TourGuide

TourGuide

Documentation fonctionnelle et technique

Sommaire

1. Présentation du projet

TourGuide est une application Web développée en ASP.NET Core Web API, conçue pour aider les utilisateurs à découvrir rapidement les attractions touristiques situées à proximité. Elle leur offre également l'accès à des offres promotionnelles sur des hébergements et divers spectacles. L'application a connu une croissance stable, avec plusieurs centaines d'utilisateurs par jour, ce qui a conduit l'équipe de développement à travailler sur d'autres projets il y a environ 6 mois.

Cependant, récemment un article publié dans un magazine de voyages très influent a changé la situation : en l'espace de 2 mois, plus de 30 000 nouveaux utilisateurs ont rejoint la plateforme. Les projections actuelles estiment une montée en charge allant jusqu'à 100 000 utilisateurs quotidiens dans un futur très proche.

Cette croissance rapide représente à la fois une opportunité commerciale majeure et un défi technique nécessitant une modernisation de la solution.

1.1 Objectifs du projet

Pour prévoir cette explosion de fréquentation, nous avons plusieurs objectifs pour ce projet :

- *Améliorer les performances :* L'application n'était pas faite pour fonctionner avec autant d'utilisateurs, ce qui fait qu'elle était trop lente et les utilisateurs s'en plaignaient. Il faudrait qu'elle supporte facilement 100 000 utilisateurs.
- *Tests unitaires qui ne passent pas :* NearAllAttractions et GetTripDeals ne passent pas les tests.
- *Ajout d'une fonctionnalité :* Il faudrait pouvoir proposer aux utilisateurs les 5 attractions les plus proches d'eux.
- *Mise en place d'un pipeline d'intégration continue*

1.2 Hors du champ d'application

- Le front-end n'a pas été couvert par ce projet
- La mise en place d'un système de connexion/authentification, via Identity par exemple, n'est pas dans le projet actuellement
- Le calcul de différents itinéraires selon le moyen de déplacement de l'utilisateur est absent (le trajet n'est pas le même en voiture/vélo/à pied)
- Un système de notation pourrait être appréciable, pour que les utilisateurs puissent noter les attractions qu'ils ont vues/visitées, afin de créer une sorte de communauté sur l'application

1.3 Mesures du projet

- Les tests unitaires de PerformanceTest, c'est à dire HighVolumeGetRewards et HighVolumeTrackLocation, passent bien jusqu'à 100 000 utilisateurs dans les temps

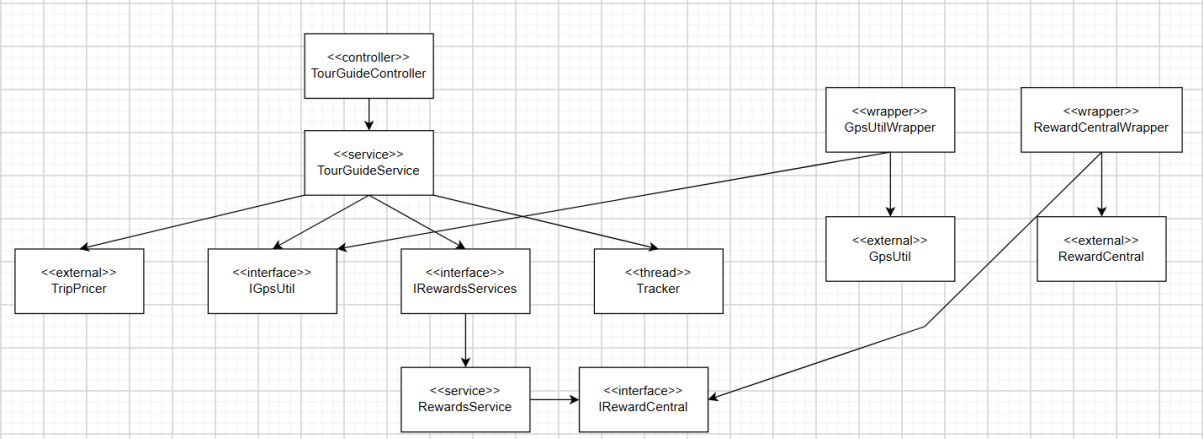
recommandés grâce à l'amélioration des performances des appels de différentes fonctions.
- Aucune erreur n'est à déplorer dans le projet actuellement, notamment grâce à la mise en place du pipeline d'intégration continue via GitHub Actions

2. Spécifications fonctionnelles

- *Localisation des utilisateurs :* Récupération de la position GPS approximative des utilisateurs grâce au endpoint GET/getLocation/{userName}, utilisé pour afficher les attractions proches et calculer les récompenses basées sur les visites des attractions. De plus, l'application possède un système de tracking en arrière-plan qui permet de mettre à jour de manière continue la position des utilisateurs
- *Attractions touristiques à proximité des utilisateurs :* l'utilisateur peut obtenir une liste des attractions autour de sa position avec l'endpoint GET/getNearByAttractions/{userName}, grâce à ça chaque utilisateur peut voir les 5 attractions les plus proches ainsi que leur distance
- *Récompenses utilisateurs :* le système calcule automatiquement les récompenses en fonction des attractions visitées grâce à l'endpoint GET/getRewards/{userName}, en sachant qu'une visite d'attractions génère un certain nombre de points qui doit être stocké par utilisateur
- *Proposition de séjours :* l'utilisateur peut obtenir des offres personnalisées (hôtels, spectacles,...) grâce à l'endpoint GET/getTripDeals/{userName}, ce qui permet de retourner toujours 10 offres en prenant en compte différents critères tel que le cumul des points, les préférences de l'utilisateur, la durée du séjour, le nombre d'adultes et d'enfants

3. Spécifications techniques

3.1 Schémas de conception technique



3.2 Glossaire

Terme	***Définition***
User	Correspond à un utilisateur de l'application
Tracker	Thread qui met à jour la position des utilisateur de manière régulière
Attraction	Est le nom donné aux lieux, villes, coordonnées GPS qui suscitent un point d'intérêt dans l'application
Reward (utilisé suivi de : Points, Services, Central)	Correspond aux points donnés à un utilisateur après la visite d'une attraction
TripPricer	Librairie externe générant des offres de voyage personnalisées en fonction des préférences et des points de récompense d'un utilisateur
GpsUtil	Librairie externe fournie pour obtenir les coordonnées GPS d'un utilisateur ainsi que la liste des attractions enregistrées dans l'application
Wrapper	Classe intermédiaire qui encapsule une librairie externe afin de fournir une interface stable, testable et indépendante de l'implémentation réelle
Controlleur	Point d'entrée de l'API , il reçoit les requêtes du front et renvoie les réponses en appelant les services
Service	Classe métier qui contient la logique principale de l'application et coordonne les appels aux différents composants nécessaires à l'exécution d'une fonctionnalité

3.3 Solutions techniques

- *Amélioration des performances de l'application :*
  Pour HighVolumeGetRewards : Afin de traiter 100 000 utilisateurs qui obtiennent leurs récompenses en moins de 20min, il a fallu optimiser la fonction CalculateRewards en surchargeant la méthode pour qu'elle retourne soit une liste d'utilisateurs, soit un seul utilisateur et en la rendant asynchrone. Dans la surcharge qui renvoie une liste d'utilisateurs on utilise du parallélisme avec Parallel.ForEachAsync pour pouvoir calculer les récompenses de plusieurs utilisateurs dans différents threads (sur la méthode CalculateRewards qui renvoie qu'un utilisateur). Ainsi, grâce à cette optimisation, le test passe en 39ms pour 100

000 utilisateurs contre plus de 17h.
Pour HighVolumeTrackLocation : Avec le même but de traiter 100 000 utilisateurs mais cette fois ci en 20min, il a fallu optimiser la fonction TrackUserLocation en utilisant la même technique le test précédent, c'est à dire en faisant une surchage de méthode pour renvoyer soit une liste d'utilisateurs, soit un seul utilisateur. Dans la surcharge qui renvoie une liste d'utilisateurs on utilise du parallélisme avec Parallel.For pour pouvoir ajouter les lieux visités de plusieurs utilisateurs sur différents threads (sur la méthode TrackUserLocation qui renvoie qu'un utilisateur). Ainsi, grâce à cette optimisation, le test passe en 3min pour 100 000 utilisateurs contre plus de 2h.

- *Autres optimisations :* Pour le test *HighVolumeGetRewards*, plusieurs méthodes sont appelés par CalculateRewards, il a donc fallu les optimiser en les passant en Task (GetRewardPointsAsync, GetAttractionRewardPointsAsync). Pour cette dernière, en plus de la passer en méthode asynchrone, on a dû l'améliorer en changeant le Thread.Sleep par Task.Delay, ce qui permet de ne pas bloquer le thread en le libérant immédiatement.
- *Ajout d'une nouvelle fonctionnalité :* Modification de l'endpoint GetNearbyAttractions afin qu'il renvoie désormais les 5 attractions les plus proches de l'utilisateur. Pour cela, on récupère toutes les attractions, on calcule pour chacune la distance avec la position de l'utilisateur et les points de récompense associés, puis on les trie par distance et on garde seulement les 5 premières. Enfin, on renvoie un DTO structuré contenant le nom de l'attraction, ses coordonnées, la localisation de l'utilisateur, la distance en miles (arrondie) et les rewards points
- *Tests unitaires qui ne passaient pas :* .
  Pour NearAllAttractions : Le test échouait car CalculateRewards n'attribuait des récompenses qu'aux attractions « proches », en fonction du proximityBuffer. Sauf que le test attend que toutes les attractions donnent une récompense, même très éloignées. En configurant le proximityBuffer avec int.MaxValue, la condition devient vraie pour toutes les attractions, ce qui permet d'attribuer une récompense pour chacune et de faire passer le test.
  Pour GetTripDeals : Le test échouait car CalculateRewards ne générait pas toutes les récompenses, ce qui faisait que parfois l'utilisateur n'avait pas de récompenses, vu que c'était uniquement les attractions les plus proches qui en avaient une.
- *Mise en place d'un pipeline d'intégration continue :* Grâce à GitHub Action, ce qui permet de rester tous sur la bonne version du projet (téléchargable sous forme de zip)

3.4 Autres solutions non retenues

- *Modifier directement les tests :* Ce n'est pas une solution qui a été retenue car elle ne correspond pas à la démarche TDD et il était précisé dans le cahier des charges de ne pas modifier les tests unitaires
- *Utilisation de Jenkins au lieu de GitHub Action :* Plus complexe pour un résultat presque similaire, de plus il était demandé de faire au plus simple en respectant le pattern KISS

- *Utilisation des Tasks/async-await au lieu du parallélisme* : Plus complexe et moins adapté à la situation car il s'agissait de calculs assez lourds