Cahier des charges technique

1. Introduction

- **Nom du projet** : NewChef
- **Description** : Ce document détaille les spécifications techniques pour la mise en œuvre de l'application de gestion de recettes favorites utilisant Firebase. Le projet comprend une API RESTful sécurisée avec Firebase Authentication et Firestore pour la gestion des données.

2. Architecture du Système

- **Architecture de l'application** :
    - **Firebase Authentication** : Pour gérer l'inscription, la connexion et la déconnexion des utilisateurs.
    - **Firestore** : Pour le stockage et la gestion des données des recettes favorites.
    - **Firebase Functions** : Pour implémenter la logique de l'API RESTful.

3. Technologies Utilisées

- **Langage** : JavaScript (Node.js)
- **Services Cloud** : Firebase (Authentication, Firestore, Functions)
- **Framework de Tests** : Jest
- **Documentation API** : Swagger / OpenAPI

4. Endpoints de l'API

Authentification

- **Inscription**
    - **Méthode** : POST
    - **Endpoint** : /signup

Corps de la requête :
json
Copier le code

```
{
  "email": "string",
  "password": "string"
}
```

    -
    - **Réponses** :
        - 201 : Compte utilisateur créé
        - 400 : Données invalides
        - 500 : Erreur serveur
- **Connexion**
    - **Méthode** : POST
    - **Endpoint** : /login

Corps de la requête :
json
Copier le code

```
{
  "email": "string",
  "password": "string"
}
```

    -
    - **Réponses** :
        - 200 : Connexion réussie (JWT retourné)
        - 400 : Identifiants invalides
        - 500 : Erreur serveur

Gestion des Recettes Favorites

- **Consulter les recettes favorites**
    - **Méthode** : GET
    - **Endpoint** : /getFavoriteRecipes
    - **Paramètres de requête** :
        - `userId` : string (requis)
    - **Sécurité** : Token JWT
    - **Réponses** :
        - 200 : Liste des recettes favorites
        - 400 : userId manquant
        - 404 : Aucune recette trouvée
        - 500 : Erreur serveur
- **Ajouter une recette favorite**
    - **Méthode** : POST
    - **Endpoint** : /addFavoriteRecipe

Corps de la requête :
json
Copier le code

```
{
```

```
  "userId": "string",
  "recipe": {
    "recipeId": "number",
    "title": "string",
    "ingredients": ["string"],
    "instructions": "string"
  }
}
```

        -
        - **Sécurité** : Token JWT
        - **Réponses** :
            - 200 : Recette ajoutée
            - 400 : Données manquantes
            - 500 : Erreur serveur
- **Mettre à jour une recette favorite**
    - **Méthode** : PUT
    - **Endpoint** : /updateFavoriteRecipe

Corps de la requête :
json
Copier le code

```
{
  "userId": "string",
  "recipeId": "number",
  "updateFields": {
    "title": "string",
    "ingredients": ["string"],
    "instructions": "string"
  }
}
```

        -
        - **Sécurité** : Token JWT
        - **Réponses** :
            - 200 : Recette mise à jour
            - 400 : Données manquantes
            - 404 : Recette non trouvée
            - 500 : Erreur serveur
- **Supprimer une recette favorite**
    - **Méthode** : DELETE
    - **Endpoint** : /deleteFavoriteRecipe

- **Paramètres de requête** :
  - `userId` : string (requis)
  - `recipeId` : number (requis)
- **Sécurité** : Token JWT
- **Réponses** :
  - 200 : Recette supprimée
  - 400 : Données manquantes
  - 404 : Recette non trouvée
  - 500 : Erreur serveur

5. Modèles de Données

Utilisateur
json
Copier le code

```
{
  "uid": "string",
  "email": "string"
}
```

-

Recette
json
Copier le code

```
{
  "recipeId": "number",
  "title": "string",
  "ingredients": ["string"],
  "instructions": "string"
}
```

-

Collection de Recettes Favorites
json
Copier le code

```
{
  "userId": "string",
  "recipes": [
    {
      "recipeId": "number",
      "title": "string",
```

```
      "ingredients": ["string"],
      "instructions": "string"
    }
  ]
}
```

-

6. Sécurité

- **Token JWT** : Utilisation de tokens JWT pour sécuriser les endpoints d'API. Les utilisateurs doivent fournir un token JWT valide dans le header Authorization pour accéder aux endpoints sécurisés.
- **Validation des Entrées** : Validation rigoureuse des données d'entrée pour chaque endpoint pour éviter les injections et les attaques de type XSS.

7. Tests

- **Tests Unitaires** : Utilisation de Jest pour les tests unitaires des fonctions Firebase.
- **Tests d'Intégration** : Tests des interactions entre les différentes parties de l'application (Firebase Authentication, Firestore, Functions).
- **Tests de Performance** : Assurer que l'application supporte une charge utilisateur importante sans dégradation de la performance.

8. Déploiement

- **Configuration Firebase** : Déploiement des fonctions Firebase et des règles de sécurité Firestore.
- **Scripts de Déploiement** : Scripts pour automatiser le déploiement des fonctions Firebase et la configuration de la base de données.
- **Environnement de Production** : Configuration d'un environnement Firebase pour la production avec des règles de sécurité strictes.

9. Maintenance et Évolutivité

- **Documentation** : Documentation complète du code et des endpoints API pour faciliter la maintenance et les futures évolutions.
- **Scalabilité** : Assurer que l'architecture choisie permet de facilement scaler horizontalement en cas d'augmentation du nombre d'utilisateurs.

10. Annexes

- **Exemples de Requêtes et Réponses** : Exemples détaillés des requêtes et réponses pour chaque endpoint API.