# Pilotage du système logiciel en production

## Table des matières

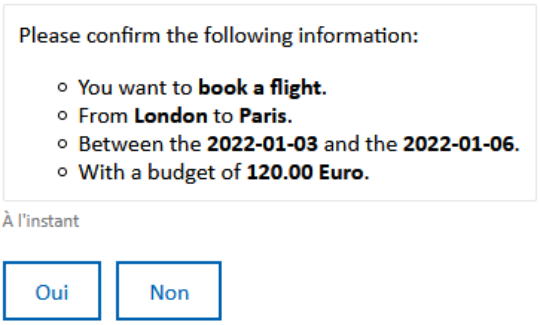
## Critères d'évaluation du modèle

### Nombre d'insatisfactions

Un des risques que l'on avait identifiés était que le bot commette beaucoup d'erreurs et que cela puisse nuire à l'image de la société.

Pour la v1 de ce MVP, nous allons de commencer par tester uniquement la satisfaction des utilisateurs par rapport à la capacité du bot à extraire correctement leurs critères.

A la fin d'un dialogue, le bot demande explicitement à l'utilisateur de valider les informations qu'il a extrait et qu'il compte utiliser pour trouver un vol :



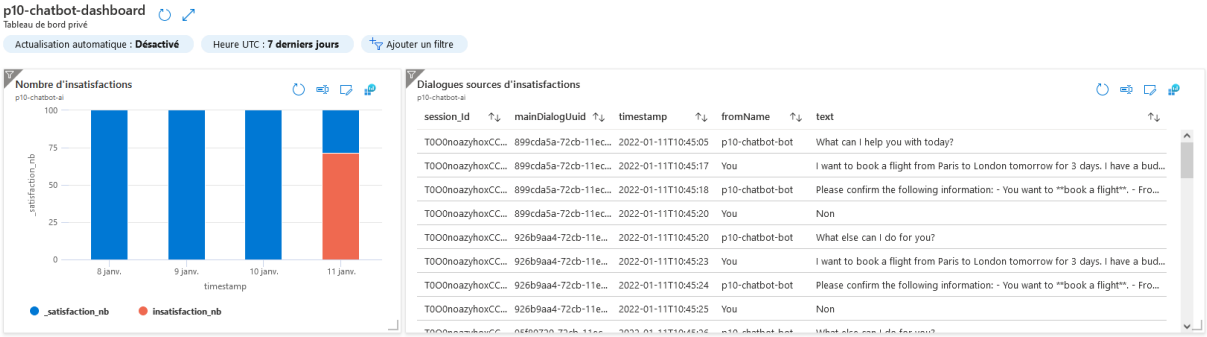

Demande de satisfaction du bot à la fin de chaque dialogue

Si l'utilisateur répond « Non », on obtiendra alors une insatisfaction. Il s'agit donc d'un score explicite fournit par l'utilisateur à la fin du dialogue.

Ce score est automatiquement évalué 1 fois par jour sur les dernières 24h par une alerte créés sur Aure App Insights. L'alerte se déclenche si on a eu plus de 5 insatisfactions sur les dernières 24h. Pour cette première version du MVP, il est a noté que nous monitorons le nombre et non le taux d'insatisfactions. En effet, nous souhaitons rapidement itérer sur le MVP et étant donné que l'application sera d'abord diffusée et testée en interne par nos collaborateurs, il n'y aura pas beaucoup d'utilisateurs. Quand nous aurons beaucoup plus d'utilisateurs, il sera plus intéressant de monitorer le taux de satisfaction.

Enfin, on ne prend pas en compte les utilisateurs qui ne sont pas allés au bout du dialogue. On pourrait par exemple mettre en place un timeout et considérer que les utilisateurs qui n'ont pas fini le dialogue et qui ont dépassé le timeout sont insatisfait. Etant donnée que l'application sera d'abord testée en interne, nous allons notifier nos collaborateurs d'aller au bout du dialogue afin de pouvoir rapidement itérer sur cette version.

On pourra aussi observer le nombre d'insatisfactions dans le dashboard du projet :

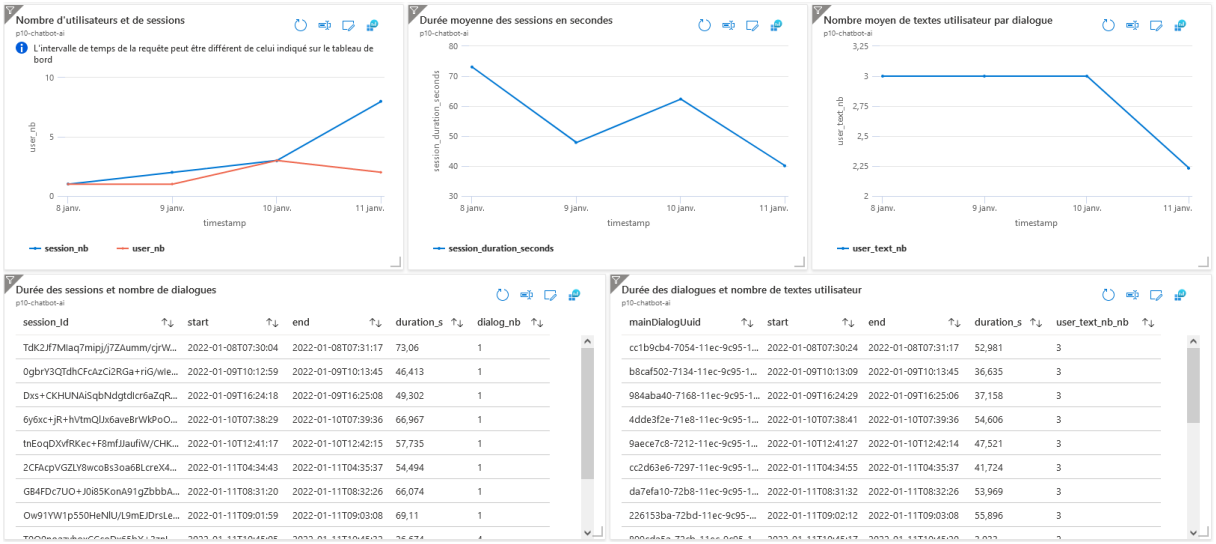



Visualisation du nombre d'insatisfactions et de leurs dialogues

## Utilisateurs et sessions

Nous allons aussi observer plusieurs métriques concernant les utilisateurs. Sur le dashboard, on pourra visualiser :

- Le nombre d'utilisateurs et des sessions par jour.
- La durée moyenne des sessions.
- Le nombre moyen de textes écrits par l'utilisateur dans un dialogue.



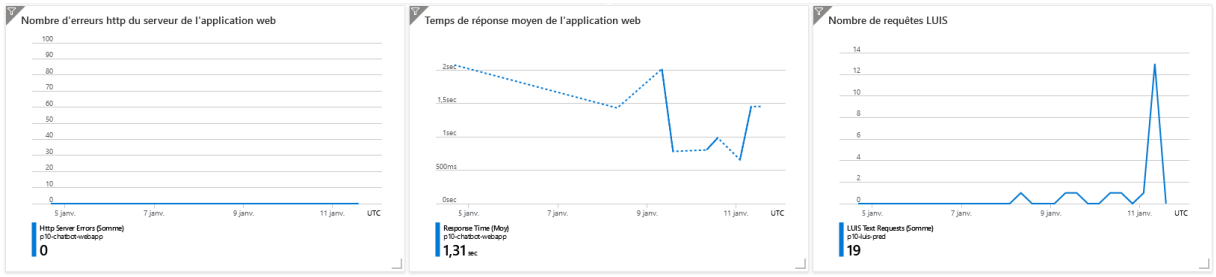

Visualisation d'informations concernant les utilisateurs

Une croissance du nombre d'utilisateurs montrera un engouement pour notre MVP. Une décroissance indiquera un manque d'intérêt ou un problème technique qui empêche sa bonne utilisation.

## Santé de l'application

Enfin, nous allons aussi visualiser des indicateurs de base qui vont nous indiquer le bon fonctionnement de l'application :

- Nombre d'erreurs http du serveur de l'application.
- Temps de réponse moyen de l'application web.
- Nombre de requêtes LUIS.




Visualisation d'informations concernant la santé de l'application

Il s'agit surtout d'effectuer un check visuel sur 3 types de problème.

Le nombre d'erreurs http va nous permettre de mettre en avant des bugs de l'application ou une utilisation frauduleuse.

Le temps de réponse moyen av nous permettre de voir s'il y a des forts ralentissements et d'agir en conséquence, comme en redimensionnant les ressources Azure qui font tourner l'application web.

Enfin, le nombre de requêtes LUIS av nous permettre de voir si des requêtes sont bien envoyée à LUIS. Cela peut servir de ping visuel.

## Modalités de mise à jour du modèle

Toutes nos données sont calculées et/ou agrégées sur les dernières 24h. Sur le dashboard, on les observera les résultats sur les 7 derniers jours. Le dashboard nous permet de modifier cette fenêtre temporelle. Cela sera notamment utile par la suite pour par exemple observer des effets de saisonnalité.

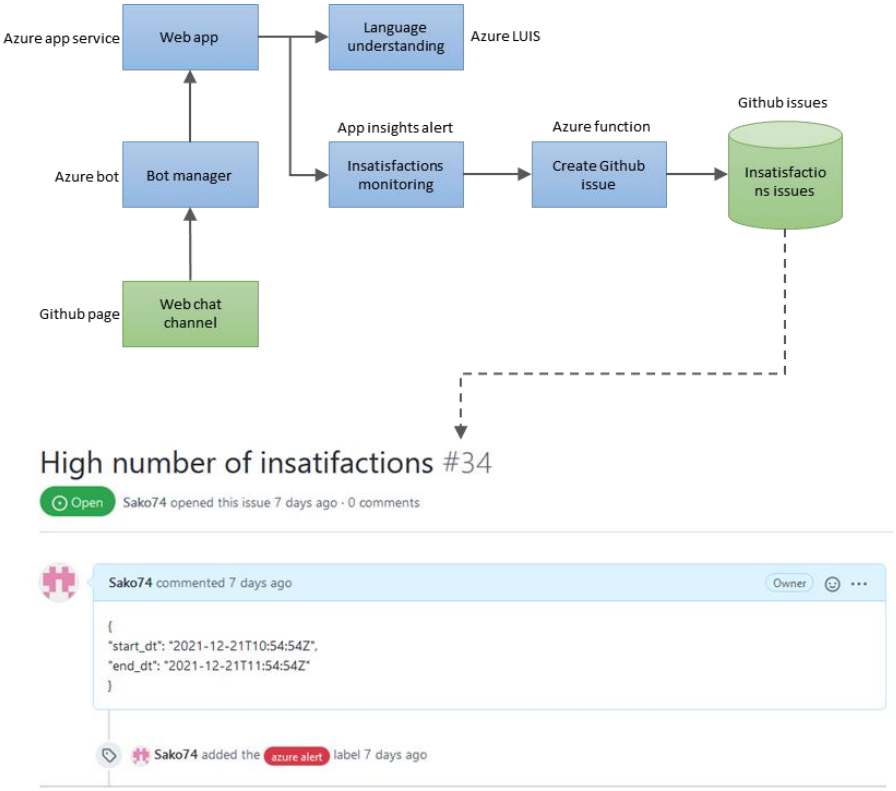
En ce qui concerne la mise à jour du modèle de language understanding et de l'application web, elles vont dans un premier temps dépendre de l'analyse des insatisfactions. En effet, nous estimons que les premières semaines du test du MVP seront dédiées à la correction de bug et l'ajout de features pour fiabiliser le chatbot. Une fois que nous aurons stabilisés le nombre d'insatisfactions et que cette v1 répondra aux attentes de nos utilisateurs, on pourra alors estimer une fréquence arbitraire de mise à jour du modèle de language understanding.

## Procédure d'analyse des insatisfactions

### Création automatique d'une Github issue

Pour rappel, nous avons mis en place une alerte automatique sur Azure qui va se déclencher automatiquement lorsqu'elle aura détecté plus de 5 insatisfactions sur les dernières 24 heures.

Cette alerte va alors déclencher une Azure function qui va créer automatiquement une Github issue :



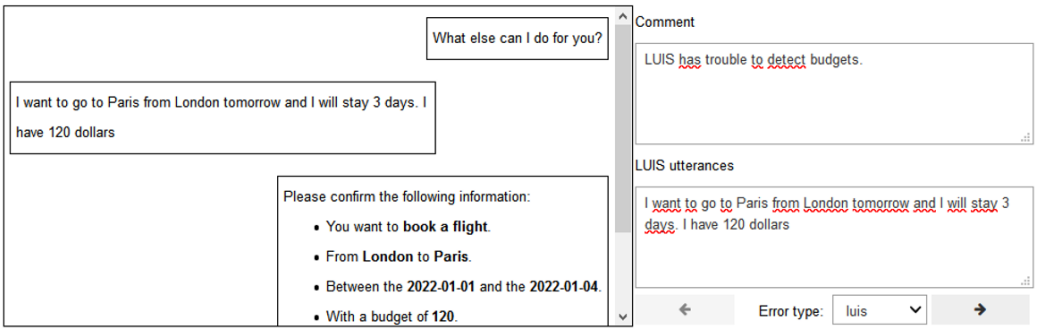

Example d'une Github issue générée automatiquement

On y a ajouté un tag « azure alert » ainsi que les informations concernant la fenêtre temporelle durant laquelle on a détecté plus de 5 insatisfactions.

## Analyse des insatisfactions

Lorsque l'on aura assez d'insatisfactions à analyser, nous pourrons utiliser le notebook « 02_analyse_insatisfactions.ipynb » pour les analyser.

Ce notebook va d'abord télécharger les dialogues source d'insatisfaction. On pourra alors utiliser un widget permettant de rejouer les dialogues un par un :




Widget permettant de visualiser les dialogues et analyser les erreurs

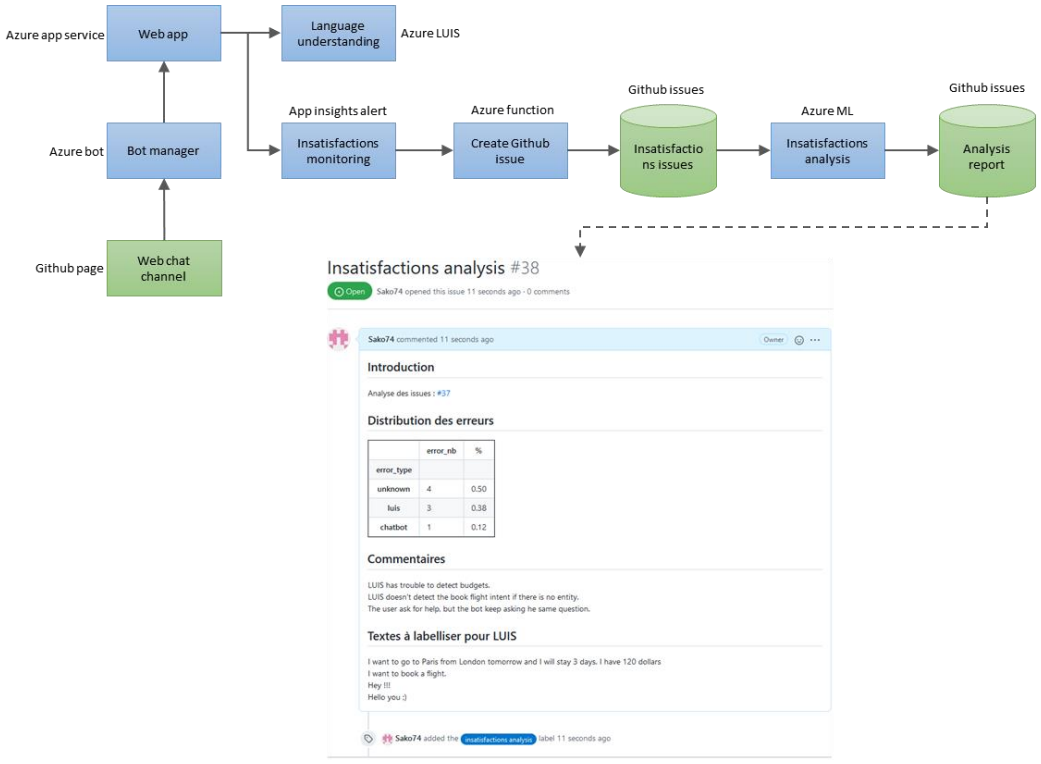
On pourra ajouter dans « Comment » des commentaires concernant la source probable de l'insatisfaction et des pistes d'amélioration.

Dans la section « LUIS utterances » on pourra ajouter des nouveaux textes à labelliser que l'on souhaiterait ajouter au jeu de données de LUIS.

Enfin, on pourra ajouter le type de l'erreur.

## Création d'un rapport d'analyse

La fin du notebook permet de créer automatiquement un rapport de l'analyse en tant que Github issue :




Example d'un rapport d'analyse sous forme d'une Github issue

Cette issue sera facilement identifiée via le tag « insatisfactions analysis ».

Dans l'introduction du rapport, on trouvera les liens vers les issues de type « azure alert » qui ont été analysée et automatiquement clôturées.

On trouvera ensuite la distribution des erreurs.

La section « Commentaires » servira de support à la prise de décision d'actions à réaliser pour améliorer le MVP :

- Mettre à jour LUIS.
- Créer de nouvelles user stories.
- Corriger des bugs.

Enfin, on retrouvera dans la dernière section les textes à labelliser et à ajouter dans le jeu de données de LUIS.

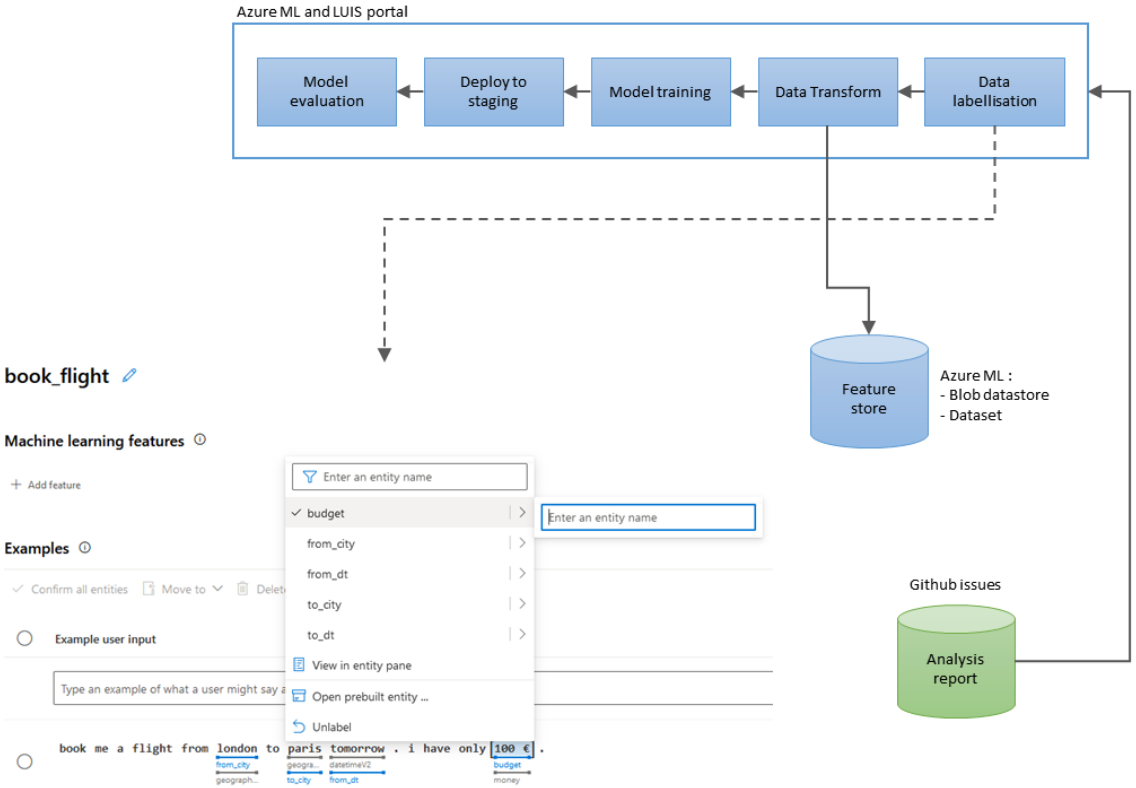
## Procédure de mise à jour de LUIS

### Mise à jour du jeu de données

Nous allons commencer par créer une nouvelle branche git.

Nous allons ensuite utiliser le notebook « 03_update_luis_model.ipynb ». On va commencer par récupérer les textes à labelliser et à ajouter à notre jeu de données à partir des rapports d'analyse des insatisfactions.

On va alors exploiter l'outil de labellisation de LUIS Portal pour labelliser nos textes :

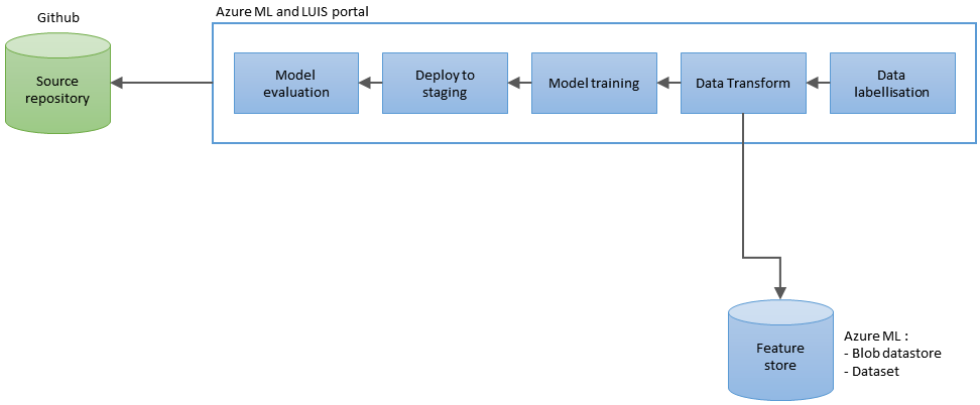



Exemple de labellisation d'une phrase sur LUIS Portal

Ensuite, nous allons ajouter ces textes dans le précédent jeu de données existant et enregistrer cette nouvelle version dans le feature store.

## Mise à jour de LUIS

Nous pouvons maintenant mettre à jour LUIS :




Mise à jour de LUIS

Nous allons créer un modèle LUIS et l'entrainer avec notre nouveau jeu de données. Nous allons le déployer sur l'environnement de test et l'évaluer avec le nouveau jeu de test :

| model_name | model_type | precision | recall | f_score |
|---|---|---|---|---|
| book_flight | Intent Classifier | 0.92 | 1.00 | 0.96 |
| None | Intent Classifier | 1.00 | 0.78 | 0.88 |
| from_dt | Entity Extractor | 0.86 | 0.90 | 0.88 |
| to_dt | Entity Extractor | 0.88 | 1.00 | 0.93 |
| budget | Entity Extractor | 0.88 | 1.00 | 0.93 |
| from_city | Entity Extractor | 0.57 | 0.85 | 0.69 |
| to_city | Entity Extractor | 0.50 | 1.00 | 0.67 |

Exemple de résultat d'évaluation du modèle

Si nous sommes satisfaits des résultats, nous pourrons enregistrer le nouveau modèle avec les informations du dataset mis à jour sur Github.

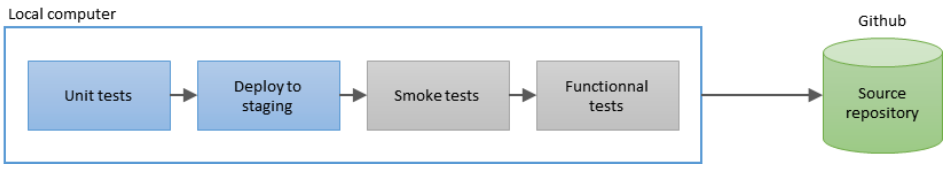
Enfin, on termine en effectuant une demande de Pull request, ce qui aura pour effet de déclencher le pipeline « LUIS test » sur Github. Les résultats de l'évaluation du modèle se retrouverons alors dans la demande de Pull request.

Si la demande est validée, la nouvelle version de LUIS sera mergée dans la branche principale et sera prête à être déployée.

## Procédure de mise à jour de l'application web

Comme précédemment, on commence par créer une nouvelle branche git.

On effectue ensuite en local les modifications et tests de l'application web :




Mise à jour de l'application web

Si nous sommes satisfaits des résultats, nous pourrons enregistrer le code source de l'application mise à jour sur Github.

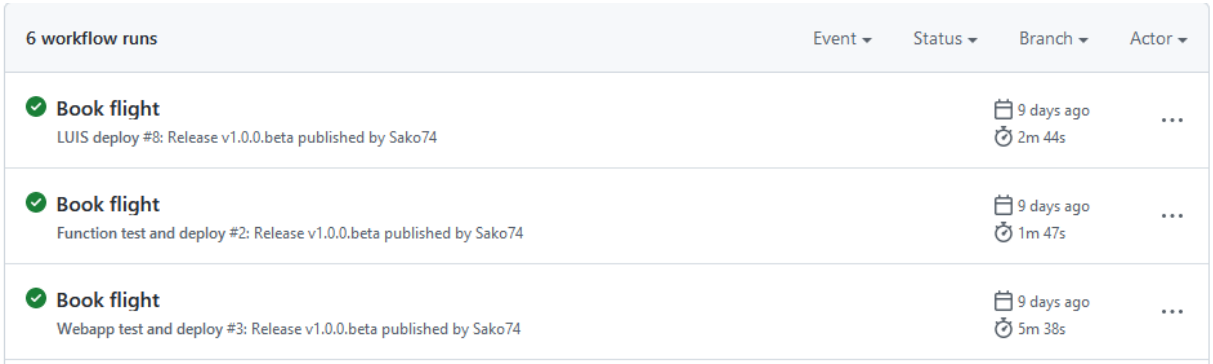
Enfin, on termine en effectuant une demande de Pull request, ce qui aura pour effet de déclencher le pipeline « Webapp test » sur Github. Les résultats des tests de l'application web se retrouverons alors dans la demande de Pull request.

Si la demande est validée, la nouvelle version de l'application web sera mergée dans la branche principale et sera prête à être déployée.

## Mise en production de la nouvelle version du système logiciel

Pour mettre à jour le système logiciel en production, il suffit de créer une nouvelle release sur Github. Cela aura pour effet de déclencher les pipelines suivants :

- LUIS deploy
- Webapp test and deploy
- Function deploy




Exemple d'exécution des pipelines de déploiement du système logiciel en production pour la release v1.0.0.beta

Le pipeline « Function deploy » est chargé de mettre à jour l'Azure function qui va créer automatiquement une Github issue lorsqu'une alerte d'insatisfaction sera déclenchée.