



Rapport de projet

Systeme de réservation pour les organisateurs de compétitions locales et régionales

Filière : GENIE LOGICIEL

Enseignant :

M.Sédrick Kouagni

Etudiant :

Cédric Assémien

Table des matières

I. Introduction au Projet Python : Mise en Œuvre et Correction des Bogues	3
1. Résolution des Bogues de la Phase 1	3
2. Mise en Œuvre des Éléments de la Phase 2	3
3. Outils Nécessaires	3
II. Description de la démarche	4
a. Objectifs et étapes des tests pour le projet de réservation	4
1. Objectifs	4
2. Étapes des tests	4
III. Plan de test	5-18
A. Description du projet	5
1. Objectifs	5
2. Fonctionnalités clés	5
3. Technologies utilisées	6
B. Objectifs des Tests dans le Projet de Réservation Légère	7
C. Description de la Mission du Projet de Réservation Légère pour les Organisateurs de Compétitions	9
D. Approche pour la Réalisation du Projet de Réservation Légère pour les Organisateurs de Compétitions	10
E. La planification d'un projet est essentielle pour assurer sa réussite	11
F. Les bugs corrigés et fonctionnalités ajoutées	12-18
1. Bugs corrigés	12-16
2. Fonctionnalités ajoutées	16-18
G. Locust	21-23
IV. conclusion	24

I. Introduction au Projet Python : Mise en Œuvre et Correction des Bogues

Cher collègue,

J'ai récemment reçu le rapport de QA pour la phase 1 de notre projet. Malheureusement, il contient plusieurs bogues, dont l'un provoque le plantage de l'application. Je comprends que vous ne serez pas au bureau dans les prochains jours en raison de la maladie de votre enfant ce week-end. Dans cette situation, je suis heureux de prendre en charge la suite du projet.

Voici ce que nous allons faire :

1) Résolution des Bogues de la Phase 1 :

- a) Je vais examiner attentivement les problèmes signalés dans la section "issues" du référentiel (repo).
- b) Ensuite, je reproduirai ces problèmes sur ma machine locale pour les résoudre.
- c) Nous utiliserons Flask et JSON pour éviter d'utiliser une base de données, ce qui nous permettra de gagner du temps de configuration.

2) Mise en Œuvre des Éléments de la Phase 2 :

- a) J'ai ajouté le travail de la phase 2 au référentiel.
- b) Je vais cloner et forker le repo, puis le mettre en place sur ma machine locale en suivant les instructions du fichier README.
- c) En plus de la résolution des bogues, j'ajouterai la gestion des erreurs pour améliorer la robustesse de l'application.

3) Outils Nécessaires :

- a) La plupart des outils requis sont répertoriés dans le fichier requirements.txt du référentiel.
- b) Je vais installer Flask, notre framework de test préféré (pytest), ainsi que notre outil de test de performance (Locust).

II. Description de la démarche

a. Objectifs et étapes des tests pour le projet de réservation :

1) Objectifs :

- a) **Garantir la qualité** de l'application de réservation légère pour les organisateurs de compétitions.
- b) **Vérifier la fonctionnalité** de l'application permettant aux clubs d'inscrire des athlètes.
- c) **Assurer la sécurité** et l'efficacité des événements pour tous les utilisateurs.
- d) **Corriger tous les bugs** qui font planter l'application.
- e) **Ajouter les fonctionnalités** demandées.
- f) **Apporter des améliorations** aux fonctionnalités existantes

2) Étapes des tests :

- a. **Conception des cas de test** : - Créer des scénarios de test pour chaque fonctionnalité. - Couvrir les différents cas d'utilisation, y compris les limites (nombre d'inscriptions, accès des secrétaires, etc.).
- b. **Analyse des exigences** : - Comprendre les spécifications et les besoins fonctionnels de l'application. - Identifier les fonctionnalités clés liées à la réservation et à l'inscription des athlètes.

- c. **Exécution des tests** : - Vérifier que les secrétaires peuvent réserver des places pour les athlètes. - Valider que les points des clubs sont correctement gérés et échangés pour les inscriptions.
- d. **Tests de performance** : - Vérifier la capacité de l'application à gérer un grand nombre de connexion. - Mesurer les temps de réponse et la stabilité du système.
- e. **Tests de sécurité** : - Vérifier l'accès restreint aux secrétaires et la confidentialité des données.
- f. **Tests d'intégration** : - Vérifier l'intégration entre l'application et la base de données des clubs. - Valider la communication entre l'application et d'autres systèmes.
- g. **Tests de convivialité** : - Évaluer l'expérience utilisateur lors de la réservation et de l'inscription. - Vérifier la convivialité de l'interface pour les secrétaires.
- h. **Tests de régression** : - S'assurer que les modifications n'ont pas affecté les fonctionnalités existantes. - Réexécuter les tests après chaque mise à jour.
- i. **Validation finale** : - Vérifier que l'application répond aux objectifs fixés. - Assurer que les clubs peuvent inscrire un maximum de 12 athlètes par compétition.

III. Plan de test

A. Description du projet

Le projet de Réservation Légère vise à créer une version plus efficace de notre système de réservation existant, spécifiquement pour les organisateurs de compétitions locales et régionales. Voici les détails essentiels du projet :

1. Objectifs :

a. Optimisation de l'administration : Réduire la charge administrative des organisateurs en automatisant la réservation.

b. Simplicité d'utilisation : Permettre aux secrétaires des clubs d'inscrire des athlètes sans passer par l'organisateur de la compétition.

c. Gestion des points : Permettre aux clubs d'échanger des points contre des inscriptions futures.

2. Fonctionnalités clés :

a. Inscription par les secrétaires : Les secrétaires des clubs peuvent réserver des places pour les athlètes.

b. Gestion des points : Les clubs accumulent des points en organisant des compétitions et peuvent les échanger pour inscrire des athlètes.

c. Limitation des inscriptions : Chaque compétition a un nombre limité d'inscriptions, et chaque club peut inscrire jusqu'à 12 athlètes.

3. Technologies utilisées :

Flask : Framework léger pour le développement web.

JSON : Stockage des données sans base de données.

Unittest : Framework de test pour garantir la qualité du code.

Locust : Outil de test de performance.

a. Étapes de mise en œuvre : Clonage et configuration du repo : - Cloner et forker le repo depuis la branche QA. - Installer les dépendances à partir du fichier requirements.txt.

b. Analyse des bogues : - Examiner les problèmes signalés dans la section "issues". - Reproduire les bogues localement pour les résoudre.

c. Implémentation de la phase 2 : - Ajouter les fonctionnalités prévues pour la phase 2. - Assurer la compatibilité avec les fonctionnalités existantes.

d. Tests approfondis : - Tester les scénarios "happy path" et "sad path". - Suivre une approche TDD pour garantir la qualité.

e. Rapports de test et de performances : - Préparer des rapports conformes aux normes du guide de développement. - Couvrir la validation des résultats requis et la gestion des erreurs.

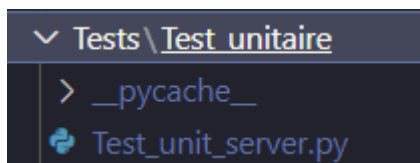
B. Objectifs des Tests dans le Projet de Réservation Légère

Test exhaustif :

Toujours tester : Ne laissons aucune fonctionnalité non testée.

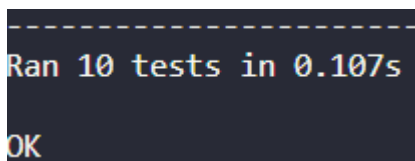
Indépendance des frameworks : Utilisons pytest, unittest, ou Morelia, mais assurons-nous de pouvoir les exécuter en ligne de commande. Nous utiliserons Unittest dans notre cas

Organisation des tests : Plaçons tous les tests dans un dossier dédié pour faciliter la recherche.



Priorités des tests :

Tests unitaires > tests d'intégration > tests fonctionnels : Priorisons les tests unitaires pour comprendre le fonctionnement et détecter les dysfonctionnements.



Objectif de couverture : Visons au moins 80% de couverture de code avec nos tests. Nous utiliserons Coverage, j'ai ressenti à recouvrir 85%

Name	Stmts	Miss	Cover	Missing
Tests\Test_unitaire\Test_unit_server.py	80	6	92%	8, 65-66, 69-77
server.py	65	10	85%	29-31, 54-55, 64-65, 83-86
TOTAL	145	16	89%	

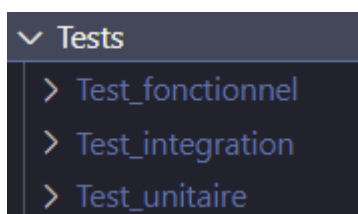
Équilibre entre types de tests :

Deux fois plus de tests unitaires : Écrivons davantage de tests unitaires que de tests d'intégration ou fonctionnels.

Sommeil tranquille : Plus de tests signifie moins de soucis.

Organisation des dossiers :

Regroupement logique : Séparons les tests unitaires, d'intégration et fonctionnels dans des dossiers distincts.



Évitons les douleurs oculaires : Une structure bien organisée facilite la maintenance.

En suivant ces objectifs, nous garantissons la qualité et la fiabilité de votre système de réservation léger.

C. Description de la Mission du Projet de Réservation Légère pour les Organisateurs de Compétitions

Le projet de **Réservation Légère** vise à créer une version plus efficace de notre système de réservation existant, spécifiquement pour les organisateurs de compétitions locales et régionales. Voici les détails essentiels de la mission du projet :

1. Optimisation de l'Administration :

- Réduire la charge administrative des organisateurs en automatisant le processus de réservation.
- Permettre aux secrétaires des clubs d'inscrire des athlètes sans passer par l'organisateur de la compétition.

2. Simplicité d'Utilisation :

- Les secrétaires des clubs pourront réserver des places pour les athlètes.
- Les clubs accumuleront des points en organisant des compétitions et pourront les échanger pour inscrire des athlètes.

3. Gestion des Points :

- Chaque compétition aura un nombre limité d'inscriptions, et chaque club pourra inscrire jusqu'à 12 athlètes.
- Le responsable de réservation supervisera l'activité des agents de réservation et coordonnera les équipes d'accueil et de réception.

4. Objectifs des Tests :

- Tester exhaustivement chaque fonctionnalité (tests unitaires, d'intégration et fonctionnels).
- Atteindre au moins 80% de couverture de code avec les tests.
- Suivre une approche TDD pour garantir la qualité.

D. Approche pour la Réalisation du Projet de Réservation Légère pour les Organisateurs de Compétitions

La réalisation du projet de réservation légère nécessite une approche méthodique et adaptée. Voici une proposition d'approche que vous pourriez envisager :

1. Analyse des Besoins :

- Comprenez en profondeur les besoins des organisateurs de compétitions et des secrétaires de clubs.
- Identifiez les fonctionnalités essentielles, les contraintes budgétaires et les délais.

2. Choix de la Méthodologie :

- Évaluons les avantages des différentes approches :
 - **Prédictive (Waterfall)** : Basée sur un périmètre défini à l'avance.

- **Adaptative (Agile)** : Priorise les besoins réels en fonction du budget et de l'échéancier.
- **Hybride** : Combinaison de méthodes prédictives et adaptatives.

3. Planification :

- Établissez un plan de projet détaillé :
 - Découpons le projet en itérations (sprints pour l'approche agile).
 - Identifions les livrables, les jalons et les responsabilités.

4. Développement :

- Pour l'approche prédictive :
 - Concevons l'application en suivant un processus séquentiel.
 - Implémentons les fonctionnalités selon le périmètre défini.
- Pour l'approche adaptative :
 - Priorisons les fonctionnalités en fonction de la valeur ajoutée.
 - Itérons rapidement pour livrer des fonctionnalités incrémentales.

5. Tests :

- Élaborons des cas de test pour chaque fonctionnalité.
- Testons exhaustivement les scénarios "happy path" et "sad path".
- Suivons une approche TDD si possible.

6. Rapports et Documentation :

- Préparons des rapports de test et de performances.
- Documentons les décisions prises, les problèmes résolus et les leçons apprises.

7. Validation et Livraison :

- Vérifions que l'application répond aux objectifs fixés.
- Assurons-nous que les clubs peuvent inscrire un maximum de 12 athlètes par compétition.

En adoptant cette approche, nous pourrions créer une solution de réservation légère efficace et adaptée aux besoins des utilisateurs.

E. La planification d'un projet est essentielle pour assurer sa réussite.

1. Comprendre l'Objectif et les Conditions du Projet :

- Analysez les besoins des organisateurs de compétitions et des secrétaires de clubs.
- Identifiez les contraintes, les délais et les ressources disponibles.

2. Définir les Tâches du Projet :

- Énumérez toutes les tâches nécessaires à la réalisation du projet.
- Hiérarchisez ces tâches en fonction de leur importance.

3. Ordonnancement des Tâches :

- Organisez les tâches dans un ordre logique.
- Déterminez les dépendances entre les tâches (quelles tâches doivent être terminées avant que d'autres puissent commencer).

4. Estimation de la Durée :

- Évaluez la durée de chaque tâche.
- Calculez la durée totale du projet en tenant compte des dépendances.

5. Attribution des Ressources :

- Identifiez les personnes ou les équipes responsables de chaque tâche.
- Allouez les ressources humaines et matérielles nécessaires.

6. Construction du Calendrier :

- Créez un calendrier avec les dates de début et de fin de chaque tâche.

- Tenez compte des jours fériés, des congés et des autres événements pertinents.

En suivant ces étapes, nous disposerons d'un planning solide pour la réalisation du projet de réservation.

F. Les bugs corrigés et fonctionnalités ajoutées

Les bugs corrigés

BUG 1 : CHAMPS SAISIR vide

Pour ce premier bug que nous avons rencontré, lorsqu'on appuie sur le bouton « ENTER » quand le champs de saisir est sans aucune donnée nous avons cette

erreur « **Internal Server Error** »

Le code que nous avons eu initialement était :

```
@app.route('/showSummary',methods=['POST'])
def showSummary():
    club = [club for club in clubs if club['email'] ==
request.form['email']][0]
    return
render_template('welcome.html',club=club,competitions=competitions)
```

pour corriger cette erreur j'ai ajouter une condition :

```
@app.route('/showSummary',methods=['POST'])
```

```
def showSummary():
    entered_email = request.form['email']

    if any(club['email'] == entered_email for club in clubs):
        # Si oui, récupère le club correspondant
        club = next(club for club in clubs if club['email'] ==
entered_email)
        return render_template('welcome.html', club=club,
competitions=competitions)

    else:

        return redirect(url_for('index'))
```

cette condition nous permet de régler cette erreur en mettant : « veuillez saisir ce champs » jusqu'à ce que l'utilisateur entre de bonne donnée.

Et dans le fichier index.html initialement s'était :

```
<input type="email" name="email" id=""/>
```

J'ai ajouté « required » dans le input pour corriger le bug

```
<input type="email" name="email" id=""required/>
```

BUG 2 : Email incorrect

Pour ce deuxième bug que nous avons rencontré, lorsqu'on appuie sur le bouton « ENTER » quand l'email saisi est incorrecte nous avons cette

erreur « **Internal Server Error** »

Le code que nous avons eu initialement était :

```
@app.route('/showSummary',methods=['POST'])
def showSummary():
    club = [club for club in clubs if club['email'] ==
request.form['email']][0]
    return
render_template('welcome.html',club=club,competitions=competitions)
```

pour corriger cette erreur j'ai ajouter une condition :

```
@app.route('/showSummary',methods=['POST'])
def showSummary():
email = request.form['email']
    club = next((club for club in clubs if club['email'] == email),
None)

    if club:
        return render_template('welcome.html', club=club,
competitions=competitions)
    else:
        flash('Adresse e-mail invalide. Veuillez entrer une adresse
e-mail valide.', 'error')
        return redirect(url_for('index'))
```

cette condition nous permet de régler cette erreur en envoyer un message d'erreur « **Adresse e-mail invalide. Veuillez entrer une adresse e-mail valide.** »

Et pour afficher un message d'erreur j'ai ajouté au fichier index.html

```
{% with messages = get_flashed_messages() %}
{% if messages %}
    <ul>
        {% for message in messages %}
            <li>{{ message }}</li>
        {% endfor %}
    </ul>
{% endif %}
{% endwith %}
```

Je l'ai inserer en dessous du champs de saisir

Bug : Réservez plus de places que celle disponible**Dans ce troisième bug**

Nous allons corriger une erreur. Lorsqu'on achète des tickets pour une compétition et que le nombre qu'on entre dans le champs de saisir est supérieur au nombre de place disponible, la réservation est pas même effectuer. Ça affiche :

- **Great-booking complete!**

Et il met la différence entre le nombre de place demandé et le nombre de place disponible.

par exemple : s'il y a 13 places disponibles et tu met 15 places il va valider et afficher la différence.

```
Fall Classic
Date: 2020-10-22 13:30:00
Number of Places: -2
```

Pour corriger ce bug, j'ai créé une condition dans la fonction

```
def purchasePlaces()
```

cette condition empeche que le nombre de place disponible dans la competition soit plus élevé au nombre de place que celles demandée.

```
if int(competition['numberOfPlaces']) < placesRequired :
    flash('Désolé, il n\'y a pas assez de places disponibles.')
    return render_template('welcome.html', club=club,
competitions=competitions)
```

En affiche a l'utilisateur : ('Désolé, il n'y a pas assez de places disponibles.')

et il devra recommencé

BUG : Points déduits du compte du club**Pour ce quatrième Bugs**

Le problème rencontrer

Jusqu'ici l'utilisateur pouvait acheter des tickets pour des places mais ses points n'était pas déduit de son compte, donc il pouvait commander en illimité.

Dans la fonction :

```
def purchasePlaces()
```

J'ai ajouté ce code :

```
club ['points'] = int (club ['points']) - placesRequired
```

Ce code ci-dessus permet de réduire les points du club en fonction de la quantité qu'il a acheté

Nous pouvons alors voir le nombre de points du club réduits comme demandé

Les fonctionnalités

Fonctionnalité 1 :

Lecture seule de nombre de points disponible pour chaque clubs

Pour notre première fonctionnalité.

Dans la PHASE 2, il nous a été demandé de rajouter un tableau public en lecture seule voici l'instruction :

« Par souci de transparence, il devrait y avoir un tableau public, en lecture seule, des totaux de points indiquant le nombre de points disponibles pour chaque club. Il ne devrait pas être nécessaire de se connecter au site pour voir la page. »

Dans cette instruction, on nous a demandé un affichage à vue unique (pas de possibilité de modifier l'affichage)

Pour ajouter cette fonctionnalité j'ai ajouté du code basique dans le fichier **index.html**

Dans la balise <FORM/>, j'ai ajouter des balises d'affichage a vue unique que sont <h3></h3> et <h4></h4>

```
<form action="showSummary" method="post">
    <h3>Points disponible des équipes</h3>
    <span></span>
    <span></span>
    <span></span>
    <h4> Simply Lift = 25 points</h4>
    <h4>Iron Temple = 4 points</h4>
    <h4>She Lifts = 12 points</h4>
</form>
```

Fonctionnalité 2 : message de confirmation le nombre de place achetée

Dans cette deuxième fonctionnalité nous avons ajouter a la fonction

```
def purchasePlaces()
```

qui permet d'acheter des tickets.

Un message flash pour informer le client du succès et du nombre de places qui a réussi a acheter.

```
flash('Génial !! Vous avez réussi à réserver {} places
:').format(placesRequired))
    return render_template('welcome.html', club=club,
competitions=competitions)
```

grâce a cette accolade {} et .format(placesRequired) nous récupérons le nombre de place acheter par le club.

Fonctionnalité 3: ne pas réserver plus de 12 places dans une compétition

Dans cette troisième fonctionnalité

Il nous a été demandé : « Ils ne doivent pas pouvoir réserver plus de 12 places dans une compétition »

Pour cette fonctionnalité j'ai ajouté une condition

```
if placesRequired > 12:
    flash('Désolé, vous ne pouvez pas réserver plus de 12 places.')
    return render_template('welcome.html', club=club, competitions=competitions)
```

dans la fonction :

```
def purchasePlaces()
```

FONCTIONNALITE 4 : Points déduits du compte du club

Pour cette quatrième fonctionnalité

Il nous a été demandé cette fonctionnalité. Voici l'instruction : « Les points utilisés sont alors déduits du compte du club. »

Jusqu'ici l'utilisateur pouvait acheter des tickets pour des places mais ses points n'étaient pas déduits de son compte donc il pouvait commander en illimité

Dans la fonction :

```
def purchasePlaces()
```

j'ai ajouté ce code :

```
club ['points'] = int (club ['points']) - placesRequired
```

Ce code ci-dessus permet de réduire les points du club en fonction de la quantité qu'il a achetée.

Fonctionnalité 5 : ne pas afficher les compétitions antérieures

Il nous a été demandé de ne pas permettre l'affichage des compétitions dépassées, de base les compétitions étaient en 2020 mais comme nous sommes aujourd'hui en 2024 j'ai modifié la date des compétitions dans le fichier « competitions.json »

les compétitions initiale

```
"competitions": [
    {
        "name": "Spring Festival",
        "date": "2020-03-27 10:00:00",
        "numberOfPlaces": "25"
    },
    {
        "name": "Fall Classic",
        "date": "2020-10-22 13:30:00",
        "numberOfPlaces": "13"
    }
]
```

La fonction permettant d'afficher les compétitions initiale

```
#Chargement des competitions
def loadCompetitions():
    with open('competitions.json') as comps:
        listOfCompetitions = json.load(comps)['competitions']
    return listOfCompetitions
```

Lorsque j'ai modifier la date

```
"competitions": [
  {
    "name": "Spring Festival",
    "date": "2024-03-27 10:00:00",
    "numberOfPlaces": "25"
  },
  {
    "name": "Fall Classic",
    "date": "2024-10-22 13:30:00",
    "numberOfPlaces": "13"
  }
]
```

Les compétitions s'affiche correctement



Maintenant on mettra la date d'une compétition antérieure à 2024 pour ne pas que la compétition qui est inferieur a 2024 s'affiche

```
"competitions": [
  {
    "name": "Spring Festival",
    "date": "2024-03-27 10:00:00",
    "numberOfPlaces": "25"
  },
  {
    "name": "Fall Classic",
    "date": "2023-10-22 13:30:00",
    "numberOfPlaces": "13"
  }
]
```

```
{
    "name": "Fall Classic",
    "date": "2023-10-22 13:30:00",
    "numberOfPlaces": "13"
}
```

Grace a cette fonction modifier

```
loadCompetitions()
```

```
def loadCompetitions():
    current_year = datetime.datetime.now().year
    with open('competitions.json') as comps:
        competitions = json.load(comps)['competitions']
        filtered_competitions = [comp for comp in competitions if
int(comp['date'][:4]) >= 2024]
    return filtered_competitions
```

dans le fichier index.html

```
<ul>
    {% for comp in competitions %}
    <li>
        <h9>{{comp['name']}}<br /> </h9>
        <h9> Date: {{comp['date']}}<br /> </h9>
        <h9> Nombre de place: {{comp['numberOfPlaces']}}
    </h9>
    </li>

    <hr />
    {% endfor %}
</ul>
```

nous ne chargeons plus les compétitions inferieur a 2024



Donc notre nouvelle fonctionnalité est fonctionnelle.

G. Locust


J'ai créé un fichier pour donner des instructions sur les routes à tester

 locustfile.py M

Voici les instructions

```
class test_de_route(HttpUser):
    @task
    def test_route(self):
        self.client.get('/')
        self.client.get('/showSummary', methods=['POST'])
        self.client.get('/purchasePlaces', methods=['POST'])
        self.client.get('/logout')
        self.client.get('/book/<competition>/<club>')
```

On lance notre teste avec 100 utilisateurs qui se connecte instantanément sur notre application web avec cette adresse IP : `http://127.0.0.1:5000`



HOST

STATUS

RPS

READY

0

Start new load test

Number of users (peak concurrency)

100

Ramp Up (users started/second)

1

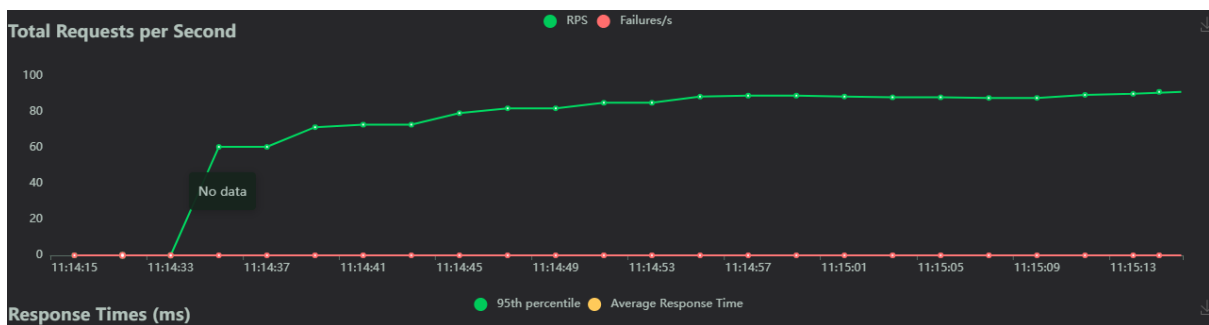
Host

http://127.0.0.1:5000

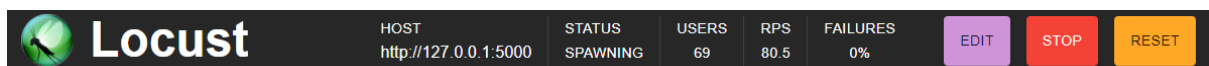
Advanced options

START SWARM

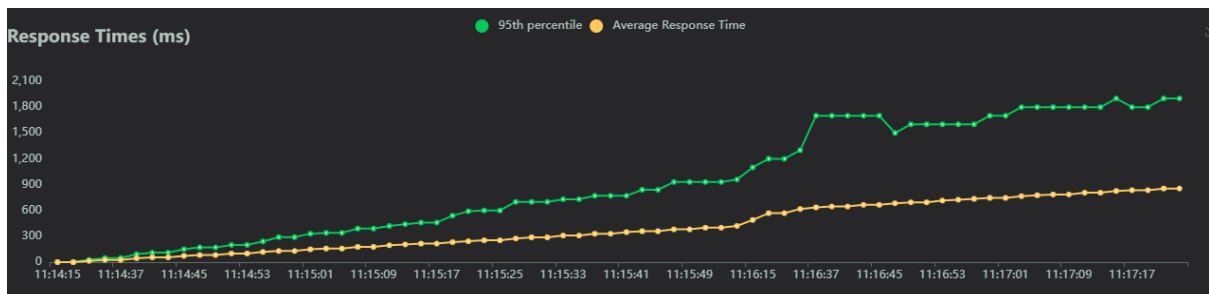
Nous pouvons voir les graphiques de requête envoyé et les échecs rencontrer



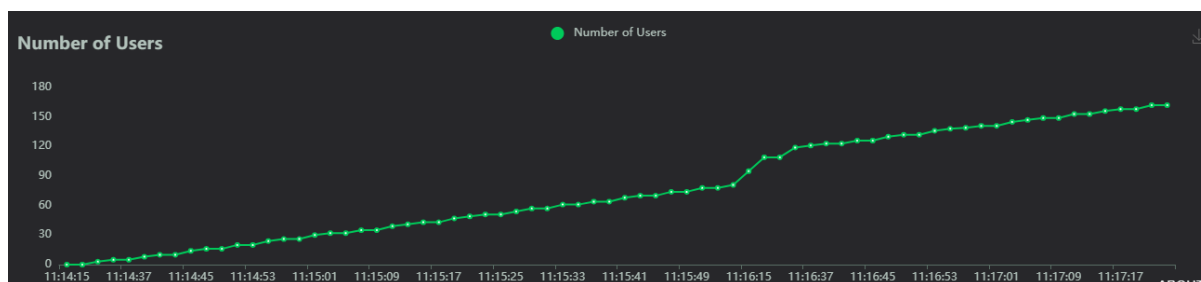
Il y a 0 % d'échec dans nos requêtes



Le temps de réponse est bien inferieur a 2 secondes



Le nombre d'utilisateur s'accroît et toujours aucune erreur de requête ou de temps de réponse élevé



H. Conclusion

Conclusion du Projet de Réservation Légère pour les Organisateurs de Compétitions

En résumé, le projet de réservation légère vise à simplifier le processus de réservation pour les organisateurs de compétitions locales et régionales. Voici les points clés à retenir :

1. **Objectifs :**
 - Réduire la charge administrative des organisateurs.
 - Permettre aux secrétaires des clubs d'inscrire des athlètes.
 - Gérer les points accumulés par les clubs.
2. **Fonctionnalités Clés :**
 - Inscription par les secrétaires.
 - Gestion des points.
 - Limitation des inscriptions par compétition.
3. **Technologies Utilisées :**
 - Flask pour le développement web.
 - Stockage des données en JSON (sans base de données).
 - Pytest pour garantir la qualité du code.
 - Locust pour les tests de performance.
4. **Approche de Réalisation :**
 - Analyse approfondie des besoins.
 - Choix d'une méthodologie adaptée (prédictive ou agile).
 - Planification des tâches, estimation des durées et attribution des ressources.
 - Développement, tests et rapports conformes aux normes.

En suivant ces étapes, nous pourrions créer un système de réservation léger efficace et répondant aux attentes des utilisateurs.

