states: 'The authorization header will be automatically generated when you send the request. [Learn more about authorization](#)'. On the right, the 'Username' field contains 'html5' and the 'Password' field contains a series of dots. A 'Show Password' checkbox is located below the password field and is currently unchecked. At the top right of the interface, 'Cookies' and 'Code' links are visible.

POST ▼ http://localhost:9191/uaa/oauth/token

Authorization ● Headers (2) ● Body ● Pre-request Script Tests

☐ form-data ☒ x-www-form-urlencoded ☐ raw ☐ binary

	Key	Value
<input checked="" type="checkbox"/>	password	Hugo
<input checked="" type="checkbox"/>	grant_type	password
<input checked="" type="checkbox"/>	scope	read
<input checked="" type="checkbox"/>	client_id	html5
<input checked="" type="checkbox"/>	client_secret	thesecret
<input checked="" type="checkbox"/>	scope	write
<input checked="" type="checkbox"/>	username	Victor

En renseignant ce token dans le header de chaque requête nous pouvons à présent effectuer toutes les actions autorisées pour les internes à l'organisme bancaire. A noter que son expiration a été volontairement poussé à son maximum pour les besoins de la démonstration.

Erreurs

En cas de non renseignement du token généré précédemment l'utilisateur interne se verra refusé toutes requêtes avec un code 401 Unauthorized.

```
{
  "timestamp": 1522609543921,
  "status": 401,
  "error": "Unauthorized",
  "message": "Accès rejeté",
  "path": "/demandes/1/actions/55"
}
```

Figure 2 : Message lorsque l'utilisateur est non-authentifié

Visualisation des demandes

Tout d'abord nous allons visualiser les demandes déjà existantes. Pour cela on fait une requête **GET** via l'url **http://localhost:8082/demandes**. Le résultat obtenu est le suivant avec le code 200 OK:

```

{
  "_embedded": {
    "demandes": [
      {
        "nom": "Dupont",
        "prenom": "Jean-Michel",
        "adresse": "Paris",
        "revenus": 25,
        "token": "eyJhbGciOiJIUzUxMiJ9.eyJzdWIiOiJwZW50b3R1LCJleHAiOiJ1MjQyNTF9.CXZw8TJVQUi3FmpxGjkjresT_WhWZ85n3sGOUqTA5WU7NMUB0K02aJpgFE0Zk4CwAb_sR2Z9abpW3b8pXxzng",
        "etatcourantdemande": "Debut",
        "montantcreditdemande": 25,
        "datenaissance": "10-09-1992",
        "dureecredit": 6,
        "iddemande": "1",
        "_links": {
          "self": {
            "href": "http://localhost:8082/demandes/1"
          },
          "Historique actions": {
            "href": "http://localhost:8082/demandes/1/actions/55"
          }
        }
      }
    ]
  },
  "_links": {
    "self": {
      "href": "http://localhost:8082/demandes/?status",
      "templated": true
    }
  }
}

```

Figure 3 : Liste des demandes enregistrées - GET

Dans la section « Historique actions » on retrouve les liens HATEOAS des actions rattachées à chaque demande.

Il est possible de filtrer la liste des demandes en fonction des états courants des demandes en ajoutant un request param à l'url de la requête de cette manière : **http://localhost:8082/demandes?status=Fin.**

Visualisation des actions

Visualisons l'action associée à cette demande en cliquant sur le lien et en renseignant le token requis. On obtient ceci en exécutant une requête GET via l'url :

http://localhost:8082/demandes/1/actions/55.

GET ▾	http://localhost:8082/demandes/1/actions/55
-------	---

```

{
  "numero": 2,
  "nom": "Debut",
  "personnecharge": "Titi",
  "etat": "En cours",
  "date": "30-07-1985"
}

```

Figure 4 : Détails d'une action - GET

A noter qu'une vérification est opérée afin de voir si les détails de l'action que l'on souhaite obtenir sont bien rattachés à la demande.

Ajout d'une demande

Pour créer une demande toutes les informations concernant une demande doivent être fournies à l'exception qui est généré dynamiquement via un UUID. Ci-dessous un exemple de body pour sauvegarder une demande via l'url **http://localhost:8082/demandes** en exécutant une requête POST :

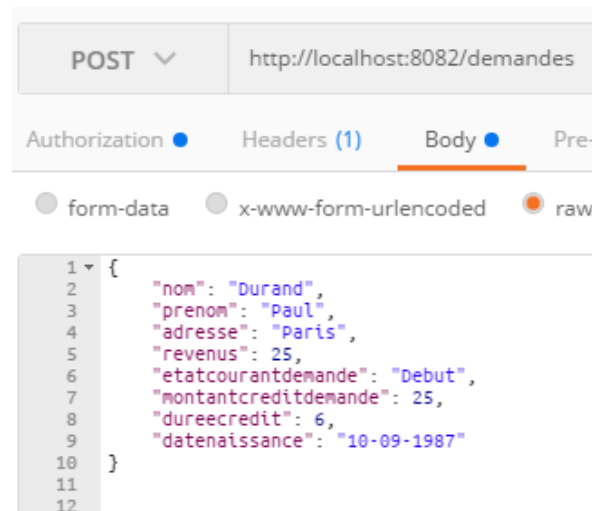


Figure 5 : Ajout d'une demande - POST

Si l'on visualise maintenant la liste des demandes on voit que notre ajout a bien été effectué :



Figure 6 : Liste des demandes avec la nouvelle demande ajoutée – GET

Le champs token est la clé d'accès aux utilisateurs externe.

Visualisation des détails d'une demande par un externe

Pour accéder aux détails d'une demande via l'extérieure le client doit spécifier l'url de cette forme : **http://localhost:8082/demandes/externe/dee66939-d205-4b47-bdcb-9047ff9d948e** comme expliqué dans la partie sécurité. Il a simplement besoin de renseigner dans le header le token du champs token de la figure 6.

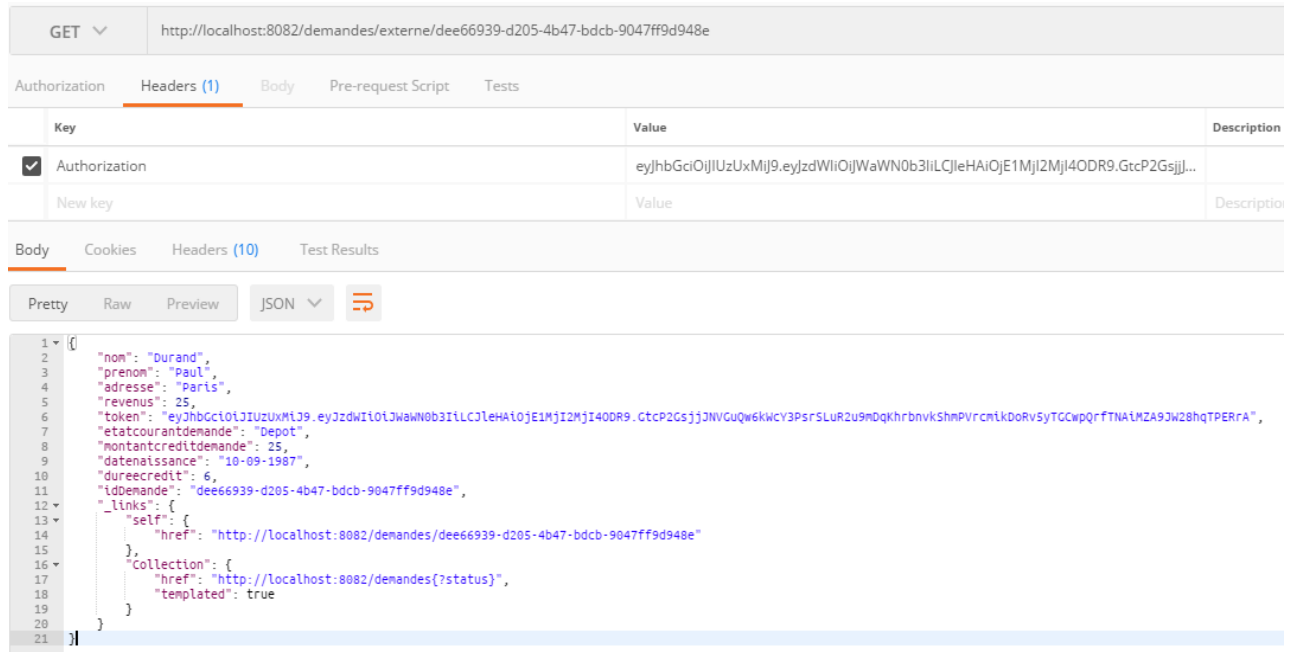


Figure 7 : Détails d'une demande pour un utilisateur externe

Visualisation des détails d'une demande par un interne

Pour accéder aux détails d'une demande en tant qu'utilisateur interne on doit spécifier l'url de cette forme : **http://localhost:8082/demandes/dee66939-d205-4b47-bdcb-9047ff9d948e**.

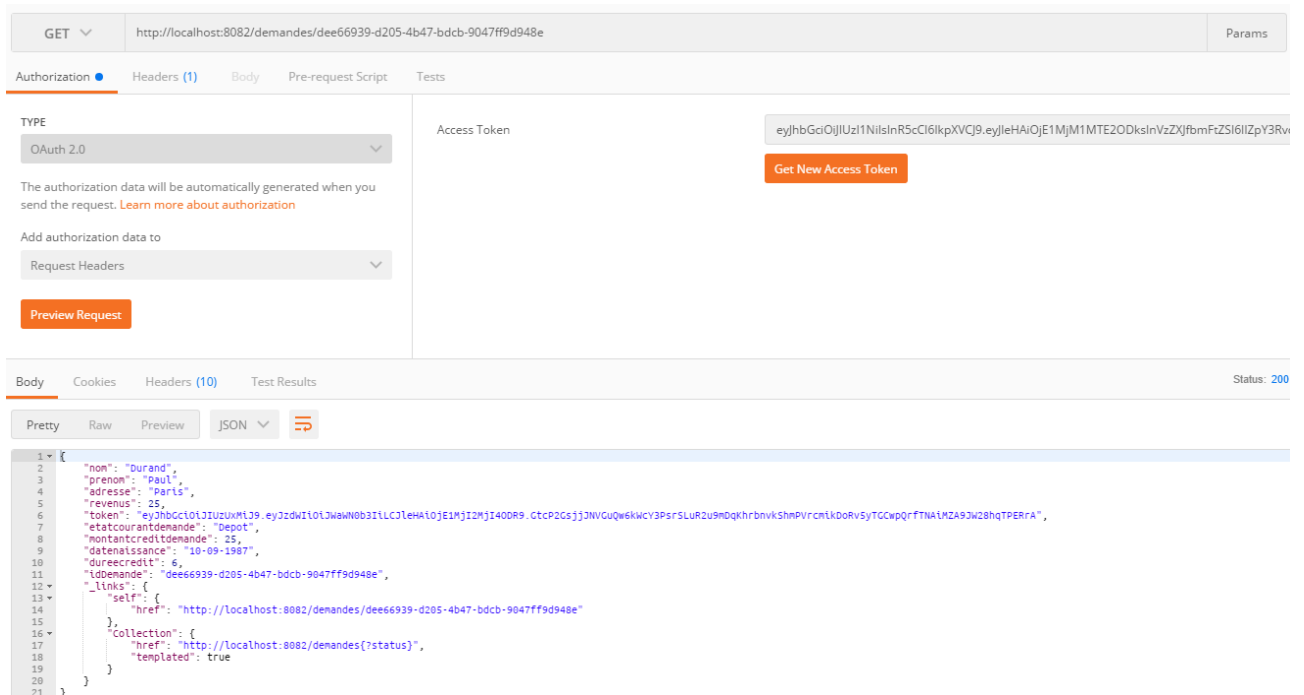


Figure 8 : Détails d'une demande pour un utilisateur interne

Modification d'une demande par un externe

A présent modifions la demande que l'on a créé précédemment. Conformément au sujet la modification d'une demande est possible uniquement si le statut de celle-ci est avant l'étape **Etude**. Ce qui est le cas ici. Ci-dessous un exemple de body pour modifier une demande via l'url `http://localhost:8082/demandes/externe/dee66939-d205-4b47-bdcb-9047ff9d948e` en exécutant une requête PUT :

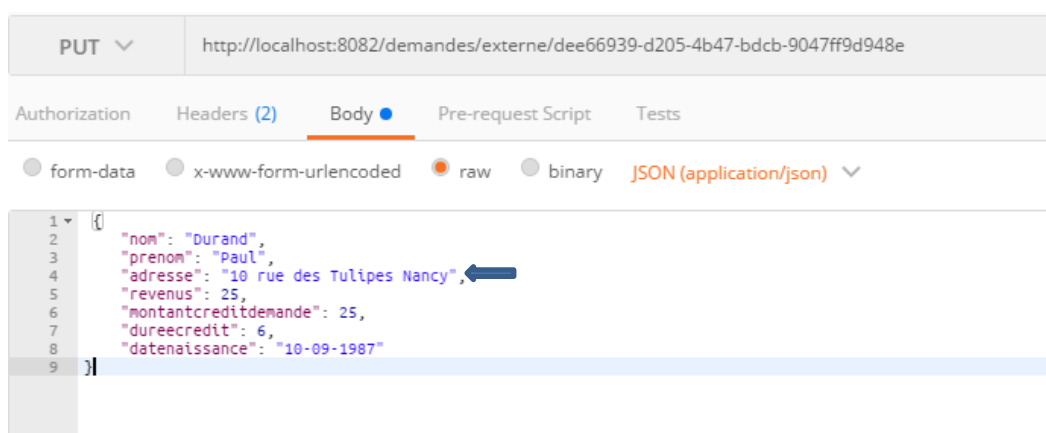


Figure 9 : Modification d'une demande par un externe

Si l'on vérifie au niveau des détails de la demande on voit que le changement a bien été répercuté :

```
{
  "nom": "Durand",
  "prenom": "Paul",
  "adresse": "10 rue des Tulipes Nancy",
  "revenus": 25,
  "token": "eyJhbGciOiJIUzUxMiJ9.eyJzdWIiOiJWYWNoYm93IiwiaWF0IjE1MjI2MjI4ODR9.",
  "etatcourantdemande": "Depot",
  "montantcreditdemande": 25,
  "datenaissance": "10-09-1987",
  "dureecredit": 6,
  "idDemande": "dee66939-d205-4b47-bdcb-9047ff9d948e",
  "_links": {
    "self": {
      "href": "http://localhost:8082/demandes/dee66939-d205-4b47-bdcb-9047"
    },
    "Collection": {
      "href": "http://localhost:8082/demandes{?status}",
      "templated": true
    }
  }
}
```

Figure 10 : Détails de la demande après modification

Ajout d'une action

En adéquation avec le sujet la création d'une action correspond au passage d'un état à un autre pour la demande associée. Prenons toujours notre même en exemple, celle-ci a actuellement pour état courant « Dépôt ». Elle a donc besoin d'être validé afin de pouvoir passer à l'état suivant qui est « Début » d'après les contraintes fournies. Pour cela nous effectuons une requête POST **<http://localhost:8082/demandes/dee66939-d205-4b47-bdcb-9047ff9d948e/actions>**.

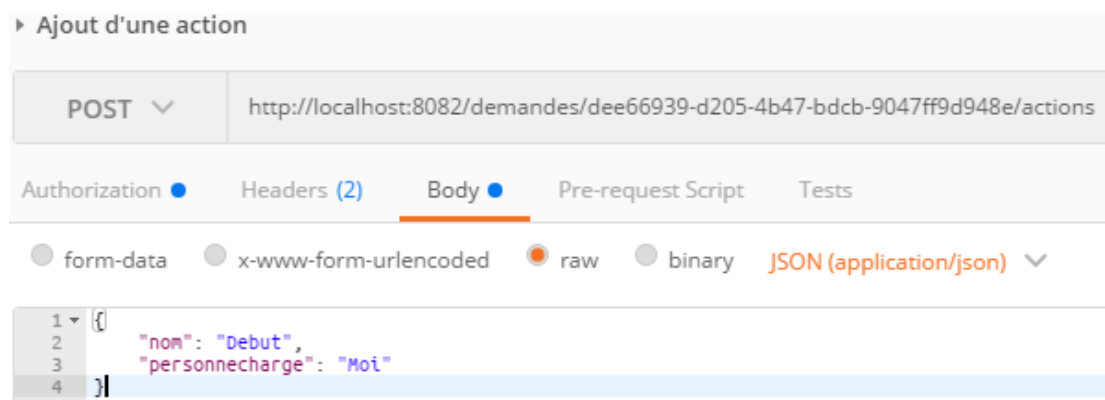


Figure 11 : Ajout d'une action pour une demande

Le statut de la demande change et une action est créée. Si il y a une erreur ou si on veut créer une action qui ne suit pas le flow, par exemple de Début à Rejet alors aucune action n'est créé.


```

{
  "nom": "Durand",
  "prenom": "Paul",
  "adresse": "10 rue des Tulipes Nancy",
  "revenus": 25,
  "token": "eyJhbGciOiJIUzUxMiJ9.eyJzdWIiOiJwYW90b3IiLCJleHAiOiJlMjI2MjI4ODR9.CtcP2Gsjj3NVGuQw6kWCY3PsrSLUR2u9mDqKhRbnvkShmPVrcmIkDoRvSyTCCwpQrftNAiMZA9JW28hqTPERRA",
  "etatcourantdemande": "Debut",
  "montantcreditdemande": 25,
  "datenaissance": "10-09-1987",
  "dureecredit": 6,
  "idDemande": "dee66939-d205-4b47-bdcb-9047ff9d948e",
  "_links": {
    "self": {
      "href": "http://localhost:8082/demandes/dee66939-d205-4b47-bdcb-9047ff9d948e"
    },
    "Historique actions": {
      "href": "http://localhost:8082/demandes/dee66939-d205-4b47-bdcb-9047ff9d948e/actions/57"
    },
    "Collection": {
      "href": "http://localhost:8082/demandes/?status",
      "templated": true
    }
  }
}

```

Figure 12 : Détails de la demande après ajout de l'action « Debut »

Si l'on continue le flow suivant cette logique :

Debut – Etude – Decision – Acceptation

L'état de la demande est à « Fin » une fois qu'une décision a été prise ici « Acceptation ».

```

{
  "nom": "Durand",
  "prenom": "Paul",
  "adresse": "10 rue des Tulipes Nancy",
  "revenus": 25,
  "token": "eyJhbGciOiJIUzUxMiJ9.eyJzdWIiOiJwYW90b3IiLCJleHAiOiJlMjI2MjI4ODR9.CtcP2Gsjj3NVGuQw6kWCY3PsrSLUR2u9mDqKhRbnvkShmPVrcmIkDoRvSyTCCwpQrftNAiMZA9JW28hqTPERRA",
  "etatcourantdemande": "Fin",
  "montantcreditdemande": 25,
  "datenaissance": "10-09-1987",
  "dureecredit": 6,
  "idDemande": "dee66939-d205-4b47-bdcb-9047ff9d948e",
  "_links": {
    "self": {
      "href": "http://localhost:8082/demandes/dee66939-d205-4b47-bdcb-9047ff9d948e"
    },
    "Historique actions": [
      {
        "href": "http://localhost:8082/demandes/dee66939-d205-4b47-bdcb-9047ff9d948e/actions/57"
      },
      {
        "href": "http://localhost:8082/demandes/dee66939-d205-4b47-bdcb-9047ff9d948e/actions/58"
      },
      {
        "href": "http://localhost:8082/demandes/dee66939-d205-4b47-bdcb-9047ff9d948e/actions/59"
      },
      {
        "href": "http://localhost:8082/demandes/dee66939-d205-4b47-bdcb-9047ff9d948e/actions/60"
      }
    ],
    "Collection": {
      "href": "http://localhost:8082/demandes/?status",
      "templated": true
    }
  }
}

```

Figure 13 : Etat final une fois toutes les actions requises ont été prises.

Si l'on vérifie les détails des actions menées on voit bien que toutes celles avant la dernière ont pour statut « Terminée ». Exemple pour l'étape de « Decision » avant « Acceptation ».

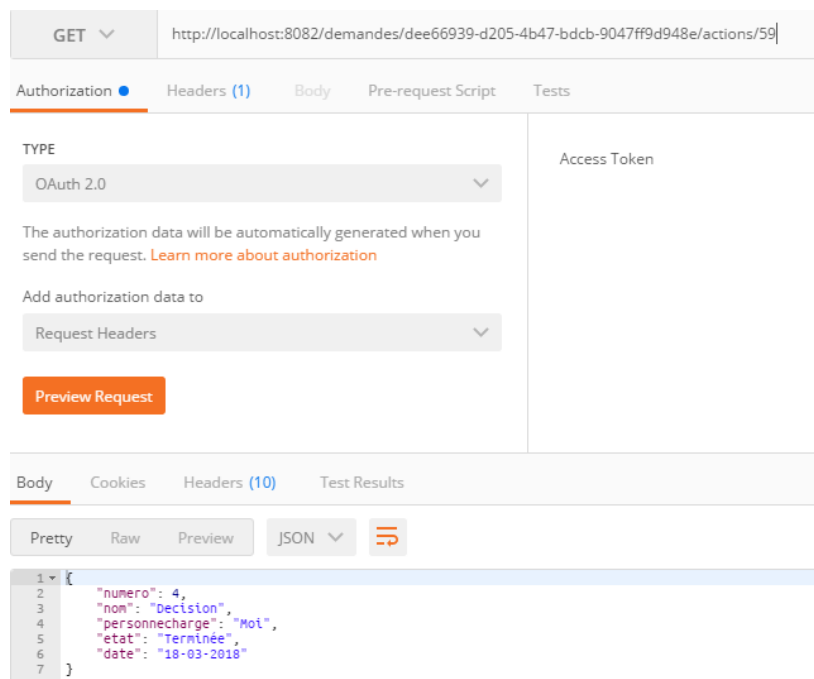


Figure 14 : Vérification que l'état est à Terminée

Clôturer une demande

Un utilisateur interne peut à tout moment cloturer une demande en effectuant une requête DELETE **http://localhost:8082/demandes/1** avec un body vide.

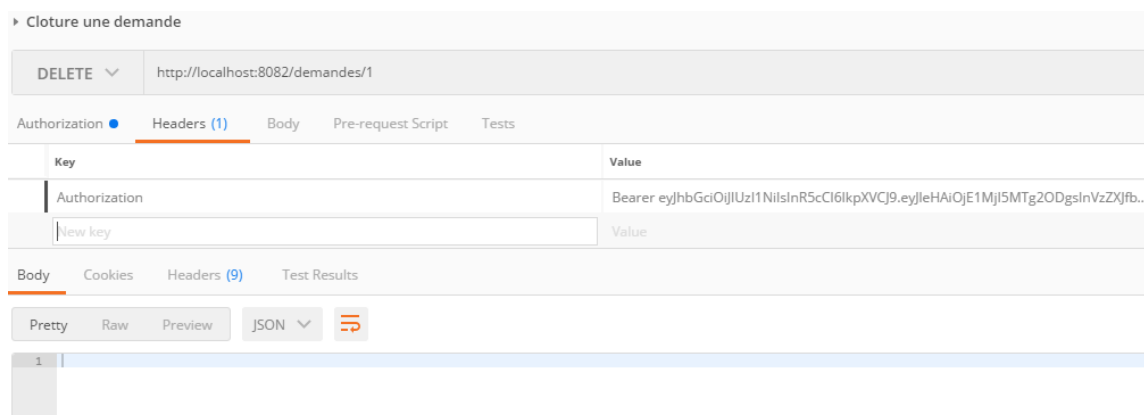


Figure 15 : Suppression d'une demande

Si l'on vérifie le détail de la demande :

```

{
  "nom": "Dupont",
  "prenom": "Jean-Michel",
  "adresse": "Paris",
  "revenus": 25,
  "token": "eyJhbGciOiJIUzUxMiJ9.eyJzdWIiOiJwYW50b3IiLCJleHAiOiJlMjI1MjQyNTF9.CXZwOTJVCUQl3FwpxCjkjrest_WhkZB5n3sCOUqTASwU7NMUB0K02aJpgFE0Zk4CwAb_sR2Z9abpw3b8pXxZng",
  "etatcourantdemande": "Fin",
  "montantcreditdemande": 25,
  "datenaissance": "10-09-1992",
  "dureecredit": 6,
  "idDemande": "1",
  "links": {
    "self": {
      "href": "http://localhost:8082/demandes/1"
    },
    "Historique actions": [
      {
        "href": "http://localhost:8082/demandes/1/actions/55"
      },
      {
        "href": "http://localhost:8082/demandes/1/actions/56"
      }
    ],
    "collection": {
      "href": "http://localhost:8082/demandes(?status)",
      "templated": true
    }
  }
}

```

Figure 16 : Vérification du statut « Fin » de la demande après clôture

Modification d'une action

Il est possible de modifier la personne en charge d'une action. Pour cela on fait une requête PUT **<http://localhost:8082/demandes/1/actions/56>**.



Figure 17 : Modification d'une action

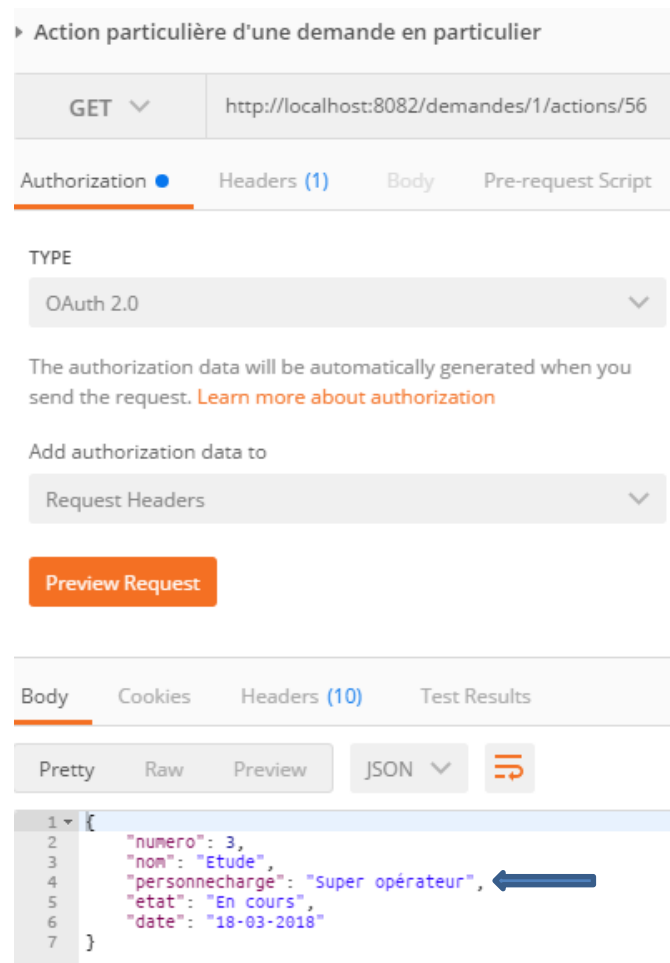


Figure 18 : Vérification de la modification de la personne en charge

Conclusion

Ce projet m'a permis d'appréhender des notions inhérentes au paradigme des API telles que les liens HATEOAS, la gestion des codes de retour ou encore la gestion de la sécurité via OAuth. Plus largement même si certaines fonctionnalités sont absentes de cet API comme le load-balancing ou encore le caching, cela m'a permis d'acquérir de nouvelles connaissances que je pourrai mettre en œuvre dans mon milieu professionnel.