Projet de session

Historique

#	Date	Description
1	25 septembre	Version initiale
2	25 septembre	Précisions sur les remises
3	25 septembre	Correction au diagramme de séquence
4	26 septembre	Instructions de remise sur Github
5	7 octobre	Ajout d'une exigence lors du paiement d'une commande et ajout de précisions sur la récupération des produits

Informations générale

L'objectif du projet de session est de développer et déployer une application Web responsable du paiement de commandes Internet.

Le projet est divisé en deux étapes pour chacune des remises.

Objectifs

- Se familiariser avec le développement Web
- Développer une API REST
- Utiliser des services Web distants
- S'assurer de la résilience et de la performance d'une application Web
- Déployer une application Web

Équipe

Le travail se fait individuellement ou en équipe de 2.

Échéances

Date	Description	Pondération
4 novembre 21h	Première remise	15%
16 décembre 21h	Remise finale	15%

Le projet de session représente 30% de la note globale du cours.

Évaluation

L'évaluation se fera à distance sans la présence du ou des membres de l'équipe.

Le code de l'application doit être hébergé sur Github.

Aux deux dates de la remise, à 21h, le dépôt sera cloné et c'est ce qui sera utilisé pour l'évaluation.

Remise

La création du projet sur Github est automatisé grâce à Github Classroom.

Pour créer un projet, vous devez suivre les instruction suivantes :

1. Créer un compte Github si ce n'est déjà pas le cas.
2. Rendez-vous sur la page du projet de session :

   https://classroom.github.com/g/oH9-4nj2
3. Acceptez le travail, et identifier vous dans la liste en choisissant votre code permanent **Assurez-vous de ne pas prendre le code de quelqu'un d'autre!**
4. Créer une équipe individuelle, ou rejoignez l'équipe de votre collègue si vous faites la remise à 2.
5. Votre dépôt Github privé sera créé, vous pouvez commencer à l'utiliser.

Langage de programmation

Le langage de programmation pour le projet de session est Python. La version minimale qui sera utilisé est 3.6. Vous pouvez utiliser la version 3.7 si vous le désirez.

Le cadriciel de développement Web est Flask 1.1.

Le démonstrateur de laboratoire sera en mesure de vous apporter du support et du soutien technique pour ces deux technologies.

Le projet

Le projet consiste à développer une application Web responsable de prendre des commandes Internet. Cette application devra répondre à une API REST, mais devra également être utilisée à travers des pages HTMLs.

Le projet est séparé en deux. Les informations pour la première remise sont disponibles ci-bas. La section pour la deuxième remise sera disponible plus tard.

Il s'agit de remises incrémentiels sur le même projet. Pour la deuxième remise, vous continuerez à utiliser le projet décrit ci-dessous.

Première remise

La première partie se concentre sur le développement et l'utilisation d'une API REST.

L'application web doit répondre aux requêtes Web suivantes :

```
GET /
Content-Type: application/json

200 OK
```

```
{
   "products" : [
      {
         "name" : "Hapiness Bottle",
         "id" : 123,
         "in_stock" : true,
         "description" : "An incredible bottle filled with
         ↪  happiness and joy. One of a kind!",
         "price" : 5999,
         "image": "http://caissy.dev/shops/images/image.png"
      },
      {
         "description" : "A limited edition special Flying
         ↪  Squirrel. Hurry up, it won't last long!",
         "image": "http://caissy.dev/shops/images/image.png",
         "in_stock" : false,
         "id" : 456,
         "name" : "Flying Squirrel Limited Edition",
         "price" : 3149
      }
   ]
}
```

La page d'index doit afficher une liste des produits disponibles pour passer une commande. Ces produits sont récupérés selon les spécifications de la section **Récupération des produits**.

```
POST /order
Content-Type: application/json
```

```
{ "product": { "id": 123, "quantity": 2 } }
```

```
302 Found
Location: /order/<int:order_id>
```

La création d'une nouvelle commande se fait avec un appel `POST` à `/order`. Si la commande est créé, le code HTTP de retour doit être `302` et inclure le lien vers la commande nouvellement créée.

Exigences

- Pour cette première remise, une commande ne peut recevoir qu'un seul produit.
- L'objet `product` est obligatoire et doit consister d'un identifiant (`id`) et d'une quantité.

S'il manque l'objet `product`, ou que l'objet `product` ne contient pas le champ `id` ou `quantity`, un message d'erreur doit être retourné.

```
422 Unprocessable Entity
```

```
{
   "errors" : {
       "product": {
           "code": "missing-fields",
           "name": "La création d'une commande nécessite un
           ↪  produit"
        }
   }
}
```

Si l'item représenté par l'identifiant unique `product_id` est en inventaire (`in_stock == True`), un processus d'achat commence. Si le produit n'est pas en inventaire, l'API doit retourner une erreur avec le code HTTP 422 et le message d'erreur suivant :

```
422 Unprocessable Entity
```

```
{
   "errors" : {
       "product": {
           "code": "out-of-inventory",
           "name": "Le produit demandé n'est pas en inventaire"
        }
   }
}
```

Une fois le processus d'achat initialisé, on peut récupérer la commande complète à tout moment avec cette requête GET.

```
GET /order/<int:order_id>
Content-Type: application/json

200 OK
```

```
{
   "order" : {
      "id" : 6543,
      "total_price" : 9148,
      "email" : null,
      "credit_card": {},
      "shipping_information" : {},
      "paid": false,
      "transaction": {},
      "product" : {
         "id" : 123,
         "quantity" : 1
      },
      "shipping_price" : 1000
   }
}
```

Exigences

1. Le champ `total_price` correspond au prix du produit multiplié par la quantité. Il n'inclus pas le prix d' exépédition (`shipping_price`)
2. Le champ `shipping_price` représente le prix total pour expédier la commande. Ce champ doit être calculé automatiquement en fonction du poids total des articles composant la commande :
   - Jusqu'à 500 grammes : 5$
   - De 500 grammes à 2kg : 10$
   - À partir de 2kg (2kg et plus) : 25$

Par défaut, une commande ne contient aucune information sur le client. On doit fournir le courriel et l'adresse d'expédition du client.

```
PUT /order/<int:order_id>
Content-Type: application/json
```

```
{
   "order" : {
      "email" : "caissy.jean-philippe@uqam.ca",
      "shipping_information" : {
         "country" : "Canada",
         "address" : "201, rue Président-Kennedy",
         "postal_code" : "H2X 3Y7",
         "city" : "Montréal",
         "province" : "QC"
      }
   }
}
```

```
200 OK
Content-Type: application/json
```

```
{
   "order" : {
      "shipping_information" : {
         "country" : "Canada",
         "address" : "201, rue Président-Kennedy",
         "postal_code" : "H2X 3Y7",
         "city" : "Montréal",
         "province" : "QC"
      },
      "credit_card" : {},
      "email" : "caissy.jean-philippe@uqam.ca",
      "total_price" : 9148,
      "transaction": {},
      "paid": false,
      "product" : {
         "id" : 123,
         "quantity" : 1
      },
      "shipping_price" : 1000,
      "id" : 6543
   }
}
```

Exigences :

1. Une commande qui n'existe pas doit retourner un code d'erreur 404.
2. Les champs `emails`, `shipping_information` et les informations de l'objet `shipping_information` (`country`, `address`, `postal_code`, `city`, `province`)
3. Tous les auters champs ne peuvent pas être changés par cet API (`total_price`, `transaction`, `paid`, `product`, `shipping_price`, `id`).

S'il manque un champ, l'API doit retourner une erreur en conséquence.

```
PUT /order/<int:order_id>
Content-Type: application/json
```

```
{
   "order" : {
      "shipping_information" : {
         "country" : "Canada",
         "province" : "QC"
      },
   }
}
```

```
422 Unprocessable Entity
Content-Type: application/json
```

```
{
   "errors" : {
       "order": {
           "code": "missing-fields",
           "name": "Il manque un ou plusieurs champs qui sont
           ↪  obligatoire",
       }
   }
}
```

Lorsque la commande contient toutes les informations, on peut la payer avec une carte de crédit. Cette fonction doit transférer les informations de carte de crédit vers le service de paiement distant disponible à `https://caissy.dev/shops/pay`.

L'API REST de paiement distant est décrite dans la section **service de paiement distant**.

```
PUT /order/<int:order_id>
Content-Type: application/json
```

```
{
   "credit_card" : {
      "name" : "John Doe",
      "number" : "4242 4242 4242 4242",
      "expiration_year" : 2024,
      "cvv" : "123",
      "expiration_month" : 9
   }
}
```

```
200 OK
Content-Type: application/json
```

```
{
   "order" : {
      "shipping_information" : {
         "country" : "Canada",
         "address" : "201, rue Président-Kennedy",
         "postal_code" : "H2X 3Y7",
         "city" : "Montréal",
         "province" : "QC"
      },
      "email" : "caissy.jean-philippe@uqam.ca",
      "total_price" : 9148,
      "paid": true, # <- une commande complète est considérée
      ↪  comme payée
      "product" : {
         "id" : 123,
         "quantity" : 1
      },
      "credit_card" : {
         "name" : "John Doe",
         "first_digits" : "4242",
         "last_digits": "4242",
         "expiration_year" : 2024,
         "expiration_month" : 9
      },
```

```
      "transaction": {
          "id": "wgEQ4zAUdYqpr21rt8A10dDrKbfcLmqi",
          "success": true,
          "amount_charged": 10148
      },
      "shipping_price" : 1000,
      "id" : 6543
   }
}
```

Exigences

1. On ne peut pas fournir `credit_card` avec `shipping_information` et/ou `email`. Ces deux appels doivent être fait de manière distincts.
2. Si on applique `credit_card` à une commande qui n'a pas de courriel et/ou les informations d'expédition, elle doit retourner un message d'erreur.
3. Les informations de transaction et de carte de crédit retournés par le service distant doivent être persistés et appliqué à la commande.
4. Si le service de paiement distant retourne une erreur, celle-ci doit être retourné au client
5. On ne peut pas payer une commande deux fois. Si une commande a déjà eu un paiement, un message d'erreur doit être retourné.

```
PUT /order/<int:order_id>
Content-Type: application/json
```

```
{
   "credit_card" : {
      "name" : "John Doe",
      "number" : "4242 4242 4242 4242",
      "expiration_year" : 2024,
      "cvv" : "123",
      "expiration_month" : 9
   }
}
```

```
422 Unprocessable Entity
Content-Type: application/json
```

```
{
   "errors" : {
       "order": {
           "code": "missing-fields",
           "name": "Les informations du client sont nécessaire
           ↪  avant d'appliquer une carte de crédit"
       }
```

```
   }
}
```

Exemple d'une commande qui a déjà été payée.

```
PUT /order/<int:order_id>
Content-Type: application/json
```

```
{
   "credit_card" : {
      "name" : "John Doe",
      "number" : "4242 4242 4242 4242",
      "expiration_year" : 2024,
      "cvv" : "123",
      "expiration_month" : 9
   }
}
```

```
422 Unprocessable Entity
Content-Type: application/json
```

```
{
   "errors" : {
       "order": {
           "code": "already-paid",
           "name": "La commande a déjà été payée."
       }
   }
}
```

Exemple d'une carte de crédit refusé par le service distant :

```
PUT /order/<int:order_id>
Content-Type: application/json
```

```
{
   "credit_card" : {
      "name" : "John Doe",
      "number" : "4000 0000 0000 0002",
      "expiration_year" : 2024,
      "cvv" : "123",
      "expiration_month" : 9
   }
}
```

```
422 Unprocessable Entity
Content-Type: application/json
```

```
{
    "credit_card": {
        "code": "card-declined",
        "name": "La carte de crédit a été déclinée."
    }
}
```

Service de paiement distant

Lorsque l'API `/order/<int:order_id>` est appelé avec une carte de crédit, il doit communiquer avec l'API distant de paiement. Cette API est disponible à l'adresse `https://caissy.dev/shops/pay`.

```
POST /shops/pay
Host: caissy.dev
Content-Type: application/json
```

```
{
   "credit_card" : {
      "name" : "John Doe",
      "number" : "4242 4242 4242 4242",
      "expiration_year" : 2024,
      "cvv" : "123",
      "expiration_month" : 9
   },
   "amount_charged": 10148 # <- montant total incluant les frais
   ↪  d'expédition
}
```

```
200 OK
```

```
{
   "credit_card" : {
      "name" : "John Doe",
      "first_digits" : "4242",
      "last_digits": "4242",
      "expiration_year" : 2024,
      "expiration_month" : 9
   },
   "transaction": {
       "id": "wgEQ4zAUdYqpr21rt8A10dDrKbfcLmqi",
       "success": true,
       "amount_charged": 10148
   }
}
```

Les informations de la transaction doivent être stockées sur la commande.

En cas d'erreur, l'API distant va retourner une erreur de type :

```
POST /shops/pay
Host: caissy.dev
Content-Type: application/json
```

```
{
   "credit_card" : {
      "name" : "John Doe",
      "number" : "4000 0000 0000 0002",
      "expiration_year" : 2024,
      "cvv" : "123",
      "expiration_month" : 9
   },
   "amount_charged": 10148 # <- montant total incluant les frais
   ↪  d'expédition
}
```

```
422 Unprocessable Entity
```

```
{
   "errors" : {
       "credit_card": {
           "code": "card-declined",
           "name": "La carte de crédit a été déclinée"
        }
   }
}
```

Seuls les 2 cartes de crédits de tests vont être acceptés :

- `4242 4242 4242 4242` : carte de crédit valide
- `4000 0000 0000 0002` : carte de crédit déclinée

Tout autre numéro de carte retourne le code `incorrect-number`.

Les champs `expiration_year` et `expiration_month` doivent être des entiers représentant une date d'expiration valide. L'API va retourner une erreur si la carte est expirée.

Le champ `cvv` doit obligatoirement être un string contenant 3 chiffres.

Lorsqu'une commande est payée, l'API doit retourner toutes les informations de la commande :

```
GET /order/<int:order_id>
Content-Type: application/json

200 OK
```


```
{
   "order" : {
      "shipping_information" : {
         "country" : "Canada",
         "address" : "201, rue Président-Kennedy",
         "postal_code" : "H2X 3Y7",
         "city" : "Montréal",
         "province" : "QC"
      },
      "email" : "caissy.jean-philippe@uqam.ca",
      "total_price" : 9148,
      "paid": true,
      "product" : {
         "id" : 123,
         "quantity" : 1
      },
      "credit_card" : {
         "name" : "John Doe",
         "first_digits" : "4242",
         "last_digits": "4242",
         "expiration_year" : 2024,
         "expiration_month" : 9
      },
      "transaction": {
          "id": "wgEQ4zAUdYqpr21rt8A10dDrKbfcLmqi",
          "success": true,
          "amount_charged": 10148
      },
      "shipping_price" : 1000,
      "id" : 6543
   }
}
```

Récupération des produits

Lors du lancement de l'application, celle-ci va devoir se connecter à ce service, et récupérer la liste complète des produits et des informations applicables. Les informations des produits devront être persistées localement, c'est-à-dire que l'application Web ne doit pas récupérer les informations à chaque requête. Seulement une fois par lancement, i.e. : lorsque la commande `FLASK_DEBUG=True FLASK_APP=inf5190 flask run` est exécuté.

Les produits sont accessibles à l'adresse suivante : `https://caissy.dev/shops/products`

La réponse sera une liste de produits répondant aux caractéristiques suivantes :

```
GET /shops/products
Content-Type: application/json
```

```
{
   "products" : [
      {
         "name" : "Hapiness Bottle",
         "id" : 123,
         "in_stock" : true,
         "description" : "An incredible bottle filled with
         ↪  happiness and joy. One of a kind!",
         "price" : 5999,
         "weight": 500,
         "image": "http://caissy.dev/shops/images/image.png"
      },
      {
         "description" : "A limited edition special Flying
         ↪  Squirrel. Hurry up, it won't last long!",
         "image": "http://caissy.dev/shops/images/image.png",
         "in_stock" : false,
         "weight": 6400,
         "id" : 456,
         "name" : "Flying Squirrel Limited Edition",
         "price" : 3149
      }
   ]
}
```

Nom du champ	Définition
`id`	Un identifiant numérique unique représentant la ressource du produit
`name`	Le nom du produit
`description`	La déscription du produit
`price`	Le prix du produit en cents
`in_stock`	Si le produit est en stock ou pas

Nom du champ	Définition
`image`	Une image représentant le produit

Exigences techniques

1. **Un fichier nommé `CODES-PERMANENTS` doit être à la racine de votre projet et contenir le ou les codes permanents séparé par un saut de ligne**
    - Donc pour un travail fait individuellement, le fichier doit simplement contenir votre code permanent
    - Pour un équipe de deux, le fichier doit contenir les code permanent des deux étudiants, un par ligne.
2. Votre dépôt Github doit avoir été créé avec Github Classroom (les instructions sont dans la section **Remise**
3. Le projet devra rouler sous Python 3.6+ et Flask 1.11+
4. Seul les paquets `flask`, `pytest`, `pytest-flask`, et `peewee` sont permis. Vous avez droit d'utiliser tous les modules de la librairie standard de Python
5. La base de données utilisée sera un fichier local `sqlite3`
6. Vous devez utiliser l'ORM `peewee`
7. Toutes les données doivent être stockés dans la base de donnée
8. La base de données doit être initialisée avec

    ```
    FLASK_DEBUG=True FLASK_APP=inf5190 flask init-db
    ```

9. À partir de la racine de votre projet, l'application doit pouvoir rouler avec la commande suivante :

    ```
    FLASK_DEBUG=True FLASK_APP=inf5190 flask run
    ```

Critères d'évaluations

- 20% : Respect des **exigences techniques**
- 30% : Fonctionnalités de l'application Web
- 10% : Tests (unitaire, fonctionnel et d'intégration) : couverture et qualité des tests
- 20% : Qualité du code
- 20% : Respect des exigences pour chacun des API

N.B. : Toutes les *exigences techniques* mentionnées ci-haut doivent être respectés, sinon c'est 0/20.

Deuxième remise

À venir.

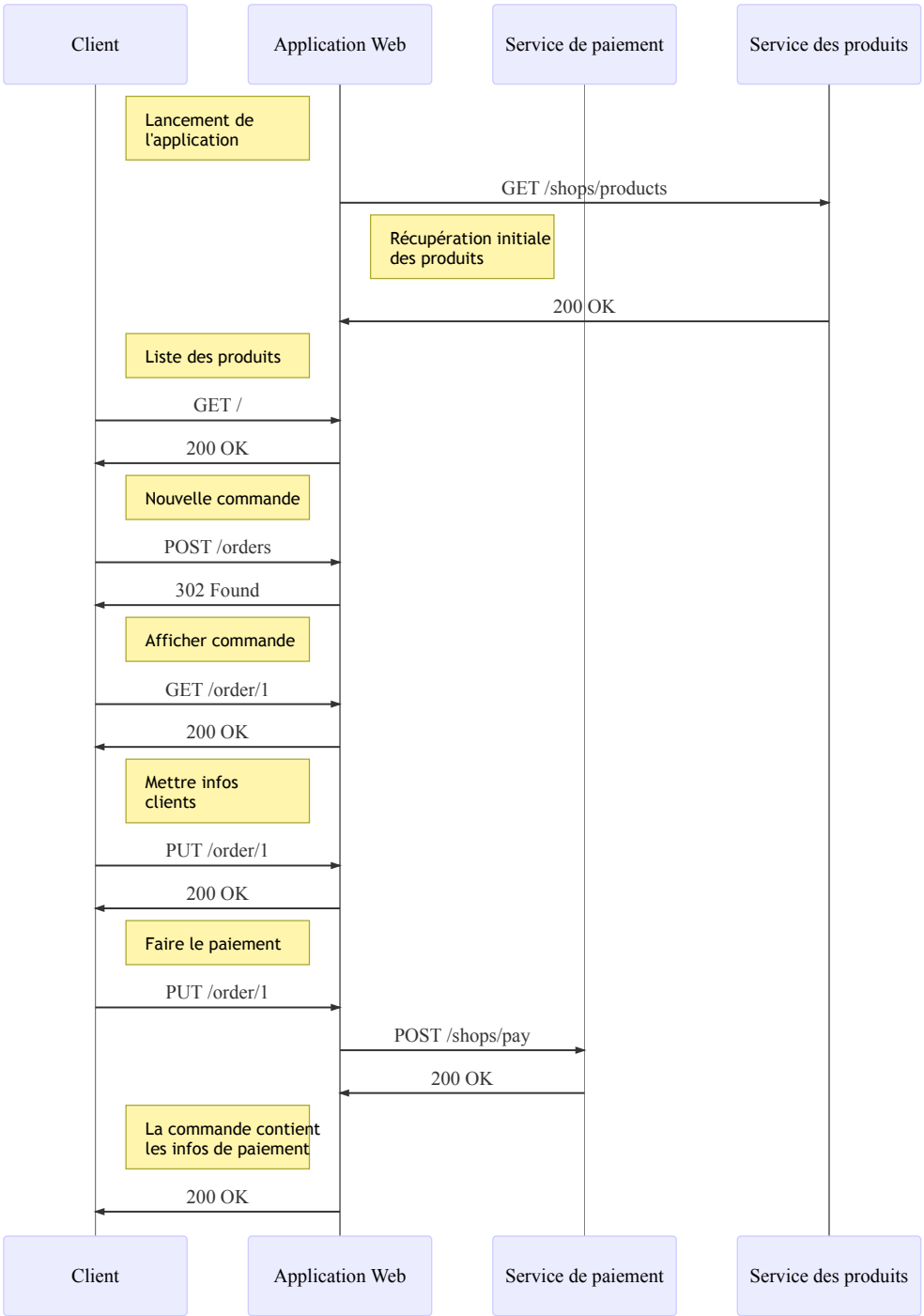

Figure 1: Diagramme de séquence