



TripMaster Application TourGuide

PRÉSENTATION DE L'APPLICATION

Justine CABROL

Parcours: Développeur d'Applications – Java

Projet n°8: Améliorez votre application avec des systèmes distribués

TourGuide



Changez votre manière de voyager



Sommaire

- 1) Présentation de l'application
- 2) Spécifications techniques
- 3) Schémas de conception
 - a) Diagramme UML de classes
 - b) Présentation de l'architecture TourGuide
- 4) Problèmes à résoudre
- 5) Solutions proposées
 - a) Au niveau des fonctionnalités
 - b) Au niveau des performances
 - c) Au niveau de la maintenance
- 6) Performances de l'application
- 7) Rapports de test
- 8) Rapport de couverture de code
- 9) Intégration Continue avec GitLab

TourGuide



Changez votre manière de voyager



1) Présentation de l'application

TourGuide est une application SpringBoot permettant aux utilisateurs de voir quelles sont les attractions touristiques à proximité du lieu où ils se trouvent et d'obtenir des réductions sur les séjours d'hôtel ainsi que sur les billets de différents spectacles.

Pour cela TourGuide fait appel au service gpsUtil afin de collecter l'emplacement géographique de l'utilisateur.

Pour que l'application soit efficace, cet emplacement doit être mis à jour régulièrement.

L'application interagit également avec le service RewardsCentral pour obtenir des récompenses pour les attractions via un réseau de fournisseurs.

Elle fait également appel au service TripPricer, lié à différentes agences, qui fournit des offres de séjours personnalisées pour l'utilisateur.



2) Spécifications techniques

- Java 1.8
- Gradle 4.8.1
- SpringBoot 2.1.6
- JUnit 4.1.0
- Mockito
- JaCoCo
- GitLab



3) Schémas de conception

a- Diagramme UML de classes

Objet de l'application TourGuide:

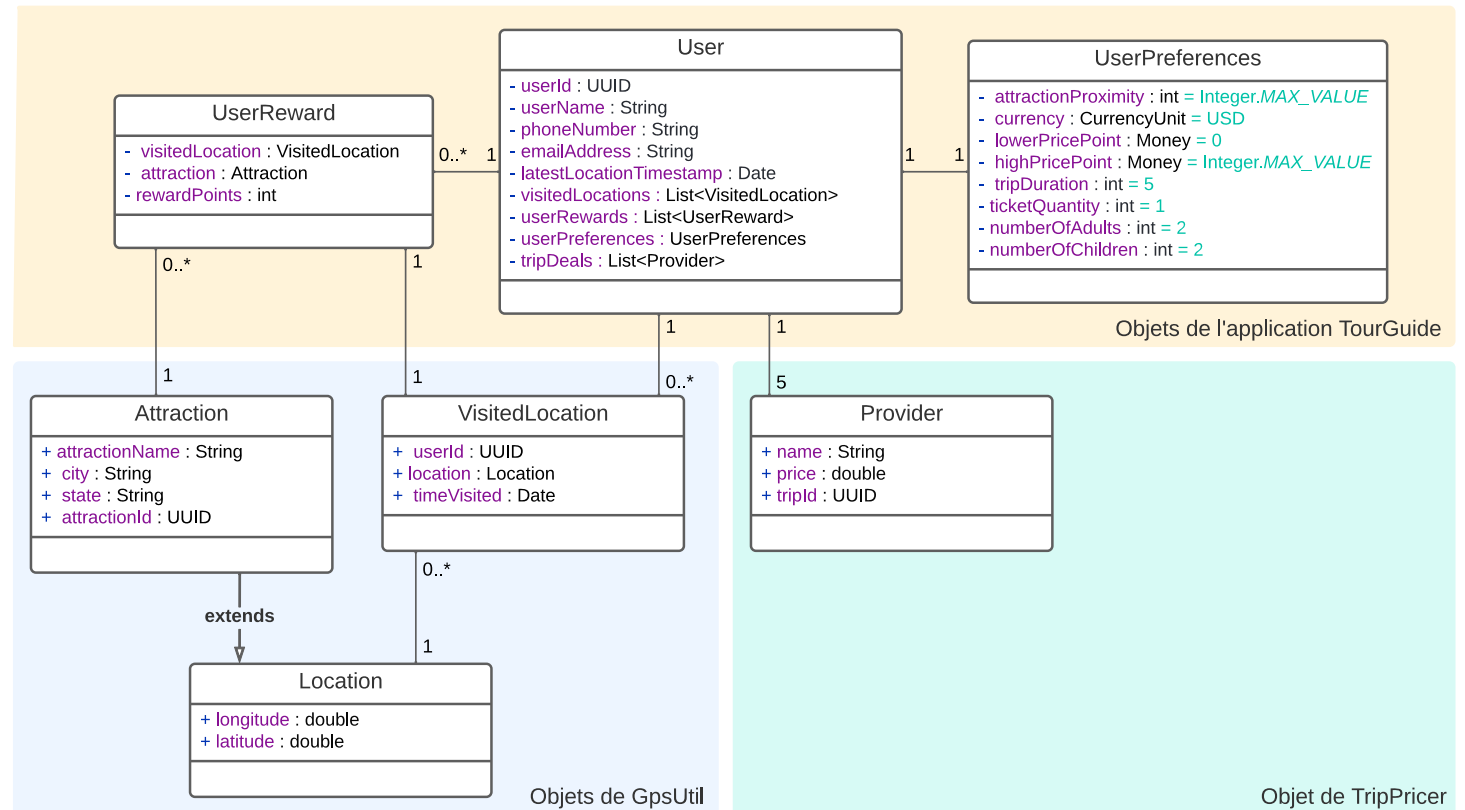
- User
- UserReward
- UserPreferences

Objets importés du service GpsUtil:

- Location
- Attraction
- VisitedLocation

Objet importé du service TripPricer:

- Provider





3) Schémas de conception

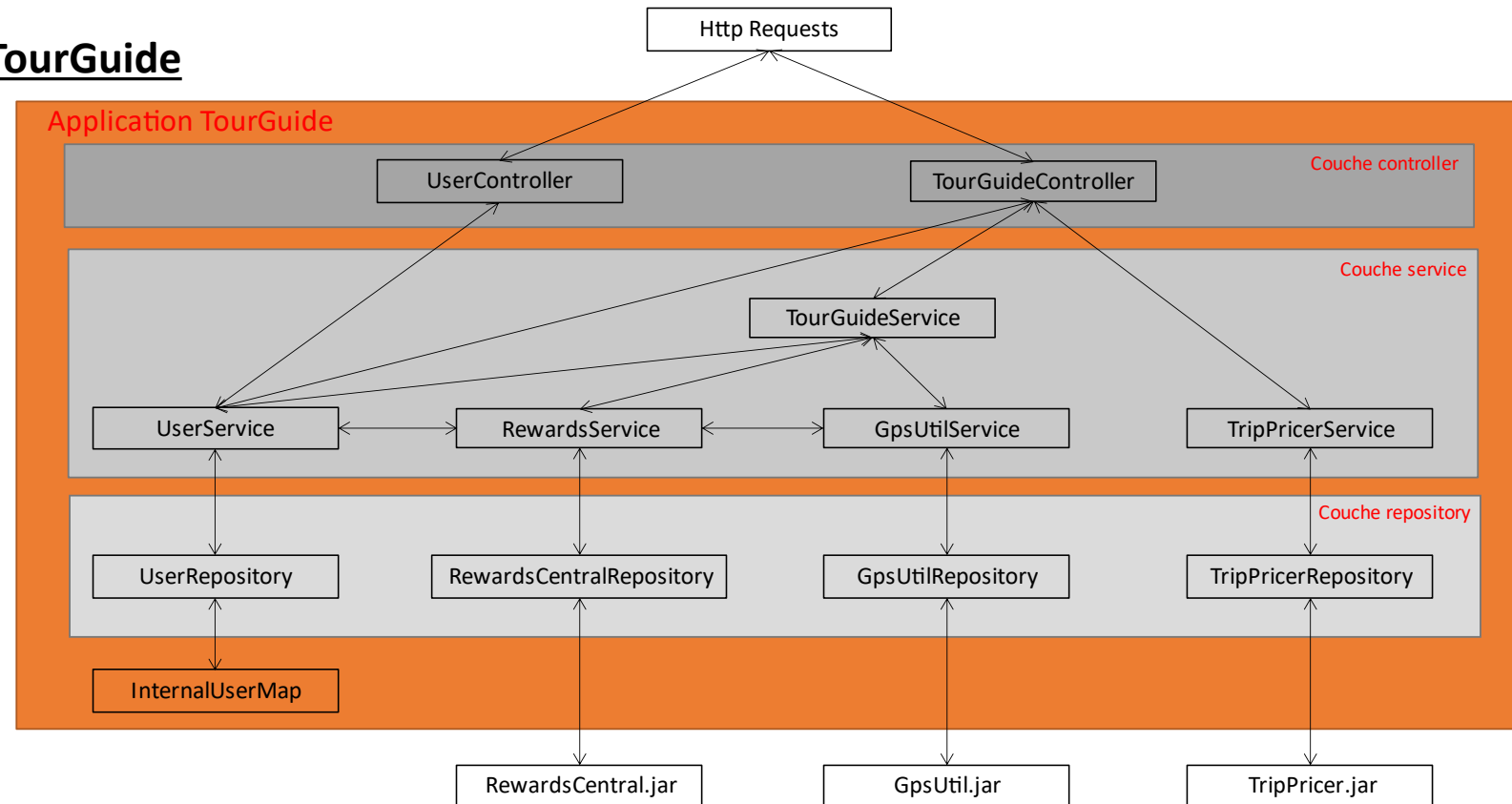
b- Présentation de l'architecture TourGuide

Respect des principes SOLID

- Principe de responsabilité unique
- Principe ouvert/fermé
- Principe de substitution de Liskov
- Principe de ségrégation des interfaces
- Principe d'inversion des dépendances

Modèle MVC et architecture 3-tiers

- Couche controller pour communiquer avec les utilisateurs
- Couche service pour exécuter la logique métier
- Couche repository pour communiquer avec les données extérieures





4) Problèmes à résoudre

a- *Au niveau des fonctionnalités*

- Des attractions ne sont pas toujours proposées à l'utilisateur
- Les propositions de séjour ne tiennent pas toujours compte des préférences de l'utilisateur
- Il faut ajouter une fonctionnalité permettant d'obtenir la localisation de tous les utilisateurs connectés

b- *Au niveau des performances*

- La localisation des utilisateurs est trop lente lorsqu'il y a beaucoup d'utilisateurs connectés
- Le calcul des récompenses est trop lent lorsqu'il y a beaucoup d'utilisateurs connectés

c- *Au niveau de la maintenance*

- Le code est difficilement maintenable, il doit être réorganisé
- Certains tests échouent, les tests doivent être corrigés et complétés
- Il faut assurer un suivi plus régulier des tests pour éviter les problèmes futurs



5) Solutions proposées

a- Au niveau des fonctionnalités

- **Modification de la fonction « *getNearbyAttractions* » :**

Elle fait appel à la fonction « *searchFiveClosestAttractions* » qui retourne les 5 attractions les plus proches de l'utilisateur, quelle que soit la distance, avec leur nom et leurs coordonnées géographiques.

- **Ajout d'un « *UserController* » :**

Il propose des méthodes permettant de visualiser et mettre à jour les données et préférence de l'utilisateur, afin de pouvoir en tenir compte notamment lors du calcul des « *TripDeals* ».

- **Ajout d'une fonction « *getAllCurrentLocations* » :**

Elle permet d'obtenir la position de tous les utilisateurs à partir de la dernière position enregistrée (sauf si l'utilisateur n'a jamais été localisé, dans ce cas la position actuelle est calculée).

Elle affiche la position de chaque utilisateur avec son identifiant.



5) Solutions proposées

b- Au niveau des performances

- ***Modification de la méthode utilisée par le Tracker :***

La méthode « trackAllUsers » prend maintenant en paramètre la liste des utilisateurs, elle utilise une « RecursiveTask » avec un « ForkJoinPool » pour obtenir la localisation de tous les utilisateurs et une « LinkedBlockingQueue » pour mettre ensuite à jour les positions des utilisateurs et calculer les éventuelles récompenses.

Ainsi la méthode est plus rapide, la position des utilisateurs est mise à jour régulièrement et la dernière position enregistrée peut être utilisée lorsqu'un utilisateur a besoin d'être localisé.

- ***Modification de la méthode pour calculer les récompenses :***

La liste des attractions est mise en attribut de la classe « RewardsService » lors de son instanciation pour un accès plus rapide. Elle peut être mise à jour en cas de modifications avec le setter.

La méthode « calculateRewards » utilise une « RecursiveTask » avec un « ForkJoinPool » pour parcourir la liste des attractions et des userRewards de l'utilisateur et renvoyer uniquement les « RewardElements » pour lesquels il faut ajouter une récompense à l'utilisateur.

Ces derniers sont ensuite soumis à un « ExecutorService » pour calculer les points de récompense et les ajouter à l'utilisateur.

Cette implémentation aura l'avantage de rester rapide même si le nombre d'attractions ou d'userRewards augmente.



5) Solutions proposées

c- Au niveau de la maintenance

- **Réorganisation du code :**

Le code a été réorganisé pour correspondre aux principes SOLID et au modèle MVC.
De cette façon il sera plus facilement maintenable et modifiable.

- **Tests :**

L'application est couverte par des tests unitaires et d'intégration, la couverture de code est mesurée par JaCoCo.

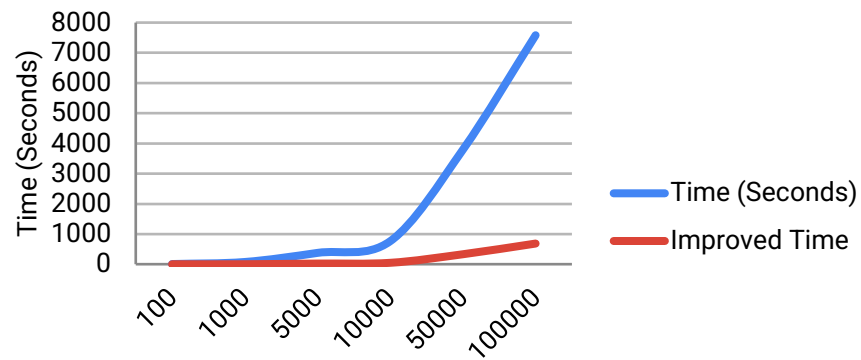
- **Suivi :**

Un pipeline est installé sur GitLab afin d'avoir une intégration continue et d'effectuer les tests avant chaque commit.
De cette façon le suivi est régulier et les éventuels dysfonctionnement sont immédiatement repérés.



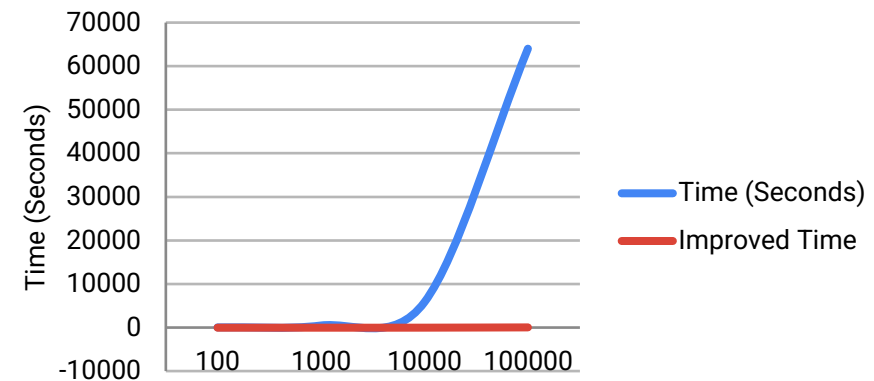
6) Performances de l'application

Get Location: Time / Users



	Users	Time (Seconds)	Improved Time
Get Location	100	7	1
	1000	75	2
	5000	376	24
	10000	762	49
	50000	3791	331
	100000	7579	683

Get Rewards: Time / Users



	Users	Time (Seconds)	Improved Time
Get Rewards	100	44	5
	1000	472	5
	10000	5820	8
	100000	64020	40



7) Rapports de test

Test Summary

116 tests 0 failures 0 ignored 14m28.12s duration

100%
successful

- Tests unitaires
- Tests d'intégration
- Tests de performance

Packages

Classes

Rapport de test généré lors de l'exécution des tests avec Gradle.




























Class	Tests	Failures	Ignored	Duration	Success rate
tourGuide.TestPerformance	2	0	0	14m10.21s	100%
tourGuide.integrationTests.TourGuideControllerIntegrationTests	16	0	0	7.458s	100%
tourGuide.integrationTests.UserControllerIntegrationTests	18	0	0	0.227s	100%
tourGuide.unitTests.RewardsServiceUnitTests	12	0	0	3.243s	100%
tourGuide.unitTests.TourGuideControllerUnitTests	15	0	0	0.389s	100%
tourGuide.unitTests.TourGuideServiceUnitTests	10	0	0	5.121s	100%
tourGuide.unitTests.UserControllerUnitTests	18	0	0	0.956s	100%
tourGuide.unitTests.UserServiceUnitTests	25	0	0	0.516s	100%



8) Rapports de couverture de code

Rapport JaCoCo généré lors de l'exécution des tests avec Gradle.

TourGuide

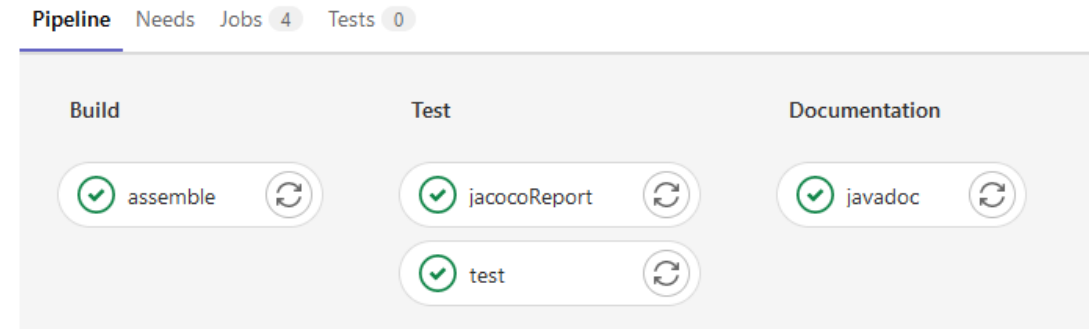
Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
 tourGuide.repository		85 %		83 %	3	24	3	36	1	18	0	4
 tourGuide.exception		87 %		n/a	2	16	3	34	2	16	0	4
 tourGuide.service		98 %		100 %	2	76	8	208	2	54	0	5
 tourGuide.model.DTO		94 %		n/a	2	30	0	31	2	30	0	5
 tourGuide.dataSource		94 %		n/a	2	12	1	25	2	12	0	1
 tourGuide		37 %		n/a	1	2	2	3	1	2	0	1
 tourGuide.model		98 %		n/a	1	39	1	31	1	39	0	3
 tourGuide.tracker		96 %		50 %	2	6	2	30	0	4	0	1
 tourGuide.helper		72 %		n/a	1	4	1	5	1	4	0	1
 tourGuide.recursiveTask		100 %		100 %	0	9	0	45	0	4	0	2
 tourGuide.controller		100 %		n/a	0	14	0	29	0	14	0	2
 tourGuide.configuration		100 %		50 %	2	8	0	26	0	6	0	2
Total	99 of 2 391	95 %	6 of 74	91 %	18	240	21	503	12	203	0	31

9) Intégration Continue avec GitLab



Lien du repository du projet sur GitLab : <https://gitlab.com/JCabrol/tourguideproject>

Exécution de 4 jobs répartis en 3 Stages



Création d'artéfacts téléchargeables sur GitLab :

jar, rapports de tests, rapports de couverture JaCoCo, javadoc

PipelineNeedsJobs4Tests0

Status	Job	Stage	Name	Duration	Coverage
<div>passed</div>	#2756737348 main ↻ 5b62e9c8	documentat...	javadoc	00:01:10 4 days ago	<div>↺↻</div>
<div>passed</div>	#2756737346 main ↻ 5b62e9c8	test	test	00:12:13 4 days ago	<div>↺↻</div>
<div>passed</div>	#2756737344 main ↻ 5b62e9c8	test	jacocoReport	00:01:10 4 days ago	<div>↺↻</div>
<div>passed</div>	#2756737342 main ↻ 5b62e9c8	build	assemble	00:01:10 4 days ago	<div>↺↻</div>