haaaaaaaaaa
Clever-Party-Thrower, une application web hébergée via Kubernetes et Ansible

Louis De Wilde

Technologie de l'informatique 3TL2
Ephec : Av. du Ciseau 15, 1348
Ottignies-Louvain-la-Neuve
Novembre 2022

Contents

1 Cahier des charges

1.1 Choix du sujet

En tant qu'étudiant (ou personnes aimant nous amuser de manière générale), nous sommes souvent amenés à organiser une soirée ou une fête entre amis. Il faut alors trouver une date qui convient au au plus grand nombre et gérer les dépenses de la fête, ce qui peut s'avérer compliqué quand le nombre de participants augmente. C'est pourquoi j'ai décidé de créer une application web facilitant l'organisation de petites fêtes estudiantines.

1.2 Cahier des charges

1.2.1 Besoin du client

- Permet de répartir équitablement les frais engendrés par la fête
- Permet de créer une playlist commune (sur laquelle chacun a la possibilité d'ajouter des titres)
- Crée une liste des courses pour la soirée (éventuellement une répartition si toutes les courses ne se font pas au meme endroit ?)
- Connaître qui a amène quoi à la fête
- Crée différents évènements
- Est accessible grâce à un simple lien
- Est pourvu d'un système de connexion et de création de compte simple
- Permet de trouver une date d'évènement qui convient au plus grand nombre de participants
- Permet au participant d'organiser facilement des covoiturages
- Permet au conducteur de connaitre son trajet de covoiturage
- Calcule et reparti automatiquement les frais de transport entre les différents covoitureurs

1.2.2 Contraintes

- Accessible sur toutes les plateformes (Android, IOS, Mac, Windows, Linux)
- Le système de création de compte doit être simple
- Les données des utilisateurs seront sécurisées
- L'application doit être auto-hébergable

1.2.3 Méthodologie

N'ayant pas de client autre que moi, je ne compte pas appliquer de méthodologie vraiment spécifique (comme scrum par exemple).
En revanche, je vais m'inspirer de cette dernière en divisant les demandes de l'utilisateur en petites tâches.
Afin de suivre l'avancement du projet et de m'organiser au mieux, j'utiliserai FocalBoard, qui me servira à créer des cartes représentant les différentes tâches et me permettra de les gérer visuellement.
Le code, quant à lui, sera géré sur Github[1]. Je voudrais si possible, pour certaines parties du projet, utiliser de l'intégration voire du déploiement continu, en plus de tests automatisés.

[1] https://github.com/craftbrak/Clever-Party-Thrower

2 Analyse

Le projet ayant comme contrainte principale d'être disponible sur toutes les plateformes majeures, une base web me parait évidente. Pourtant, le web implique aussi avec la limitation principale qu'est le besoin de connection permanente au serveur.

Pour remédier à ce problème une PWA (Application Web Progressive) nous permettrait de bénéficier des avantages du web, tout en permettant aux utilisateurs de conserver une version locale de l'application. De plus une PWA permet d'etre installée comme une application native Android ou IOS pour les plateformes mobiles, ce qui facilite l'accès.

Une PWA comme toute application web nécessite plusieurs composants:

- Une interface utilisateur

  un gestionnaire de données

  un gestionnaire de requêtes

  une libraire de composant graphique

- Une Couche de calculs applicatifs et de persistance de données

  Une Base de Donnée

  Un ORM

  Une API définie

  Un système de matching pour le covoiturage

  Un système de création de trajet pour le covoiturage

  Un système de matching pour les dépenses

2.0.1 TypeScript

Avant de regarder quels outils utiliser pour les différents composants je pense qu'il est important de choisir un language de programmation.

Pour ce projet j'ai décidé d'utiliser le même language pour les 2 composants majeurs que sont le front-end et le back-end. Plusieurs choix s'offraient

alors à moi, JavaScript, Kotlin, TypeScript ou n'importe quel langage compilable en web assembly

JavaScript n'était pas vraiment un bon choix vu qu'il ne propose pas de typage, ce qui rend le code moins prédictible et robuste.

Kotlin est un bon candidat, il offre un typage statique, mais propose de l'inférence de type, peu être transpiler vers du JavaScript ou du web assembly, les outils de développement pour kotlin sont aussi très bons, mais la documentation pour une utilisation web est plutôt pauvre, comparer aux autres options.

TypeScript a quant à lui tous les avantages de Kotlin, grâce au typage fort, et profite aussi de la documentation et des outils javaScript grâce à sa nature de superset. Lorsque l'on développe un produit web surtout pour un développeur full-stack TypeScript est l'idéal.
Ses outils, l'environnement JavaScript, la sécurité des Types et du null, la documentation, sont toutes des bonnes raisons d'utiliser TypeScript. Pour toutes ces raisons TypeScript est le language idéal pour le développement Web.

2.1 Front-end

Pour ce qui est de l'interface utilisateur (le front-end) j'ai décidé d'utiliser Angular car ce framework utilise TypeScript nativement tout en supportant aussi les PWA. Angular étant un framework orienté, il m'oblige aussi à suivre une architecture propre qui permet de rendre independents les différents composants du projet .

2.1.1 Gestion des données

Pour conserver les données hors ligne et facilité l'accès à des données à jour lors d'une connexion j'ai décidé d'utiliser NgRx. Ce dernier est un framework de gestion de données qui simplifie l'accès et le caching de données

2.1.2 Gestionnaire des requêtes

Lorsque l'on veut interagir avec une API graphql il est préférable d'utiliser un client.

Il existe une multitude de clients GraphQl mais le plus populaire et le mieux documenté reste Apollo. De plus il existe un wrapper pour le client Apollo qui l'intègre dans Angular nommé Apollo Angular. Je vais donc utiliser cette librairie pour récupérer mes données au pres de mon serveur .

2.1.3 librairie CSS

Utiliser une librairie de composants permet de simplifier la création de l'interface grâce à des blocks de construction appelé composants. Angular permet d'utiliser des librairies de composants avec leur propre logique, structure, et CSS. Ces librairies fournissent aussi un ensemble de classe CSS pour rendre nos propres composants jolis.
De plus elles permettent de rendre notre application responsive sans devoir y penser pour chaque composant. Il existe plusieurs librairies majeures, les principales sont:

- Angular Material
- PrimeNg
- NgBootstrap

Angular Material est la librairie officielle, elle est d'origine Google, offre beaucoup de composants différents en plus de plusieurs composants de layout qui permettent de d'organiser d'autres composants sur la page.

PrimeNg offre aussi un grand nombre de composants, mais n'offre pas de layout par contre, PrimeNg permet d'utiliser un autre thème que Material.

NgBootstrap est un simple wrapper pour Bootstrap css et propose tous les composants de bootstrap. Vu que mon objectif est de créer une PWA qui sera installable sur Android, utiliser un thème similaire à ceux des applications Google intégrera mieux l'application.

2.2 Back-end

Le serveur back-end doit être capable de mettre à disposition les données nécessaires au fonctionnement de l'application. Cependant, le back-end sera aussi responsable des calculs d'équilibrage des dépenses, des matching de covoiturage et des trajets.

Connaissant déjà Express.js et Spring (Kotlin) je me suis naturellement tourné vers ces derniers, néanmoins je désirais utiliser du TypeScript pour ce projet. Express.js est compatible avec TypeScript mais Spring ne l'est pas. Express.js est leger, minimaliste et propose une bonne documentation, Spring est généraliste propose des outils de génération de code et utilise un système d'injection de dépendance. Ce qu'il me fallait était une hybride entre Spring et Express.js. Après quelque recherches, j'ai trouvé un framework appelé Nest.js, ce dernier supporte nativement TypeScript, propose une documentation excellente, propose un CLI permettant de générer du code boiler-plate, reste simple et leger, Utilise de l'injection de dépendance et est très utilisé et apprécié par d'autres développeurs. J'ai donc décidé d'utiliser Nest.Js pour ce projet.

2.2.1 ORM

Nest.Js (que je vais appeler Nest à partir de maintenant) n'est pas un framework batteries included comme le sont Spring ou Laravel, Il m'a alors fallu choisir aussi un ORM pour interfacer avec ma base de données. Dans la Documentation de Nest, il y a des exemples de configuration avec les ORM les plus utilisés avec Nest et TypeScrip. Ces derniers sont Sequelize, TypeOrm ainsi que Prisma. Ayant eu une experience avec Sequelize plutôt mauvaise lors de mon projet de dev3, j'ai décidé de regarder du côté de prisma.

Prisma est une ORM un peu particulière dans le sens où il propose d'utiliser un schema prisma pour définir ses entités au lieu d'utiliser du code TypeScript. Cella me paraissait une très bonne chose vu que prisma générait alors lui-même les différents DTO nécessaires. Il s'est cependant avéré qu'à l'utilisation prisma n'est pas l'idéale surtout lorsque l'on désire le coupler à une API graphQl.

À cause de son schema particulier, je devais définir le forma de mes entités à 2 endroits et tout de même créer manuellement mes DTO. Sans compter que les opérations de CRUD dans les entités étaient rendues très complexe dû aux liens entre elles et que Prisma ne permet pas d'utiliser des identifiants.

J'ai donc décidé de jeter un œil à TypeOrm, celui-ci propose de définir les entités via des annotations TypeScript. Ce qui me permet d'utiliser une seule classe pour définir mon entité d'ORM et de graphQl de plus TypeOrm est mieux documenté grâce au support d'une communauté plus large.

2.2.2 L'API back end

Il existe plusieurs types d'API très utilisées pour le Back-end mais les plus populaires sont le REST et le SOAP. SOAP est de moins en moins utilisé dû à la complexité intrinsèque du xml et à sa lourdeur.

Il nous reste alors REST qui est une bonne solution, mais nécessite pour presque chaque requête de récupérer des données qui ne seront pas toujours consommées par le client. Avec Rest lorsque l'on récupère une entité complete tous les champs sont renvoyés. C'est pour cela que graphQl a été crée, non seulement graphQL utilise un typage fort via un schema défini et exposé par le serveur, mais en plus graphql permet de ne récupérer que les champs nécessaires au client grace au GQL .

2.2.3 La base de donnée

Ma base de donnée devait me permettre 2 choses, stocker des données géographiques et faire des operations sur ces données. Le seul choix qui s'offrait à moi compatible avec TypeOrm était Postgres avec une extension nommée POSTGIS. Postgres est une base de donnée très utiliser en production, est très performante et stable.

Structure de donnée Pour stocker toutes les données nécessaires au fonctionnement de l'application, il nous faut une structure de donnée cohérente. Voici un schema1 de la base de donnée et de sa structure:

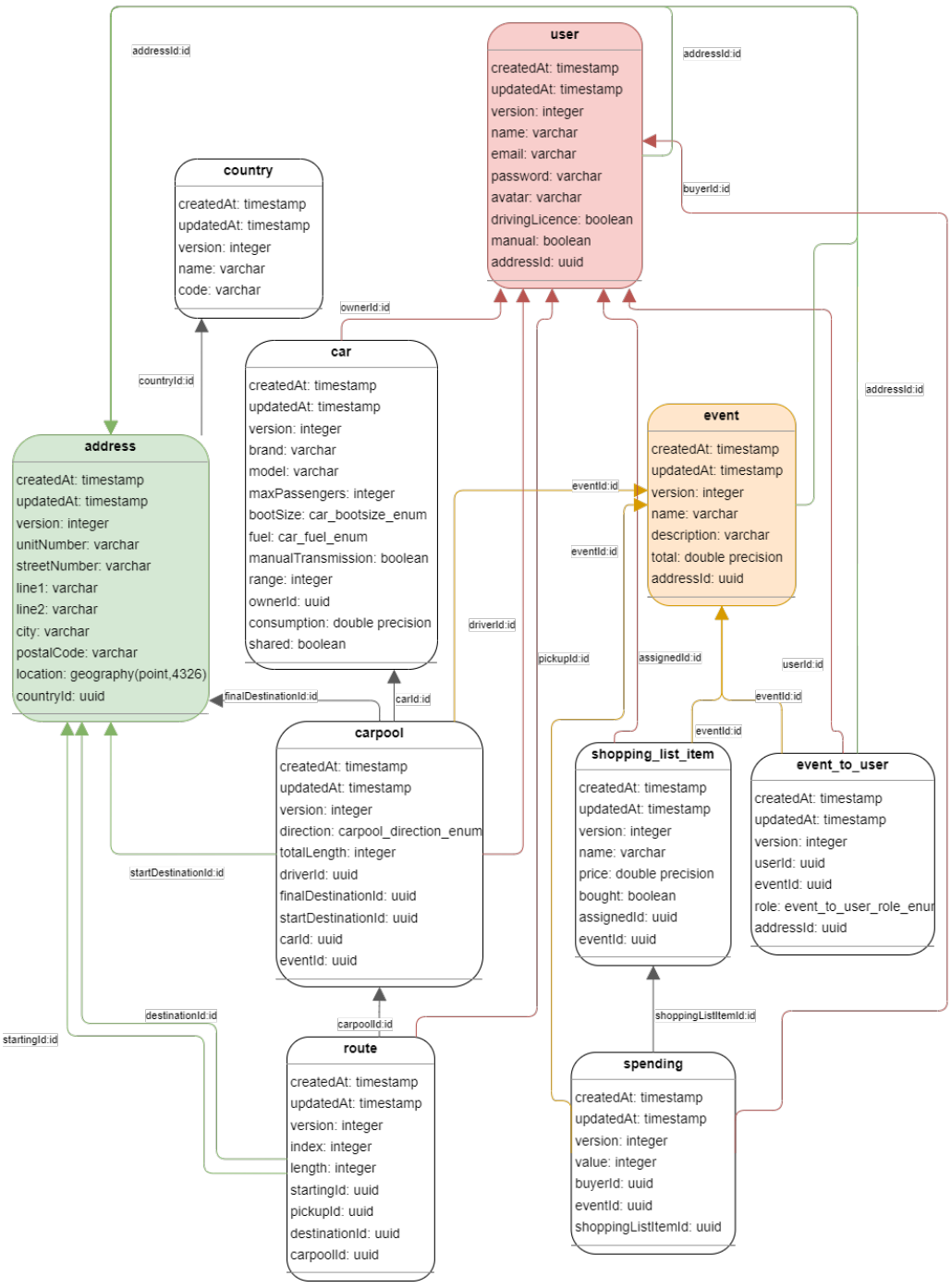

Figure 1: Architecture de la base de donnée