**1. Architecture générale :**

* **Microservices** : Deux microservices qui communiquent entre eux via des API REST. Chacun a sa propre responsabilité, par exemple :
  + **Service de gestion des budgets** : Traite la création, la mise à jour, et la suppression de budgets.
  + **Service de gestion des transactions** : Gère les transactions financières associées aux budgets (dépenses, revenus).
* **Front-end Angular** : Interface utilisateur pour interagir avec les deux services.
* **PostgreSQL** : Base de données relationnelle pour stocker les informations des deux microservices.
* **Docker** : Conteneurisation des services pour une exécution isolée.
* **Kubernetes** : Orchestration des conteneurs pour déployer les services de manière évolutive.
* **Jenkins** : Automatisation des pipelines CI/CD pour déployer et tester les services.

**2. Découpage en microservices :**

* **Microservice 1 : Service de gestion des budgets**
  + Expose des API REST pour créer, consulter, modifier et supprimer des budgets.
  + Utilise une base de données PostgreSQL pour stocker les budgets.
  + Chaque budget est lié à un utilisateur et contient des informations comme le montant total, le montant alloué, etc.
* **Microservice 2 : Service de gestion des transactions**
  + Expose des API REST pour gérer les transactions financières (entrées et sorties d'argent).
  + Ces transactions sont liées à un budget et peuvent influencer son solde.
* **API Gateway ou communication interservices** : Les deux services doivent pouvoir interagir, soit directement via des appels HTTP, soit via un message broker comme Kafka (facultatif dans un premier temps).

**3. Base de données PostgreSQL :**

* Chaque microservice doit gérer sa propre base de données, mais les schémas doivent être bien définis. Le service de budget peut avoir une table budgets, et le service de transaction une table transactions.
* Relations à considérer : Une transaction est liée à un budget (clé étrangère).

**4. Back-end avec Spring Boot :**

* **Technologies à utiliser** :
  + **Spring Boot** pour le développement des microservices.
  + **Spring Data JPA** pour interagir avec la base de données PostgreSQL.
  + **Spring Security** (si nécessaire) pour sécuriser les API.
  + **Spring Cloud** pour gérer les communications interservices (facultatif au début).
* Chaque service sera un projet Spring Boot distinct, avec des configurations spécifiques.

**5. Front-end avec Angular :**

* **Vue globale** : L'interface Angular permettra aux utilisateurs de gérer leurs budgets et de suivre leurs transactions.
* **Communication avec les microservices** : Via des services Angular qui effectuent des requêtes HTTP vers les API exposées par Spring Boot.
* **Routing** : Créer un système de navigation avec Angular Router pour gérer les vues (budgets, transactions, etc.).
* **Gestion de l’état** : Utiliser RxJS pour la gestion des flux de données asynchrones (si nécessaire).

**6. Conteneurisation avec Docker :**

* Chaque microservice (backend) et le front-end (Angular) doivent être conteneurisés individuellement.
* Créer un fichier Dockerfile pour chaque composant :
  + **Microservices** : Un Dockerfile pour chaque microservice Spring Boot.
  + **Front-end Angular** : Un Dockerfile pour empaqueter l'application Angular dans un serveur web comme NGINX.
  + **Base de données PostgreSQL** : Utiliser une image officielle de PostgreSQL depuis Docker Hub.
* **Docker Compose** : Pour démarrer les conteneurs ensemble lors du développement local.

**7. Déploiement avec Kubernetes :**

* **Kubernetes Pods** : Chaque microservice sera déployé dans un pod Kubernetes. Chaque pod contiendra un conteneur Docker.
* **Services** : Créer des services Kubernetes pour exposer les microservices et l'application front-end.
* **Ingress** : Configurer un ingress controller pour gérer les accès externes (facultatif au début).
* **Secrets et ConfigMaps** : Utiliser les ConfigMaps pour les variables d'environnement et Secrets pour les informations sensibles (comme les mots de passe PostgreSQL).

**8. Intégration continue et déploiement continu (CI/CD) avec Jenkins :**

* **Pipelines Jenkins** :
  + Créer un pipeline pour chaque microservice et le front-end. Chaque pipeline doit :
    1. Construire le projet.
    2. Effectuer les tests unitaires (JUnit pour Spring Boot, Karma/Jasmine pour Angular).
    3. Créer l'image Docker.
    4. Déployer l'image sur Kubernetes.
* **Tests automatisés** : Inclure des tests unitaires pour chaque composant.

**9. Méthodes à utiliser :**

* **Spring Boot** :
  + @RestController pour exposer les API REST.
  + @Entity pour définir les entités de la base de données.
  + @Service pour la logique métier.
  + @Repository pour l’interaction avec PostgreSQL via JPA.
* **Angular** :
  + HttpClient pour effectuer les appels API.
  + RouterModule pour la gestion du routing.
  + FormsModule pour les formulaires interactifs.
* **Docker & Kubernetes** :
  + Créer un Dockerfile pour chaque service.
  + Configurer les fichiers yaml pour déployer sur Kubernetes (deployment, service, etc.).

**Prochaine étape :**

Une fois cette analyse validée, nous pourrons passer à la conception des schémas de base de données et à la création des microservices individuellement. Qu'en penses-tu ?