ÉCOLE DE TECHNOLOGIE SUPÉRIEURE
Université du Québec
Le génie pour l'industrie

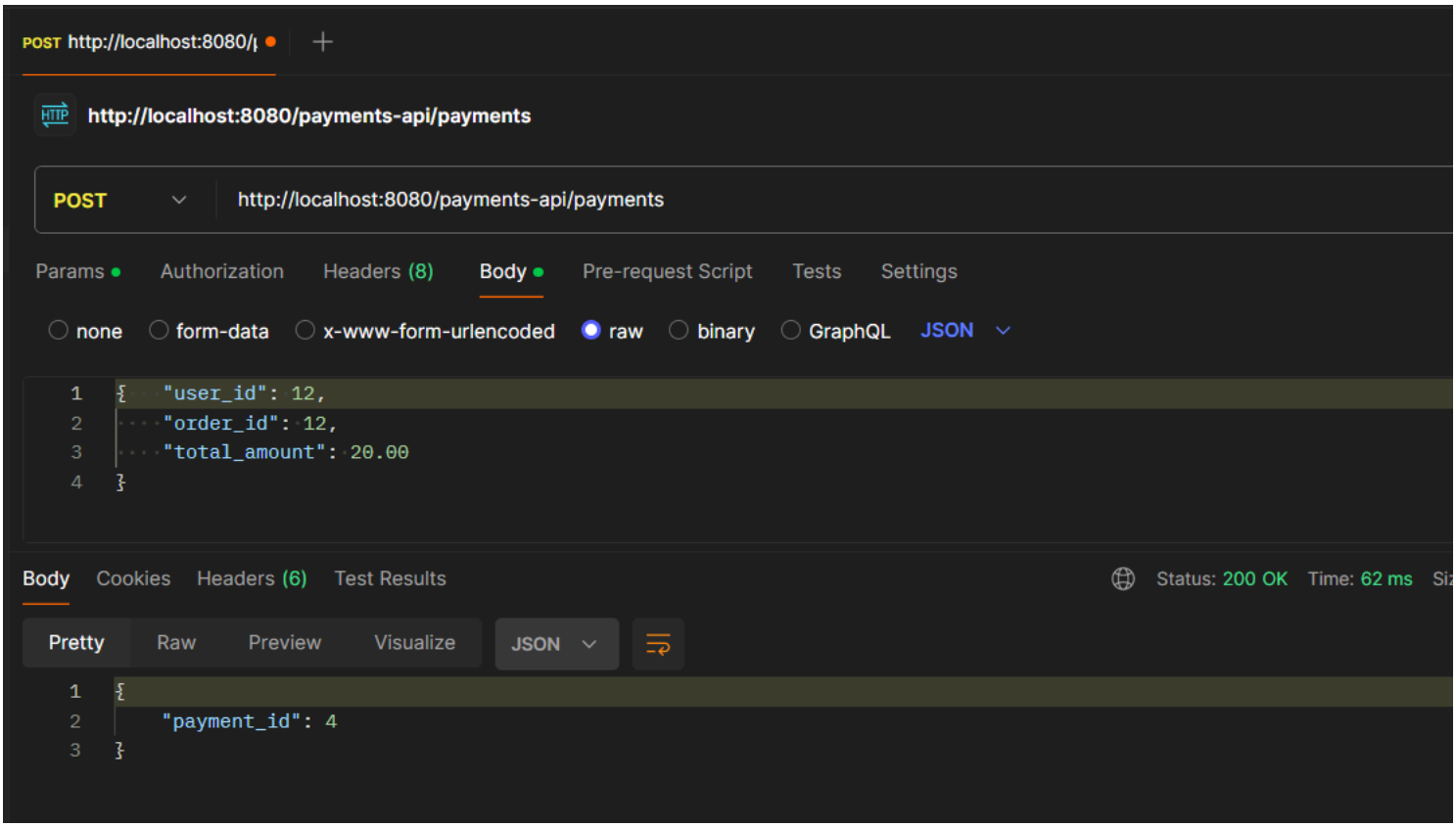
William Lavoie
Rapport de laboratoire
LOG430 — Architecture logicielle
28 Octobre 2025 École de technologie supérieure

Questions

Question 1

Quelle réponse obtenons-nous à la requête à POST /payments ? Illustrez votre réponse avec des captures d'écran/terminal.

En envoyant la requête à `POST http://localhost:8080/payments-api/payments` par Postman, ce qui correspond à la route utilisé par `store-manager`, à l'exception que `localhost` est remplacé par le nom du conteneur dans le réseau Docker, on obtient la réponse suivante. Le service de paiement retourne l'id du paiement qui a été exécuté, avec un code 200 pour signifier que l'opération s'est bien passé.

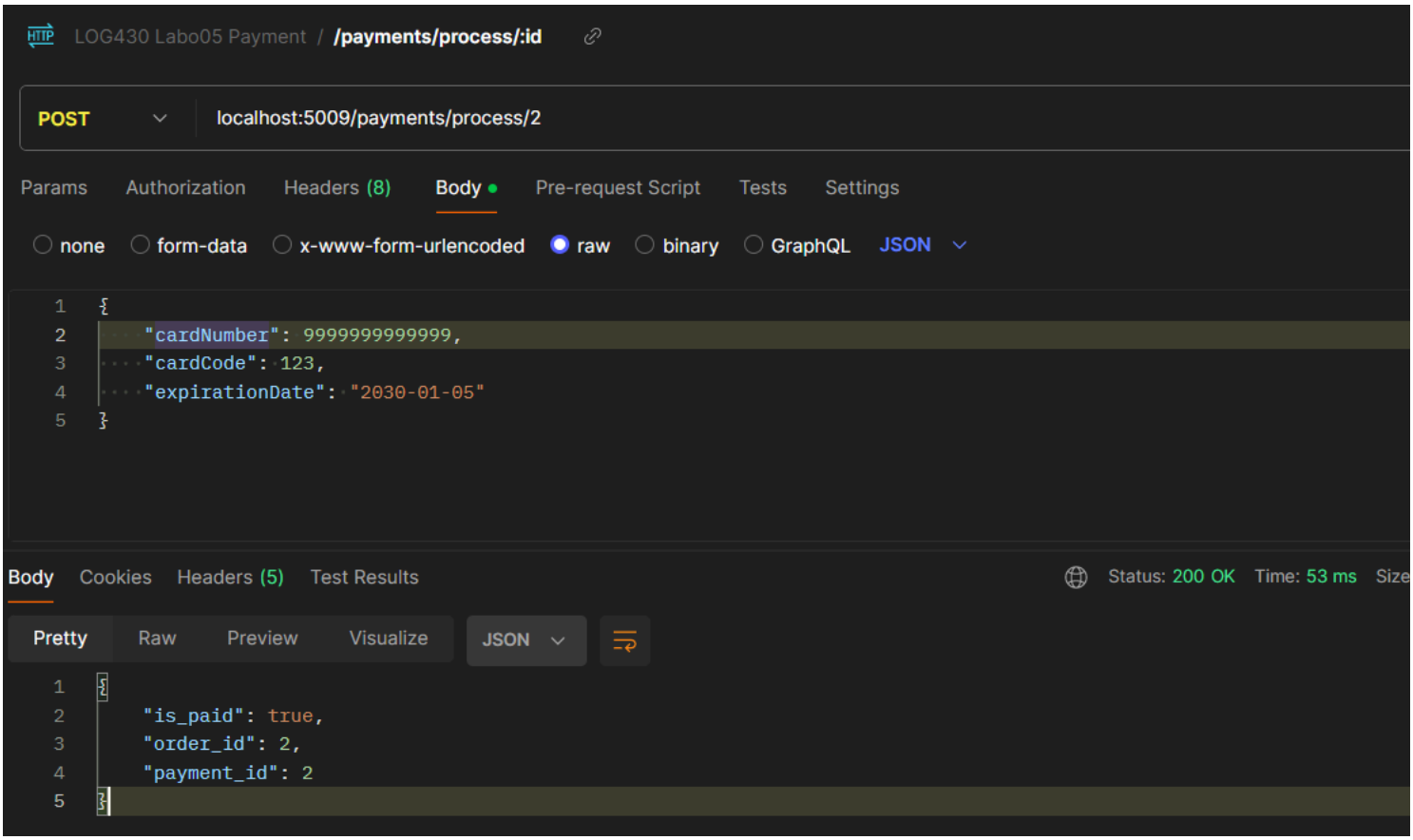


Question 2

Quel type d'information envoyons-nous dans la requête à POST payments/process/:id ? Est-ce que ce serait le même format si on communiquait avec un service SOA, par exemple ? Illustrez votre réponse avec des exemples et captures d'écran/terminal.

On envoie le *cardnumber* et le *cardCode* qui correspondent probablement aux identifiants de carte de crédits de l'utilisateurs. C'est nécessaire pusique dans une architecture microservices chaque service a une base de données séparées, donc si une requête fait référence à un élément d'une base de données, dans ce cas ci un usager, il faut envoyer dans les requêtes vers les autres services un identifiant qui sert en quelque sorte de clé étrangère, bien que la table a laquelle elle fait référence,existe pas dans les autres services.

Dans le cas d'un SOA ce ne serait pas nécessaire car les données sont partagées.



Question 3

Quel résultat obtenons-nous de la requête à POST payments/process/:id?

Comme on peut le voir dans l'image ci-dessus, on reçoit `is_paid`, un booléen qui indique si le paiement a bien fonctionné, `payment_id` qui est l'identifiant qui a été passé dans l'url et finalement `order_id`, qui est la commande rattaché au paiement en question.

Question 4

Quelle méthode avez-vous dû modifier dans log430-a25-labo05-payment et qu'avez-vous modifié ? Justifiez avec un extrait de code.

J'ai modifié la méthode `update_status_to_paid` car c'est elle qui s'occupe de faire passer l'état des paiements à `is_valid = True`, donc il est logique que ce soit cette méthode qui appelle `store_manager` afin de transmettre cette information. J'ai ajouté la partie suivante à la méthode:

```python
        # Update the payment status
        payment.is_paid = True
        session.commit()

        order_payment  = {
            "order_id": payment.order_id,
            "is_paid": payment.is_paid,
        }

        response_from_store_manager = requests.put('http://api-gateway:8080//store-api/orders',
            json=order_payment,
            headers={'Content-Type': 'application/json'}
        )

        if response_from_store_manager.ok:
            print(f"Payment of order {payment.order_id} was processed sucessfully.")

        return {
            "payment_id": payment_id,
            "order_id": payment.order_id,
            "is_paid": True
        }
```

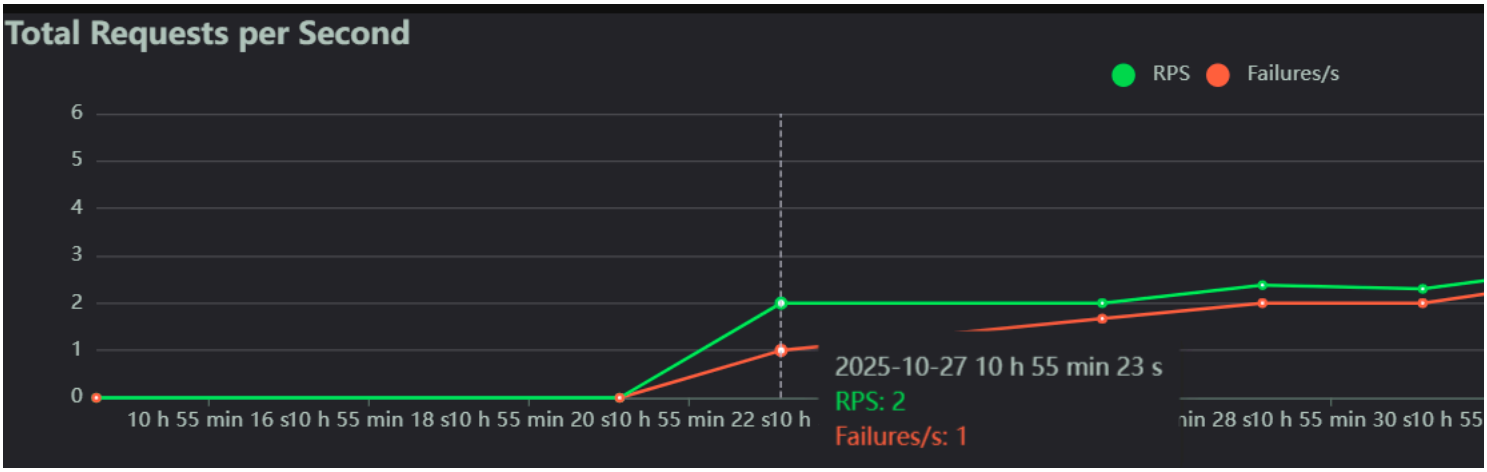
Dans les logs de `store_manager`, on voit les lignes suivantes qui démontre que les commandes sont mises à jour correctement.

```
172.29.0.1 - - [27/Oct/2025 14:29:11] "POST /orders HTTP/1.1" 201 -
172.29.0.1 - - [27/Oct/2025 14:29:15] "GET /orders/7 HTTP/1.1" 201 -
172.29.0.8 - - [27/Oct/2025 14:29:15] "GET /metrics HTTP/1.1" 200 -
2025-10-27 14:29:18 - order_controller - DEBUG - Mettre à jour la commande 5, status=True
```

Question 5

À partir de combien de requêtes par minute observez-vous les erreurs 503 ? Justifiez avec des captures d'écran de Locust.

Les erreurs 503 commencent à partir de 2 RPS comme on peut le voir ci-dessous.



Ceci est dû au fait que la configuration du `endpoint` dans krakenD permet 10 requêtes par minute, cependant krakenD converti en requête par seconde, ce qui revient à 1.6 de permise, ainsi, il peut y avoir maximum 10 requêtes par minute, et au plus une par seconde.

```
  "extra_config": {
    "qos/ratelimit/router": {
      "max_rate": 10,
      "every": "1m"
    }
  }
```

Le nombre d'utilisateurs maximum n'a essentiellement pas changé avec l'introduction de Nginx, ce qui était attendu car le *bottleneck* est MySQL, or les 3 instances de `store_manager` utilisent toute la même base de données, ainsi le nombre de requêtes envoyé à MySQL n'a pas changé. Par contre, la latence a augmenté car le traffic est partagé par les instances de `store_manager`. La latence moyenne est maintenant de 5ms.

Question 6

Que se passe-t-il dans le navigateur quand vous faites une requête avec un délai supérieur au timeout configuré (5 secondes) ? Quelle est l'importance du timeout dans une architecture de microservices ? Justifiez votre réponse avec des exemples pratiques.

On obtient une erreur 500 ce qui signifie que le serveur a rencontré une erreur. Cela est causée par le timeout configuré dans KrakenD. Le but de mettre un timeout est d'éviter un "callback hell", soit d'attendre une requête qui ne viendra jamais. Par exemple, dans le cadre d'une architecture microservices il est possible qu'un service ne fonctionne plus, ainsi si les autres services l'appellent ils ne reçevront jamais de réponse, il est donc essentiel de détermine un temps d'attente maximum.

Observations additionnelles

Le script d'intégration/déploiement continu (CI/CD) est similaire à ceux des laboratoires précédents, je n'ai donc pas vraiment rencontré de problème avec ça. J'ai utilisé un `self-hosted runner` sur la VM et le script roule sur Github Actions.

J'ai également écris un script de déploiement pour le service de paiement, cependant je n'ai pas inclus de CI pour celui-ci puisqu'il ne contient pas de tests.

Le seul problème que j'ai rencontré est par rapport à la création du réseau Docker. Dans les autres laboratoires, je supprimais le réseau à chaque fois afin de le recréer, cependant puisque le déploiement de labo05 et du service de paiement séparés, il est impossible de faire ça car l'autre service roule sur le réseau.

Comme solution j'ai ajouté les lignes suivantes:

```
if ! docker network ls --format '{{.Name}}' | grep -w labo05-network; then
  docker network create labo05-network
fi
```

La commande `docker network ls --format '{{.Name}}` affiche la liste des Docker par leur nom, tandis que `grep -w` fait un match exact, ainsi le but de ce bout de code est de créer le réseau Docker seulement s'il n'existe pas déjà.