Pour notre projet de base de données polyglotte, nous avons choisi MongoDB, PostgreSQL et Neo4j. Voici notre stack de bases de données sur cette image, mais elle est utilisée uniquement pour les tests en local.  
Finalement, depuis hier, pour notre projet final en microservices, nous avons décidé de déployer les trois bases sur le cloud :

* pour PostgreSQL, nous avons choisi Supabase,
* pour MongoDB, nous continuons à utiliser MongoDB Atlas,
* et pour Neo4j, nous l’hébergerons sur Neo4j Aura.

On a choisi ces trois type de BDD parce que les besoins des chaque service.   
  
**MongoDB c’est pour gérer les infos de hébergement, point d’intérêt ,activité et ville. Parque que on a besoin une très haut flexibilité pour ce type de données, ce type de info contient des données hiérarchique et semi-structuré. Et il y a trops de champs sont null pour quelques record de quelles collection.**

**Et dans l’avenir on peut ajouter des nouvelles champs pour chaque collections facilement avec mongoDB.**Utilise **PostgreSQL** pour créer et gérer les voyage jour par jour. On le utilise parce que ce type de données a une structure fixé, il ne change pas et il y a peut de chance que il existe de champs qui sont null

**Neo4j** pour modéliser le réseau routier en graphe : les villes et points d’intérêt comme nœuds, les trajets comme arêtes. On le utilise parce que Neo4j et naturellement pour le graphe, surtout pour calculer la distance.

Ces trois services communiquent entre eux: par exemple, trip-api stocke seulement les ID d’un hébergement ou d’une activité, puis interroge info-api pour obtenir les détails à afficher.

Au tout début, on utilise **embedding** partiel.  
il n’existe pas de clé étrangère dans mongodb, du coup on utilise le référence, regarde les mots rouge, l’ID de villes dans poi et hebergement， l'ID de POI d’activité dans poi.

Et à part de la, Tout d’abord, on utilise embeding pour des données lire souvent mais mise à jour rarement,par exemple , les villes contienne des infos de ses herbergements et poi

Mais on trouve qu’un problème de consistence pour l’information intégrée Par exemple, quand on mise à jour le nom de poi dans poi, on n’est pas peut pas garantir le nom de poi dans ville est aussi mise à jour.

On utilise triggers dans atlas cloud pour synchroniser tous les modifications d’info embeded.

Du coup, au tout début, On a créer 6 triggers pour tous les informations embeded.  
  
Mais depuis cette semaine, on trouve que la quantité de données n’est pas très grande, donc c’est pas très efficace pour utiliser les infos embededs, depuis le lundi, on supprime tou les infos embededs et grandre des référence utilisé comme clé étrangère.Pour rendre la logique ne notre application plus simple.

Et pour garantir l’intégrité des données, on a utilisé validators pour créer les collection,c’est comme les constraints dans le SQL.  
Voir l’image : ici c’est tous les trucs que on a pour garantir le bon format de données. Par exemple dans le geoinfo, latitude doit entre -90 et 90. Le prix pour une activité pour enfant et adult doit suppèrieur etc.