VOLLET Mathilde M1

PRÉSENTATION DU PROJET
MESDEVIS

SOMMAIRE

Note : Les diagrammes UML et scripts SQL sont présents dans ce fichier mais également dans le dossier du projet.

I) Rapide présentation du projet

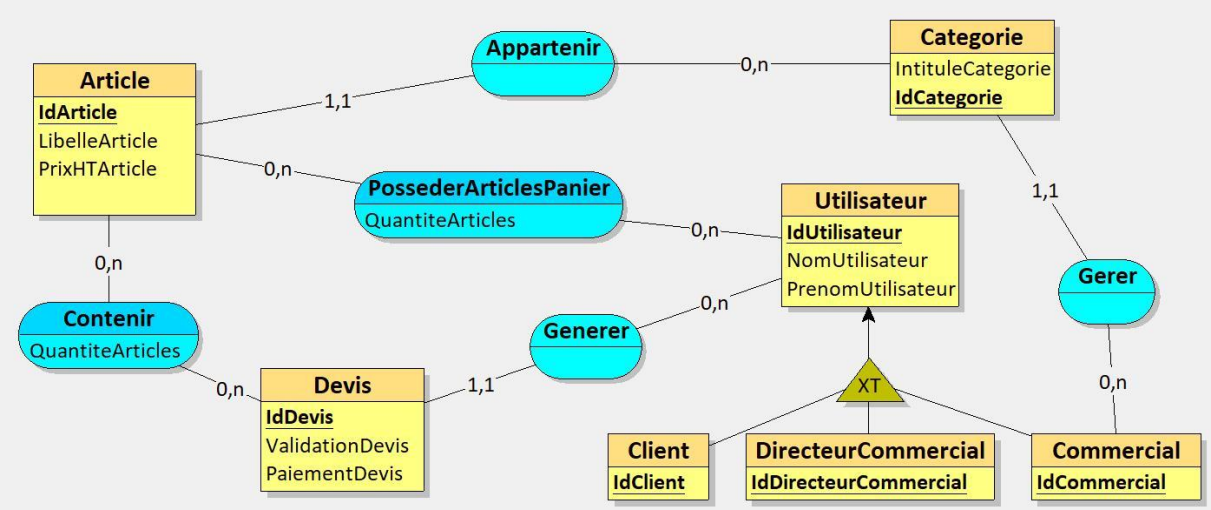
Le projet MesDevis est un projet commandé par le client JeVendsTOUS. Il s'agit d'une API REST permettant de réaliser des devis. Ces devis peuvent être fait soit par des clients, soit par des commerciaux ou soit par le directeur général. Pour générer un devis, il faut que l'utilisateur ajoute des articles à son panier et qu'il le valide. Les articles proviennent d'un catalogue de 1000 articles. Le devis ainsi généré doit être validé, en fonction des stocks des articles, soit par les commerciaux, soit par le directeur général. Le client pourra alors régler le devis.

II) Conception et modélisation du projet

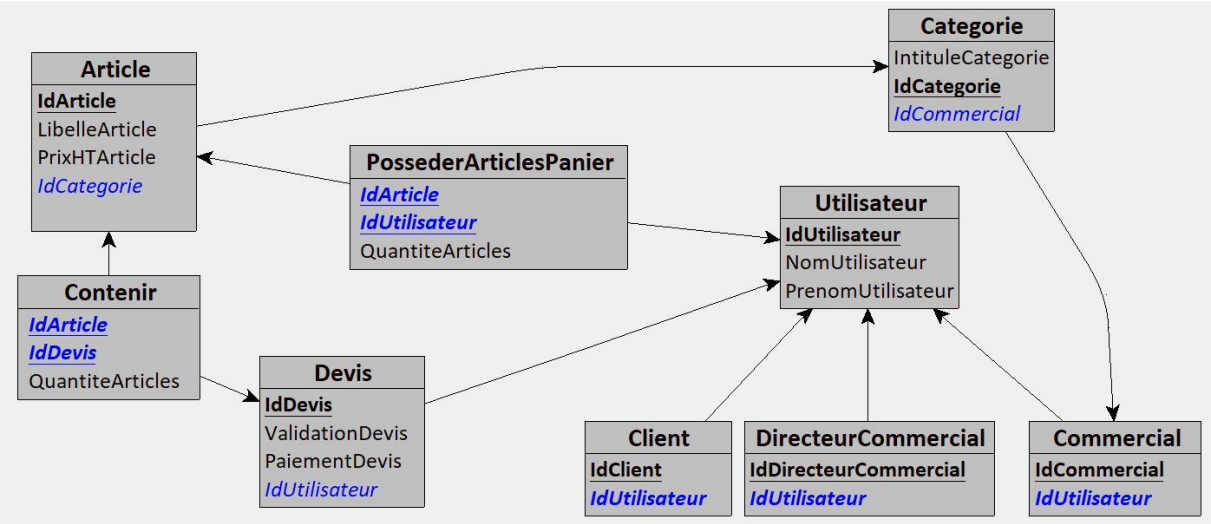
Pour conceptualiser et modifier le projet avant de le concevoir, j'ai modélisé un MCD, un MLD et différents diagrammes UML.

1) MCD – MLD

Afin de visualiser la base de données, j'ai réalisé un modèle conceptuel de données.

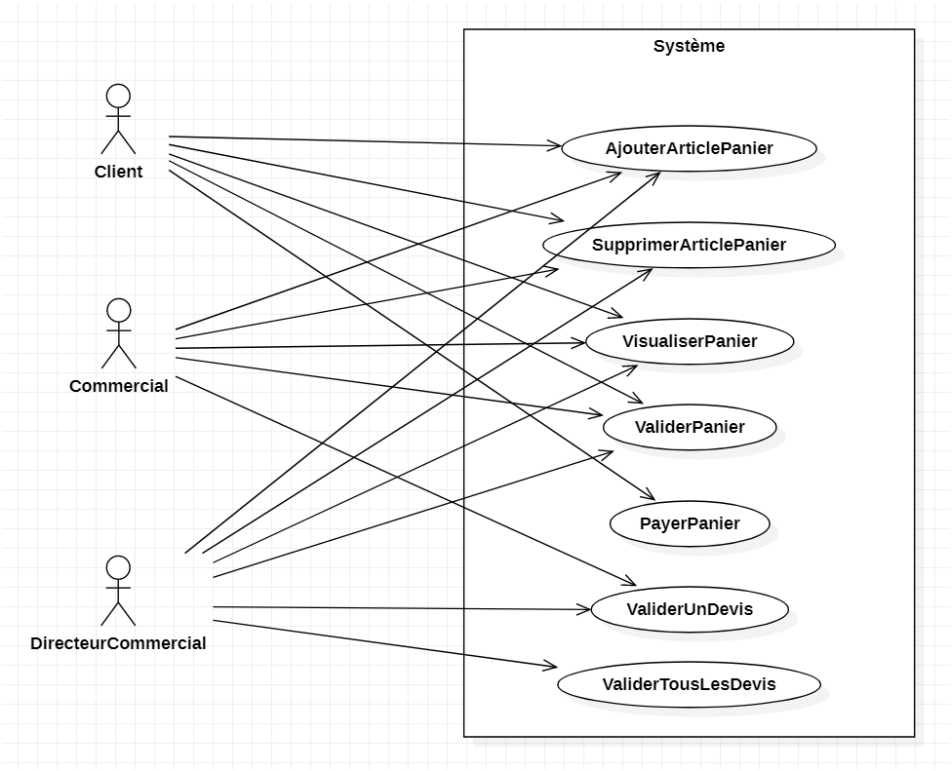


J'ai ensuite transformé ce diagramme en modèle logique de données.



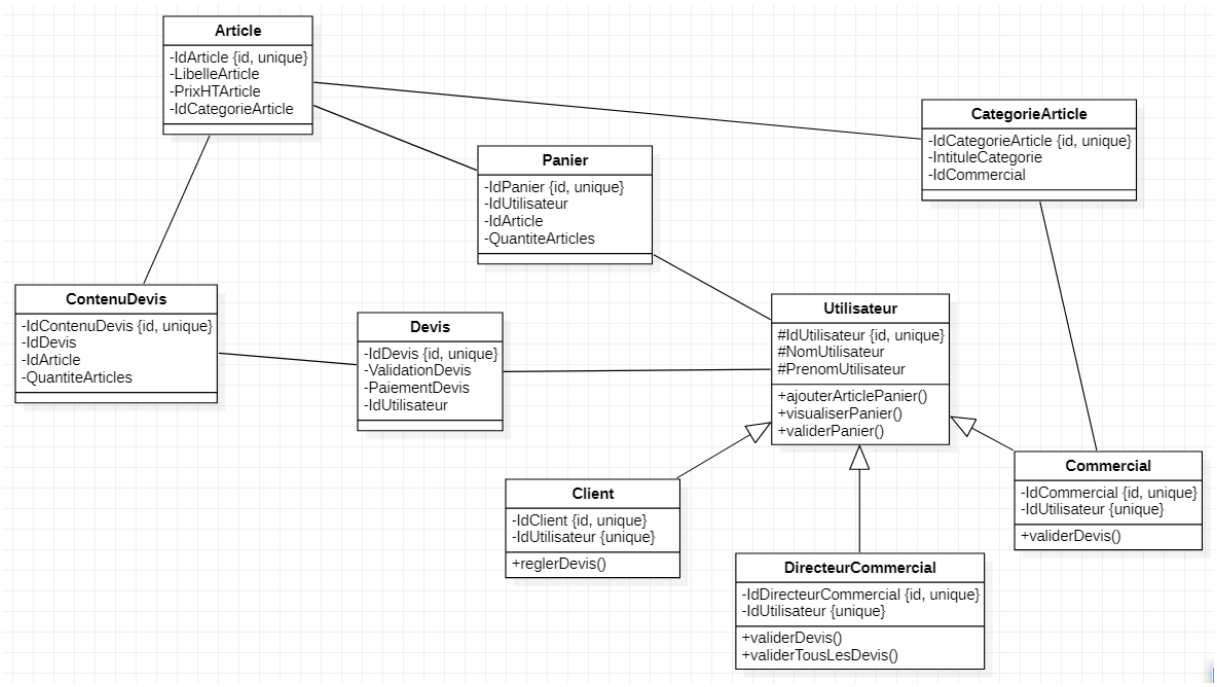
2) Diagramme de cas d'usage

Puis, j'ai identifié les différents cas d'utilisation à partir des acteurs (clients, commerciaux et directeur général).



3) Diagramme de classes

A partir des modèles conceptuels et logiques de données, j'ai conçu le diagramme de classes.



4) Diagramme de séquence

Voici le diagramme de séquence représentant le cas spécifique d'un client qui génère un devis. Ce devis est ensuite validé par un commercial. Le client peut alors régler le devis.

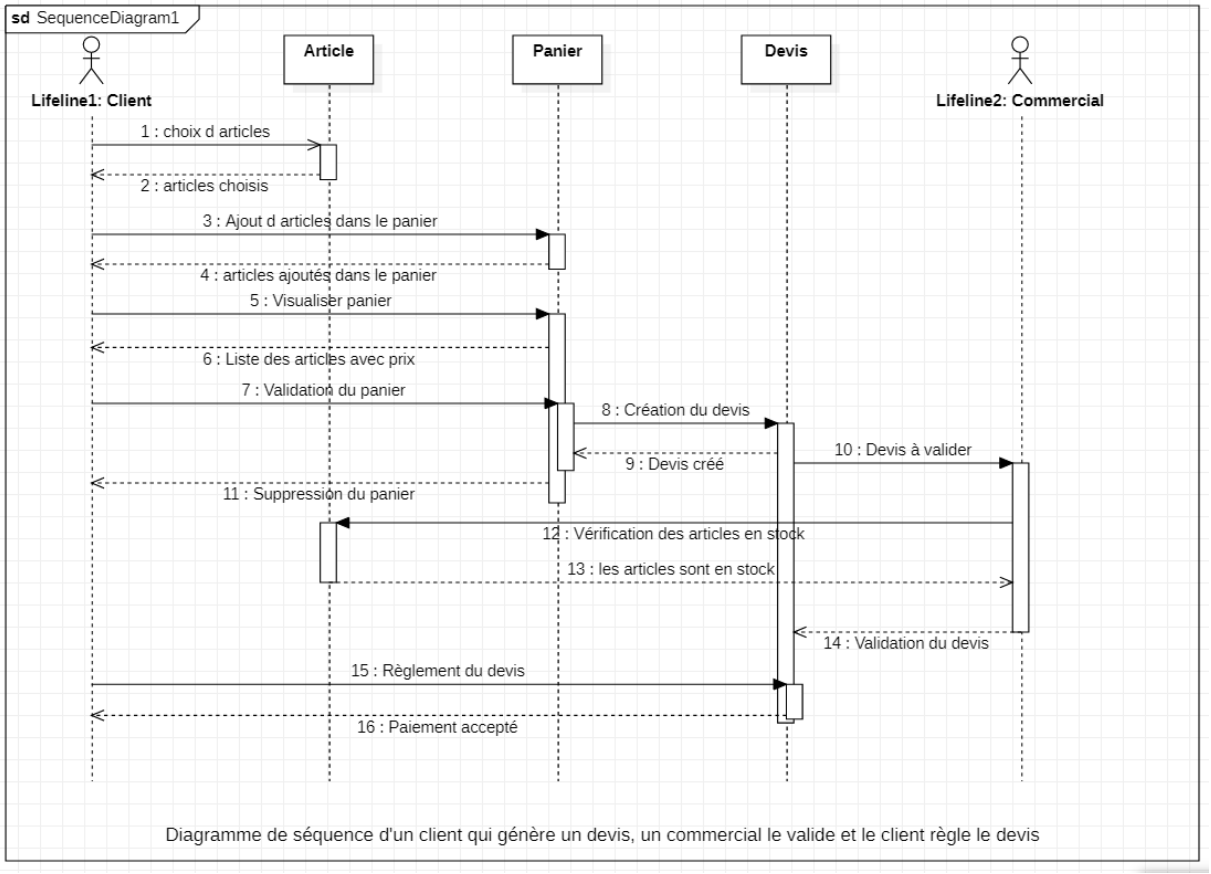
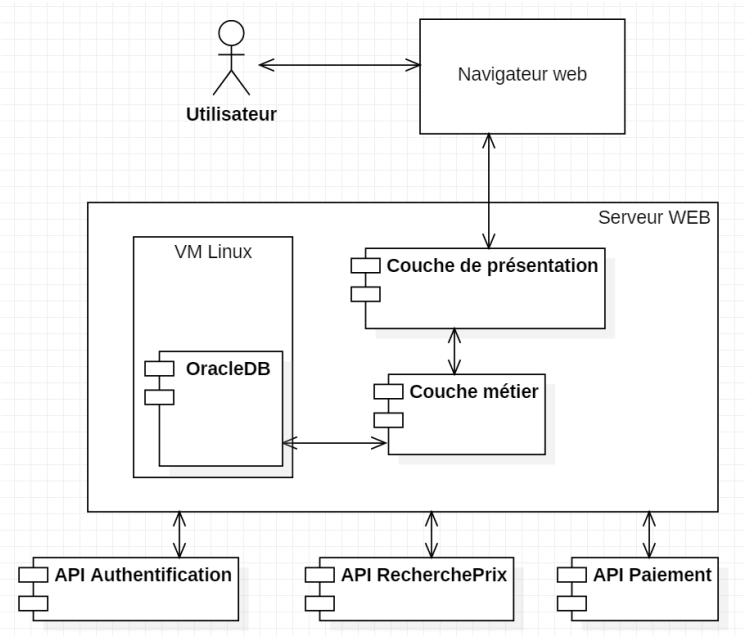


Diagramme de séquence d'un client qui génère un devis, un commercial le valide et le client règle le devis

5) Diagramme de déploiement

Ci-dessous le diagramme de déploiement avec les différentes API (internes et externes).



III) Réalisation du projet

En m'appuyant sur les différents diagrammes réalisés précédemment, j'ai codé le projet.

Lors de la conception des diagrammes présentés précédemment, on peut constater qu'un héritage est présent entre Utilisateur, Client, Commercial et DirecteurCommercial. Cet héritage permet d'utiliser des méthodes aux communes aux trois différents types d'utilisateurs. Il permet aussi de pouvoir différencier les rôles : par exemple, un client ne peut pas valider un devis. J'ai également fait le choix de différencier un panier et un devis. En effet, si un utilisateur ajoute des articles dans son panier, il pourra retrouver ses articles même après s'être déconnecté puis reconnecté.

1) La base de données

Tout d'abord, j'ai commencé par écrire le script permettant d'initialiser la base de données. Ensuite, afin d'initialiser des données dans la base, j'ai créé des utilisateurs (donts des commerciaux) et des catégories d'articles. Enfin, pour compléter la base de données, j'ai écrit une procédure stockée permettant d'initialiser les 1000 articles du catalogue. Il ne me restait plus qu'à appeler cette procédure pour que la base de données soit prête.

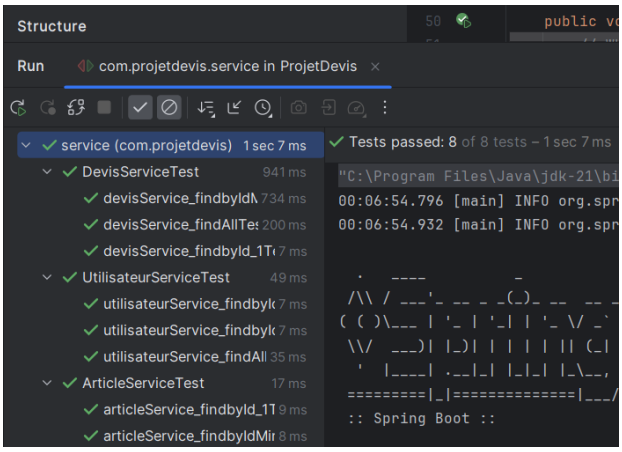
Remarque : Tous les scripts SQL sont dans le dossier « Scripts_SQL » mais aussi en annexes.

2) Le code

Pour écrire le code de cette API, j'ai utilisé le framework SpringBoot. J'ai organisé l'architecture du projet en quatre packages principaux. Il y a, tout d'abord le package des entités. Dans celui-ci on retrouve toutes les entités présentes dans le diagramme de classes. Les DTO de ces entités sont également présentes. Puis, il y a le package Repository dans lequel on retrouve des interfaces permettant d'implémenter le CRUD pour chacune des entités. Le troisième package est celui des services. On y retrouve les objets Mapper permettant de convertir une entité en DTO et une DTO en entité. Les interfaces Service se situent aussi dans ce package. Elles permettent d'implémenter plusieurs requêtes. Enfin, toujours dans ce même package, il y a des classes qui implémentent ces interfaces Service afin de définir concrètement ce que fait chaque requête dans la base de données. Le dernier package correspond aux Controllers. Les contrôleurs sont appelés directement pour effectuer des requêtes http. Ils réalisent alors un traitement spécifique utilisant ou non les fonctionnalités des CRUD. Il y a, en plus, deux packages simulant les apis internes et externes.

3) Les tests

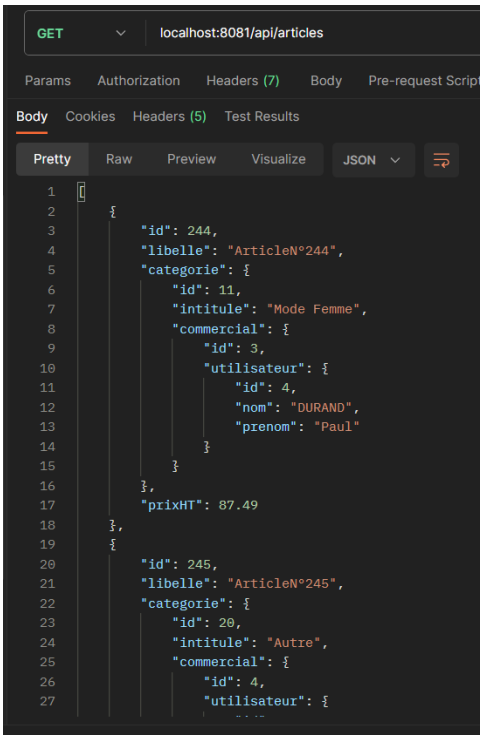
Afin de m'assurer que mon code fonctionne, j'ai mis en place quelques tests unitaires. Je n'ai pas pu mettre en place la totalité des tests unitaires par manque de temps. Cependant, j'ai testé personnellement toutes les fonctionnalités pour m'assurer qu'elles se déroulent sans erreur. Les quelques tests unitaires fonctionnent également.



IV) Exécution du projet

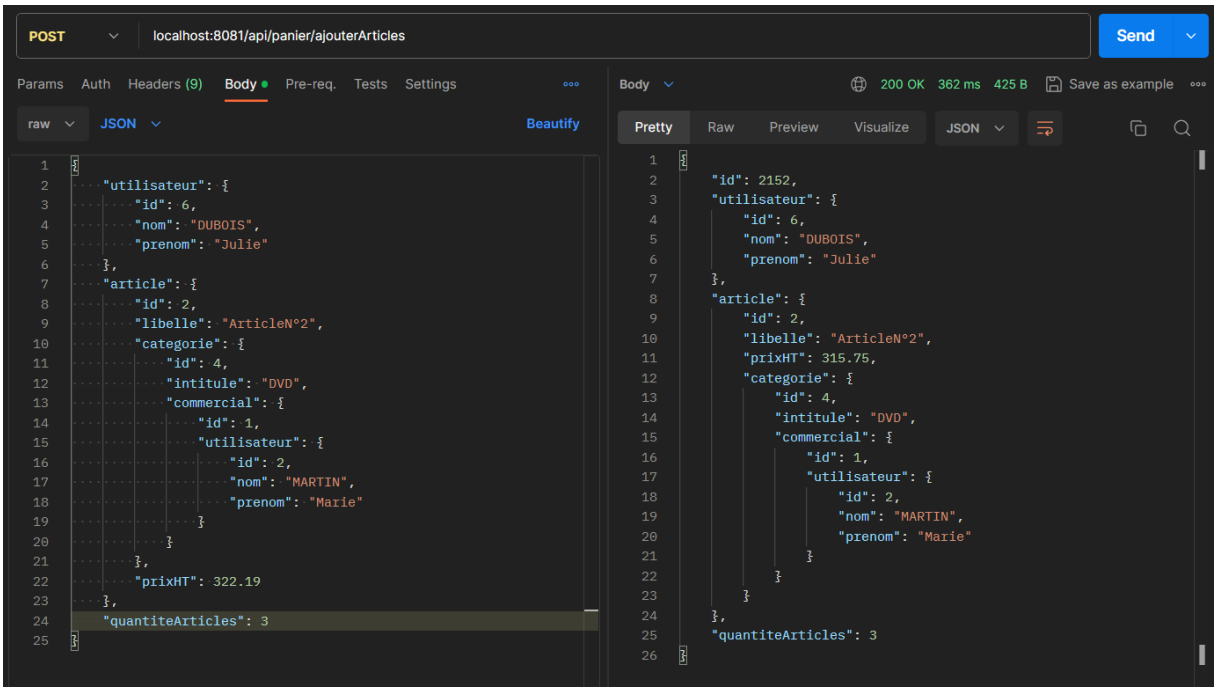
Voici quelques exemples de requêtes effectuées avec Postman.

- Récupération de tous les articles du catalogue



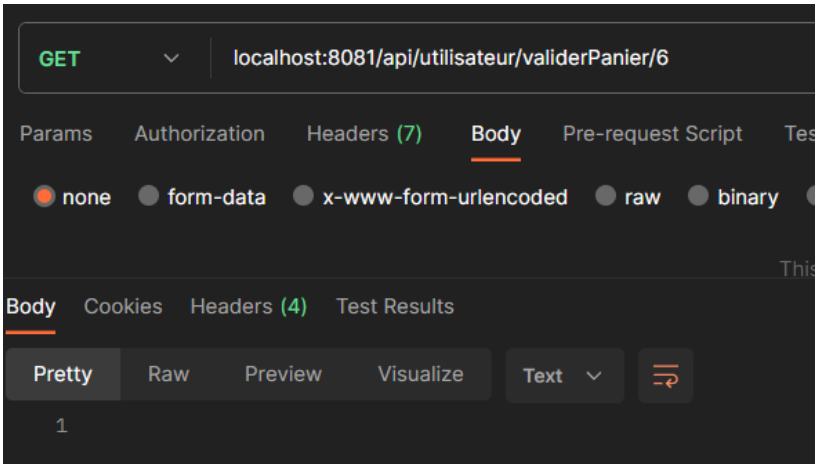
- Ajout d'articles dans le panier

Dans cette requête, 3 articles sont ajoutés au panier de l'utilisatrice. Lors de l'ajout dans le panier, on fait appel à une API externe afin de rechercher s'il existe un meilleur prix pour l'article. Ici c'est le cas puisque le prix passe de 322,19€ à 315,75€ pour cet article.



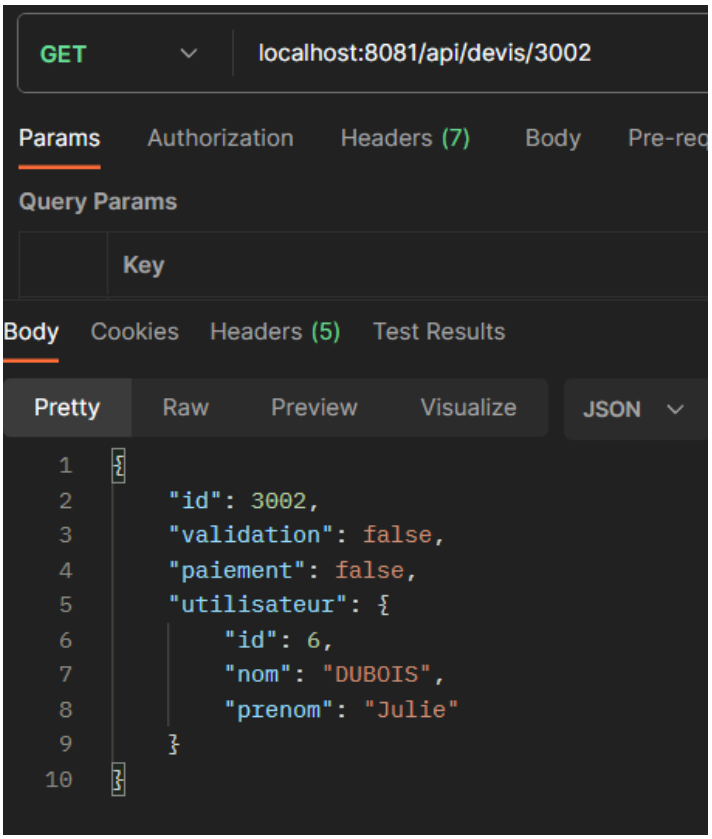
- Validation du panier par l'utilisateur

Lorsque l'utilisateur a terminé d'ajouter des articles à son panier, il peut le valider. On peut remarquer que cette requête ne retourne rien. Cependant, elle a fonctionné puisque le panier s'est bien transformé en devis comme on pourra le voir par la suite.



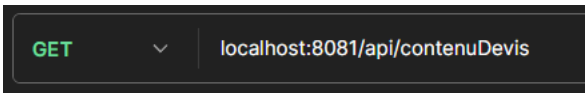
- Récupération du devis par son id

Le devis a bien été créé en base de données. Il n'est, pour l'instant, ni validé ni payé.



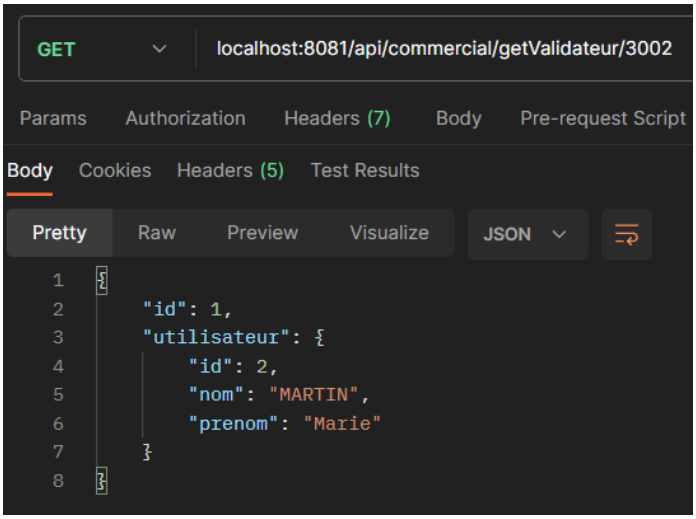
- Récupération du contenu du devis

Il est aussi possible de récupérer le contenu du devis.



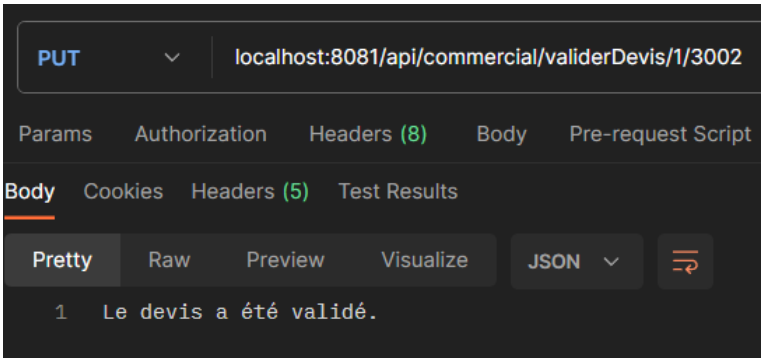
- Recherche du commercial le plus apte à valider le devis

Il existe une requête permettant de déterminer quel commercial peut valider un devis ou non. Cette méthode se base sur le poids des articles. Il s'agit de la somme la plus élevée des prix des articles par catégories dont le commercial s'occupe qui est prise en compte.



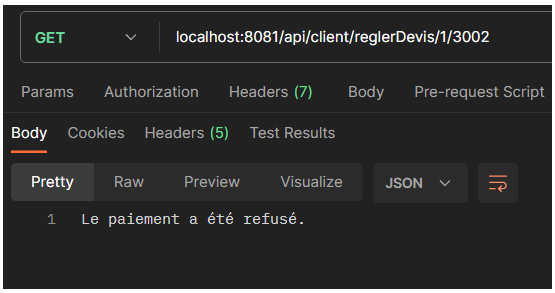
- Validation du devis par un commercial

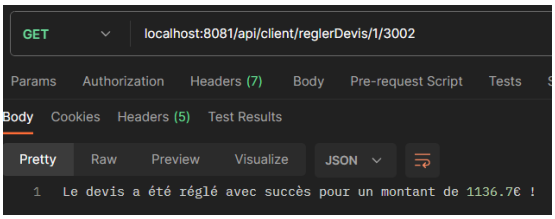
Après s'être assuré des stocks, le commercial peut valider le devis si c'est bien lui le « validateur » du devis et si ce devis est inférieur à 10 000€.



- Paiement du devis par le client

Le client peut ensuite procéder au paiement. Son paiement peut parfois échouer. Les deux cas sont présentés ici. C'est une API simulant la banque qui détermine si le paiement est accepté ou non. Le montant total que paye le client se calcule en faisant la somme des articles multipliés par leurs quantités. On multiplie ensuite ce montant par 1.2 pour ajouter la TVA.





V) Annexes

- Script SQL pour la création de la base de données :

```
CREATE TABLE utilisateur(
   id_utilisateur INTEGER NOT NULL,
   nom_utilisateur VARCHAR2(30 char) NOT NULL,
   prenom_utilisateur VARCHAR2(30 char) NOT NULL,
   PRIMARY KEY (id_utilisateur)
 );

CREATE TABLE client(
   id_client INTEGER NOT NULL,
   id_utilisateur INTEGER NOT NULL,
   PRIMARY KEY (id_client),
   FOREIGN KEY(id_utilisateur) REFERENCES utilisateur(id_utilisateur),
   UNIQUE (id_utilisateur)
 );

CREATE TABLE directeur_commercial(
   id_directeur_commercial INTEGER NOT NULL,
   id_utilisateur INTEGER NOT NULL,
   PRIMARY KEY (id_directeur_commercial),
   FOREIGN KEY (id_utilisateur) REFERENCES utilisateur(id_utilisateur),
   UNIQUE (id_utilisateur)
 );

CREATE TABLE commercial(
   id_commercial INTEGER NOT NULL,
   id_utilisateur INTEGER NOT NULL,
   PRIMARY KEY(id_commercial),
   FOREIGN KEY (id_utilisateur) REFERENCES utilisateur(id_utilisateur),
   UNIQUE (id_utilisateur)
 );

CREATE TABLE categorie_article(
   id_categorie_article INTEGER NOT NULL,
   intitule_categorie VARCHAR2(50 char) NOT NULL,
   id_commercial INTEGER NOT NULL,
   PRIMARY KEY (id_categorie_article),
   FOREIGN KEY (id_commercial) REFERENCES commercial(id_commercial)
 );

CREATE TABLE article(
   id_article INTEGER NOT NULL,
   libelle_article VARCHAR2(50 char) NOT NULL,
   prix_ht_article FLOAT NOT NULL,
   id_categorie_article INTEGER NOT NULL,
   PRIMARY KEY (id_article),
   FOREIGN KEY (id_categorie_article) REFERENCES categorie_article(id_categorie_article)
 );

CREATE TABLE devis(
   id_devis INTEGER NOT NULL,
   validation_devis NUMBER(1) NOT NULL,
   paiement_devis NUMBER(1) NOT NULL,
   id_utilisateur INTEGER NOT NULL,
   PRIMARY KEY (id_devis),
   FOREIGN KEY (id_utilisateur) REFERENCES utilisateur(id_utilisateur)
 );
```

```
CREATE TABLE contenu_devis(
   id_contenu_devis INTEGER NOT NULL,
   id_devis INTEGER NOT NULL,
   id_article INTEGER NOT NULL,
   quantite_articles INTEGER NOT NULL,
   PRIMARY KEY (id_contenu_devis),
   FOREIGN KEY (id_devis) REFERENCES devis(id_devis),
   FOREIGN KEY (id_article) REFERENCES article(id_article),
   UNIQUE (id_devis, id_article)
);

CREATE TABLE Panier(
   id_panier INTEGER NOT NULL,
   id_utilisateur INTEGER NOT NULL,
   id_article INTEGER NOT NULL,
   quantite_articles INTEGER NOT NULL,
   PRIMARY KEY (id_panier),
   FOREIGN KEY (id_utilisateur) REFERENCES Utilisateur(id_utilisateur),
   FOREIGN KEY (id_article) REFERENCES Article(id_article),
   UNIQUE (id_utilisateur, id_article)
);
```

- Script SQL pour ajouter des utilisateurs et des catégories d'articles :

```
INSERT INTO UTILISATEUR VALUES(1, 'LAMBERT', 'Pierre');
INSERT INTO UTILISATEUR VALUES(2, 'MARTIN', 'Marie');
INSERT INTO UTILISATEUR VALUES(3, 'BERNARD', 'Isabelle');
INSERT INTO UTILISATEUR VALUES(4, 'DURAND', 'Paul');
INSERT INTO UTILISATEUR VALUES(5, 'PETIT', 'Louis');
INSERT INTO UTILISATEUR VALUES(6, 'DUBOIS', 'Julie');

INSERT INTO DIRECTEUR_COMMERCIAL VALUES(1, 1);
INSERT INTO COMMERCIAL VALUES(1,2);
INSERT INTO COMMERCIAL VALUES(2,3);
INSERT INTO COMMERCIAL VALUES(3,4);
INSERT INTO COMMERCIAL VALUES(4,5);
INSERT INTO CLIENT VALUES(1, 6);

INSERT INTO CATEGORIE_ARTICLE VALUES(1, 'Musique', 1);
INSERT INTO CATEGORIE_ARTICLE VALUES(2, 'Jeux vidéos', 1);
INSERT INTO CATEGORIE_ARTICLE VALUES(3, 'Jeux de socété', 1);
INSERT INTO CATEGORIE_ARTICLE VALUES(4, 'DVD', 1);
INSERT INTO CATEGORIE_ARTICLE VALUES(5, 'Informatique', 1);
INSERT INTO CATEGORIE_ARTICLE VALUES(6, 'Livre', 2);
INSERT INTO CATEGORIE_ARTICLE VALUES(7, 'Décoration', 2);
INSERT INTO CATEGORIE_ARTICLE VALUES(8, 'Ameublement', 2);
INSERT INTO CATEGORIE_ARTICLE VALUES(9, 'Bricolage', 2);
INSERT INTO CATEGORIE_ARTICLE VALUES(10, 'Sport', 2);
INSERT INTO CATEGORIE_ARTICLE VALUES(11, 'Mode Femme', 3);
INSERT INTO CATEGORIE_ARTICLE VALUES(12, 'Mode Homme', 3);
INSERT INTO CATEGORIE_ARTICLE VALUES(13, 'Mode Enfant', 3);
INSERT INTO CATEGORIE_ARTICLE VALUES(14, 'Bébé', 3);
INSERT INTO CATEGORIE_ARTICLE VALUES(15, 'Maison', 3);
INSERT INTO CATEGORIE_ARTICLE VALUES(16, 'Nourriture', 4);
INSERT INTO CATEGORIE_ARTICLE VALUES(17, 'Animaux', 4);
INSERT INTO CATEGORIE_ARTICLE VALUES(18, 'Hygiène', 4);
INSERT INTO CATEGORIE_ARTICLE VALUES(19, 'Entretien', 4);
INSERT INTO CATEGORIE_ARTICLE VALUES(20, 'Autre', 4);
```

- Script SQL de la procédure stockée :

```
CREATE OR REPLACE PROCEDURE InitialiserCatalogue IS
   idCategorie INTEGER;
   prix DECIMAL(10,2);

BEGIN
   -- Boucle FOR de 1 à 1000
   FOR i IN 1..1000 LOOP
         idCategorie := TRUNC(DBMS_RANDOM.VALUE(1, 21));
         prix := DBMS_RANDOM.VALUE(0.00, 400.00);
         INSERT INTO ARTICLE VALUES(i, 'ArticleN°' || TO_CHAR(i), prix, idCategorie);
      END LOOP;

   COMMIT;
END InitialiserCatalogue;
```