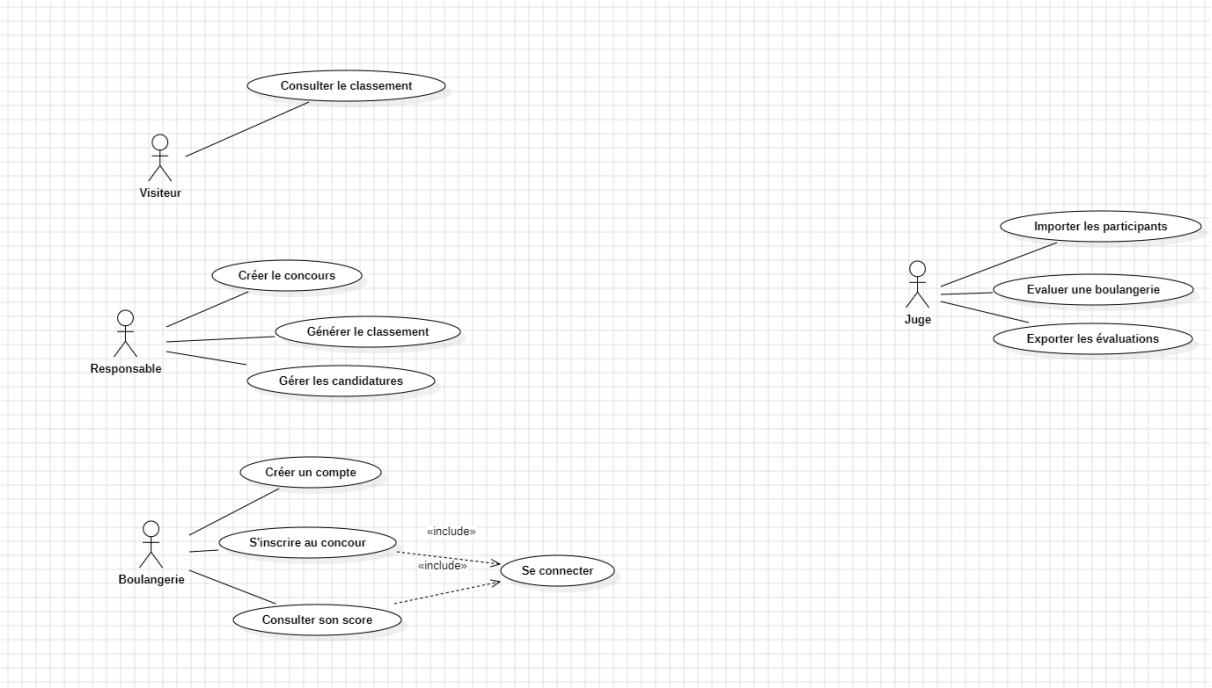
Documentation
La Gourmetise

CONTEXTE

LA GOURMETISE

L’entreprise La Gourmetise, a été fondée en 2010 par un groupe de passionnés de la gastronomie et de la boulangerie. Depuis sa création, La Gourmetise s'est engagée à promouvoir l'excellence dans le domaine de la boulangerie et de la pâtisserie. Sa mission est de célébrer l'art de la boulangerie en mettant en lumière les artisans talentueux qui créent des produits exceptionnels.

Schéma d’architecture de la solution



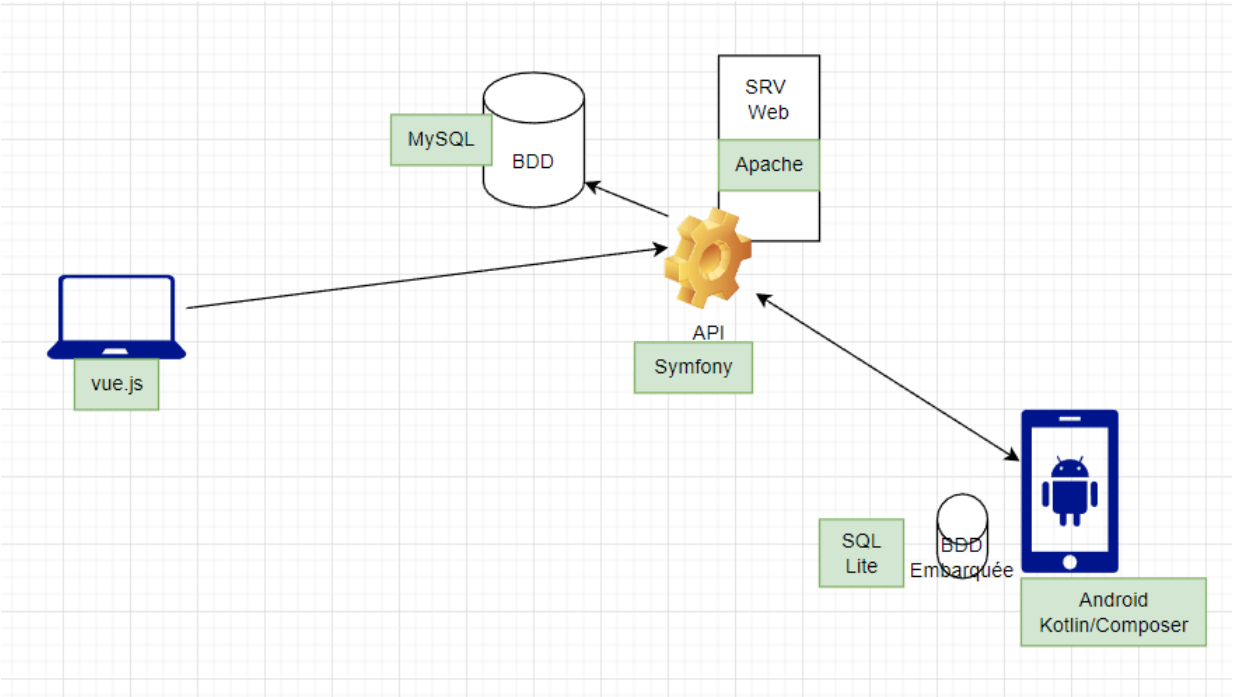
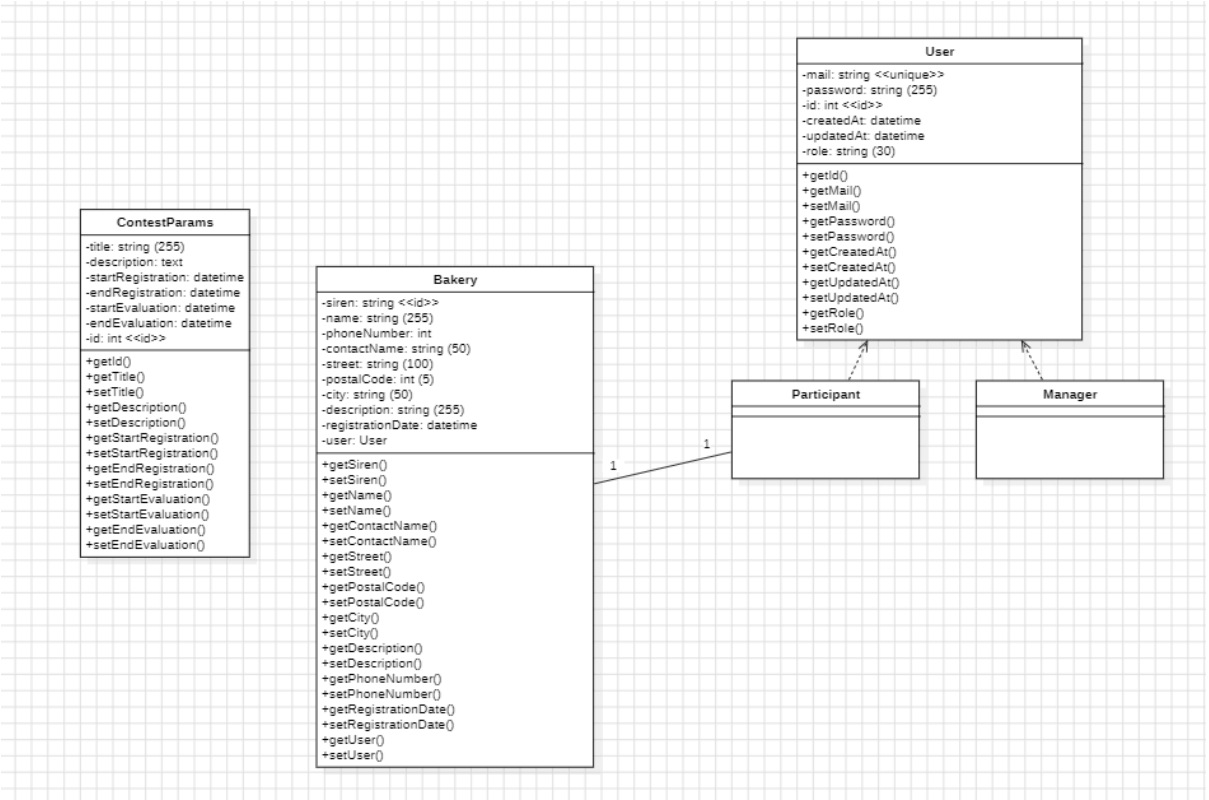


Diagramme de classe de la base de donnée



BDD et Versioning

1. BDD

- Gestion utilisateurs → permissions limitées
- Gestion des logs
- Contrôle des données → Trigger
- Laragon → (plus tard, déploiement LAMP Ferme Serveur)

2. Versioning

Solution de gestion de versions :

- Github

Documentation de l'API

1. Introduction

Cette API permet de gérer l'ensemble du processus d'inscription des boulangeries au concours, de la création des boulangeries à la gestion des utilisateurs, en passant par la configuration des paramètres du concours. Elle est composée de plusieurs contrôleurs API permettant différentes actions, telles que la création, la lecture, la mise à jour et la suppression des entités concernées.

---

2. Architecture de l'API

L'API repose sur une architecture RESTful. Elle expose les ressources principales suivantes :

- **Boulangeries** : Représente les boulangeries inscrites au concours.
- **Concours** : Gère les paramètres du concours (dates de début et de fin des inscriptions).
- **Utilisateurs** : Gère les utilisateurs inscrits, y compris leur rôle et leurs informations personnelles.

---

3. Description des Routes API

3.1 API Boulangerie (APIBakeryController)

POST /api/bakery

Permet d’inscrire une nouvelle boulangerie au concours.

- **Validation** : Le serveur valide les champs nécessaires : numéro de Siren, nom, adresse, etc.
- **Réponse** : Si la boulangerie est créée avec succès, renvoie un code HTTP 201 et un message de confirmation. Si une erreur est rencontrée (par exemple, le Siren est déjà pris ou les informations sont manquantes), renvoie une erreur HTTP 400.

GET /api/bakery

Permet de récupérer la liste de toutes les boulangeries inscrites.

Réponse :
json
Copier le code

```
[
    {
        "id": 1,
        "name": "Nom de la boulangerie",
        "contactName": "Nom du contact",
        "street": "Adresse",
        "postalCode": "Code postal",
        "city": "Ville",
        "phoneNumber": "Numéro de téléphone",
        "siren": "Numéro de Siren",
        "registrationDate": "Date d'inscription"
    }
]
```

-

3.2 API Paramètres du Concours (APIContestParamsController)

GET /api/contestParams

Permet de récupérer les paramètres du concours.

Réponse :
json
Copier le code

```
{
    "id": 1,
    "startRegistration": "Date de début des inscriptions",
    "endRegistration": "Date de fin des inscriptions",
    "otherParams": "Autres paramètres"
}
```

-

POST /api/contestParams

Permet de créer de nouveaux paramètres pour le concours.

- **Réponse** : Confirmation de la création avec un code HTTP 201 ou une erreur en cas de conflit (si les paramètres existent déjà).

PUT/PATCH /api/contestParams

Permet de mettre à jour les paramètres existants du concours.

- **Réponse** : Confirmation de la mise à jour des paramètres du concours.

DELETE /api/contestParams

Permet de supprimer les paramètres du concours existants.

- **Réponse** : Confirmation de la suppression avec un code HTTP 204 (No Content).

3.3 API Utilisateurs (APIUserController)

POST /api/users

Permet de créer un nouvel utilisateur.

Entrée (Body) :
json
Copier le code

```
{
    "name": "Nom de l'utilisateur",
    "email": "email@exemple.com",
    "role": "participant"  // "participant" ou "organizer"
}
```

- **Réponse** : Confirmation de la création de l'utilisateur avec un code HTTP 201, ou erreur si l'utilisateur existe déjà (code HTTP 400).

---

4. Sécurisation de l'API

4.1 Validation des Entrées

Chaque endpoint de l'API vérifie la validité des données reçues. Les entrées sont validées contre les règles suivantes :

- **Champs obligatoires** : Certains champs (comme le numéro de Siren, l'email de l'utilisateur, etc.) sont requis pour l'inscription d'une boulangerie ou la création d'un utilisateur.
- **Format des données** : L'API vérifie que les données (comme l'email et les numéros de téléphone) sont au bon format.
- **Conflits** : Les doublons de Siren et d'utilisateur sont vérifiés avant l'insertion dans la base de données.

4.2 Authentification et Autorisation

L'API repose sur un système d'authentification via des tokens (JWT par exemple). Elle vérifie les rôles des utilisateurs avant d'effectuer certaines actions :

- **Rôle "participant"** : Permet de créer une boulangerie.
- **Rôle "organizer"** : Permet de gérer les paramètres du concours.

4.3 Protection contre les Attaques

Des protections sont mises en place contre les attaques les plus courantes :

- **Protection contre les attaques CSRF** : Utilisation de tokens pour valider les requêtes.
- **Utilisation de HTTPS** : Toutes les communications sont sécurisées via HTTPS pour éviter les interceptions de données.

---

5. Gestion des Erreurs

L'API renvoie des messages d'erreur détaillés en cas de problème. Voici quelques exemples d'erreurs courantes :

- **400 Bad Request** : Erreur dans les données envoyées (champs manquants, format invalide, etc.).
    - Exemple : `{"error": "Siren number is required"}`
- **404 Not Found** : Ressource non trouvée (par exemple, les paramètres du concours n'existent pas).
    - Exemple : `{"error": "ContestParams not exist"}`
- **409 Conflict** : Conflit avec une ressource existante (par exemple, un doublon de Siren).
    - Exemple : `{"error": "Ce numéro de Siren est déjà prit"}`

Documentation de l'Application Web

1. Introduction

L'application web permet aux utilisateurs de participer au concours Gourmetise, organiser des inscriptions pour les boulangeries participantes, afficher les résultats du concours, et consulter les dates importantes relatives au concours. L'application est construite avec **Vue.js** et utilise le framework **Vuetify** pour l'interface utilisateur.

---

2. Composants de l'Application

A. Page d'Accueil - Concours de la Meilleure Boulangerie

- **Composant principal** : `<template>`
  - **But** : Afficher l'introduction au concours, y compris une explication générale sur le concours de la meilleure boulangerie.
  - **Contenu** :
    - Un logo de l'application (Gourmetise)
    - Un titre principal : "Bienvenue au Concours de la Meilleure Boulangerie"
    - Un paragraphe descriptif expliquant le concours et invitant les utilisateurs à s'inscrire.
  - **Cardes interactives** :
    - **Participez en tant que boulanger** : Permet aux utilisateurs de s'inscrire au concours.
    - **Voir les résultats** : Permet aux utilisateurs de consulter les résultats des concours précédents.
    - **Consulter les données du concours** : Permet aux utilisateurs de voir les dates importantes du concours.

B. Page d'Inscription - Participation au Concours

- **Composant principal** : `v-form`, `v-text-field`, `v-btn`
  - **But** : Permet aux boulangeries de s'inscrire au concours en remplissant un formulaire avec leurs informations.
  - **Champs du formulaire** :
    - **Nom de la boulangerie** : Champ de texte pour le nom de la boulangerie.

        - **Numéro Siren** : Champ de texte pour entrer le numéro Siren (avec validation).
        - **Numéro de téléphone** : Champ de texte pour le téléphone (avec format spécifique).
        - **Nom du contact** : Champ de texte pour le nom de la personne de contact.
        - **Adresse (rue et code postal)** : Champs pour l'adresse physique.
        - **Ville** : Autocomplete pour choisir la ville, avec récupération automatique des villes depuis un code postal.
        - **Description de la boulangerie** : Champ de texte pour une description libre.
        - **Consentement** : Checkbox pour accepter les conditions d'utilisation avant l'inscription.
    - **Validations** :
        - Chaque champ est validé avant que l'utilisateur puisse soumettre son formulaire.
        - Utilisation de **règles de validation** basées sur des expressions régulières et des vérifications de longueur (ex. : format de téléphone et longueur du Siren).
    - **Boutons** :
        - **S'inscrire** : Soumet le formulaire et envoie les données au serveur.
        - **Annuler** : Réinitialise tous les champs du formulaire.

C. Page des Paramètres du Concours - Dates importantes

- **Composant principal** : `<template>` avec des affichages de données
    - **But** : Afficher les dates importantes du concours comme la date de début et de fin des inscriptions.
    - **Fonctionnalité** :
        - Récupération des paramètres du concours via une API externe (`/api/contestParams`).
        - Affichage des informations concernant la période d'inscription du concours.
        - Formattage des dates pour les afficher de manière lisible.

D. Politique de Confidentialité

- **Composant principal** : `<template>` avec du texte
    - **But** : Afficher les détails de la politique de confidentialité concernant la collecte et le traitement des données des participants au concours.
    - **Contenu** :
        - Informations sur la collecte des données personnelles des participants (nom, prénom, adresse, e-mail, etc.).
        - Objectifs du traitement des données : gestion de la participation, publication des résultats, communication avec les participants.
        - Détails sur la sécurité des données et les droits des utilisateurs selon le RGPD (accès, rectification, suppression des données).

        - Un consentement explicite des participants est requis avant toute participation.

---

3. Logique Métier

A. Validation des Champs

- Chaque champ de formulaire est validé avant la soumission. La logique de validation est gérée avec des règles et des expressions régulières.
    - Le numéro de téléphone doit suivre le format français (ex : `06 56 47 36 78`).
    - Le Siren doit contenir entre 9 et 14 chiffres.
    - Le code postal doit être de 5 chiffres.
    - Les autres champs sont vérifiés pour s'assurer qu'ils ne sont pas vides avant l'envoi.

B. Envoi des Données

- Lors de l'inscription, les données de la boulangerie sont envoyées au backend via une requête **POST** à l'API `/api/bakery`.
- Si l'envoi réussit, un message de succès est affiché.
- En cas d'erreur, un message d'erreur est affiché.

C. Récupération des Villes

- Lors de la saisie du code postal, une API externe (`https://geo.api.gouv.fr/communes?codePostal=`) est appelée pour récupérer la liste des villes associées au code postal entré. Les villes sont affichées sous forme de liste déroulante dans un champ `v-autocomplete`.

---

4. Gestion des Erreurs et Notifications

- **Toast Notifications** : Lors de l'envoi du formulaire, des notifications Toast sont utilisées pour informer l'utilisateur de la réussite ou de l'échec de l'opération.
    - **Succès** : "Données envoyées !"
    - **Erreur** : "Erreur lors de l'envoi des données : [message d'erreur]"

---

5. Comportement du Bouton d'Inscription

Le bouton "S'inscrire" est désactivé tant que :

- Un ou plusieurs champs du formulaire sont invalides.
- Le consentement de l'utilisateur n'est pas donné.

Le bouton est activé uniquement lorsque toutes les validations sont passées avec succès et que l'utilisateur a accepté les conditions d'utilisation.

---

6. Styles et Mise en Page

Les composants Vuetify sont utilisés pour la mise en page responsive, notamment les `v-container`, `v-row`, et `v-col` pour organiser le contenu en colonnes et lignes.

Le style CSS utilisé dans l'application inclut une mise en page centrée avec des espacements généreux pour les sections du formulaire et de contenu. L'interface est conçue pour être lisible et intuitive.

---

7. Sécurité et Confidentialité

- Toutes les données sensibles, telles que les informations personnelles des participants, sont transmises via des requêtes sécurisées (HTTPS).
- L'application nécessite le consentement explicite des utilisateurs avant de procéder à la soumission de données.

Documentation de l'application mobile

L'application mobile est une interface Android qui permet de gérer et d'afficher des informations sur les boulangeries participantes à un concours, et d'importer ces informations via une API.

1. Classes principales

A. MainActivity :

C'est la classe principale de l'application qui contient l'écran d'accueil. Elle gère la récupération des données depuis l'API et l'affichage de la liste des boulangeries.

- **Fonctionnalités principales** :
  - Initialisation des données via l'API (`bakery` et `contest_params`).
  - Affichage d'une liste de boulangeries avec une interface de navigation.
  - Un bouton qui permet de récupérer les données depuis l'API si le concours est ouvert.
- **Composants utilisés** :
  - `Scaffold` : Pour structurer l'interface avec une `TopAppBar` et une `BottomAppBar`.
  - `Button` : Un bouton conditionnel qui devient actif si les données des boulangeries ne sont pas présentes.
  - `OkHttpClient` : Utilisé pour envoyer des requêtes HTTP vers l'API pour récupérer les données des boulangeries et des paramètres du concours.
  - `Toast` : Affiche des messages de succès ou d'erreur pendant l'exécution des requêtes.

B. BakeryList :

Affiche la liste des boulangeries disponibles après l'importation des données.

- **Fonctionnalités principales** :
  - Affiche la liste des boulangeries dans un format `LazyColumn`.
  - Permet d'étendre chaque carte pour afficher des détails supplémentaires sur chaque boulangerie.
- **Composants utilisés** :

- `LazyColumn` : Affiche une liste défilante d'éléments.
- `Card` : Utilisée pour chaque élément de la liste, qui peut être cliqué pour afficher plus de détails.
- `Text`, `Image` : Affiche les informations sur chaque boulangerie (nom, adresse, téléphone, etc.).

C. Classes de gestion de la base de données :

- **BakeryDAO** : Classe permettant d'interagir avec la base de données locale SQLite pour gérer les informations des boulangeries.
  - **Méthodes principales** :
    - `supprimerToutesLesBakery()` : Supprime toutes les boulangeries de la base de données.
    - `ajouterBakery()` : Ajoute une nouvelle boulangerie.
    - `toutesLesBakery()` : Récupère toutes les boulangeries de la base de données.
- **Contest_ParamsDAO** : Gère les informations des paramètres de concours.
  - **Méthodes principales** :
    - `supprimerTousLesContestParams()` : Supprime tous les paramètres du concours.
    - `ajouterContestParams()` : Ajoute les paramètres du concours.
    - `getContestParams()` : Récupère les paramètres du concours depuis la base de données.

D. Modèles de données :

- **Bakery** : Représente une boulangerie avec des attributs comme `name`, `street`, `postalCode`, etc.
- **ContestParams** : Représente les paramètres du concours, incluant `startEvaluation` et `endEvaluation`.

E. Base de données SQLite :

- **BakeryHelper** : Helper pour créer et gérer la base de données SQLite, qui contient les tables `Bakery` et `contest_params`. Les données sont stockées localement et peuvent être modifiées ou récupérées par les DAO correspondants.

---

2. Fonctionnalités détaillées

A. Importation des données via l'API :

- Lorsque l'utilisateur appuie sur le bouton "Voir la liste des boulangeries", une série de requêtes HTTP est effectuée pour récupérer les données de l'API (concours et boulangeries). Si les évaluations sont ouvertes (dépend de `startEvaluation` et `endEvaluation`), les données sont importées et affichées dans la liste.

B. Affichage de la liste des boulangeries :

- Une fois les données récupérées et stockées dans la base de données locale, l'application utilise une `LazyColumn` pour afficher la liste des boulangeries avec la possibilité d'afficher plus de détails sur chaque boulangerie via une `Card` cliquable.

C. Gestion de l'état de la base de données locale :

- L'application permet de supprimer et d'ajouter des boulangeries ainsi que les paramètres du concours via les classes DAO, garantissant que les données sont correctement manipulées avant d'être affichées.

D. Interface utilisateur :

- L'interface est composée de plusieurs écrans, dont un écran d'accueil qui propose un bouton d'importation et un écran secondaire qui affiche la liste des boulangeries avec leurs informations.

---

3. Prérequis

- **API Backend** : L'application nécessite un serveur d'API qui fournit des données sur les boulangeries et les paramètres de concours via les URL `http://10.0.2.2:8000/api/bakery` et `http://10.0.2.2:8000/api/contestParams`.
- **Permissions** : L'application nécessite des permissions d'accès à Internet pour récupérer les données depuis l'API.
- **SQLite** : Une base de données locale est utilisée pour stocker les informations des boulangeries et des paramètres du concours.

---

4. Code de l'application

Le code fourni montre comment récupérer des données via une API, les stocker localement, et afficher les informations dans l'interface utilisateur de manière dynamique en utilisant les outils et composants Android modernes comme `Jetpack Compose`, `Scaffold`, et `LazyColumn`.

---

5. Exemple de requêtes API :

```
// Requête pour récupérer les données de la table bakery
val bakeryRequest = Request.Builder()
    .url("http://10.0.2.2:8000/api/bakery")
    .build()

// Requête pour récupérer les données de la table contest_params
val contestParamsRequest = Request.Builder()
    .url("http://10.0.2.2:8000/api/contestParams")
    .build()
```

---

6. Exemple de gestion de la base de données SQLite :

```
fun supprimerToutesLesBakery() {
    maBase.delete("Bakery", null, null)
}

fun ajouterBakery(uneBoulangerie: Bakery) {
    val v = ContentValues()
    v.put("siren", uneBoulangerie.siren)
    v.put("name", uneBoulangerie.name)
    v.put("street", uneBoulangerie.street)
    v.put("postalCode", uneBoulangerie.postalCode)
    v.put("city", uneBoulangerie.city)
    v.put("description", uneBoulangerie.description)
    v.put("phoneNumber", uneBoulangerie.phoneNumber)
    maBase.insert("Bakery", null, v)
}
```