Projet d’application répartie

DIETRICH Corentin – KETZINGER Tom – MOUGIN Enzo – PEDRON Mathéo

https://github.com/Mape57/DIETRICH_KETZINGER_MOUGIN_PEDRON-programmation_repartie

https://webetu.iutnc.univ-lorraine.fr/www/pedron7u/mapaide

Compte rendu : https://webetu.iutnc.univ-lorraine.fr/www/pedron7u/mapaide/compterendu/

Utilisations de notre projet

Accès définitif

Lors de la réalisation de la SAE, il a été décidé de faire l’acquisition du nom de domaine mapaide.fr.

Ainsi, nous avons pu mettre à profit d'autres enseignements du BUT notamment la virtualisation. En effet, la base donnée de l’IUT n’étant accessible uniquement via VPN, nous avons dû en recréer une. Cela à été réalisé en utilisant un répertoire github permettant la création d’une image docker de base de données oracle permettant ensuite le lancement et le maintien actif de celle-ci.

Nous sortons du contexte du projet demandé dans un premier temps pour pouvoir garder notre projet dans le temps.

En plus de cela, il est hébergé sur un serveur personnel, le rendant donc accessible à tout moment sur ordinateur comme sur téléphone. Effectivement, le responsive du site à été réalisé afin de permettre une utilisation sur l’ensemble des outils possibles.

Accès sur votre machine

Pour pouvoir lancer vous-même le projet, il est nécessaire d’importer les bonnes librairies dans les différents projets :

- http_server : nécessite la librairie ‘json’ ;
- travaux_rmi : nécessite la librairie ‘json’ ;
- restaurant_rmi : nécessite les librairies ‘json’ et ‘ojdbc’

De plus, restaurant_rmi demande l’ajout d’un fichier ‘.conf’ dans le dossier resources. Afin de vous permettre du lancer, le mot de passe à la base de donnée à été simplifié veuillez donc inscrire dans le fichier :

url=jdbc:oracle:thin:@charlemagne.iutnc.univ-lorraine.fr:1521:infodb
user=pedron7u
password=motdepasse

Pour travaux_rmi et restaurant_rmi, si vous n’êtes pas en localhost, modifiez le fichier où se trouve le main, afin de se connecter à l’adresse du http_server.

Finalement, il est nécessaire de compiler les projets, si cela est fait via le terminal le paramètre '-cp' est obligatoire pour indiquer les librairies à utiliser. Avant de lancer les javas, exécuter 'rmiregistry' dans le dossier comportant les fichiers '.class' du http_server (si via intellij, il s'agit de '/out/production/http_server'). Finalement, les trois projets peuvent être lancé (dans l'ordre suivant), sans oublier le paramètre '-cp':

1. http_server
2. travaux_rmi et restaurant_rmi

Fonctionnement du projet

L'application réalisée est composée de plusieurs modules ayant chacun un rôle spécifique :

- http_server : serveur central sur lequel les services RMI vont s'inscrire. Il s'occupe de placer des classes d'enregistrement dans l'annuaire.
- restaurant_rmi : serveur permettant de faire le lien entre la base de donnée et le serveur http. Il s'inscrit auprès du serveur central, permettant de réaliser les diverses méthodes de récupération et d'envoie de donnée sur le restaurant.
- travaux_rmi : serveur permettant d'accéder à des données bloquées par les CORS (ici les incidents de circulation de la ville de Nancy). Ce serveur est rendu générique grâce à un paramètre de query (?url={...}). Il agit tel un proxy. De plus, le projet devant être exécutés sur les ordinateurs de l'IUT mais aussi sur notre propre serveur, nous avons ajouté deux méthodes de fetch :
    - fetch via le proxy de l'IUT
    - fetch directement sur l'URL (lancé si la première a échoué)
- API externes : aucun travail de notre part, il s'agit ici uniquement de comprendre les liens utiles et si la requête demande de passer par le serveur
- client_web : permet d'afficher toutes les informations sur une carte Leaflet en fonction des actions de l'utilisateur, à noter que la création d'un restaurant se fait via Ctrl-clic dans la map sur ordinateur et par simple clique sur mobile.

Plusieurs langages ont été utilisés dans ce projet afin de mettre en œuvre cette application :

- Java : A permis d'implémenter les services RMI, les serveurs http et les relations avec la base de données via JDBC.
- JavaScript : A permis de récupérer les données des API distantes via des fetch pour les utiliser par la suite. Permet également de gérer l'affichage des informations sur la carte et les différents évènements liés à l'utilisateur.
- HTML : A permis de créer le contenu du site web.
- CSS : Gère le style de l'application.
- SQL : A permis de créer la base de données, de la peupler de restaurants et de manipuler ces données par la suite.

Fonctionnement du backend java

http_server

Le service RMI (Remote Method Invocation) permet à un programme Java d'appeler des méthodes situées sur une autre machine. Il s'agit de http_server qui va gérer cela, il regroupe et centralise l'ensemble des accès et requêtes.

Accès HTTP

- /travaux?url=... : initialement uniquement pour la récupération des données de travaux, il a été transformé de manière générique grâce au paramètre 'url'. Le nom '/travaux' serait donc à modifier.ù
- /restaurants :
    - /restaurants : retourne la liste de tout les restaurants avec leur id et coordonnées
    - /restaurants/{id} : retourne les données du restaurant correspondant à l'id
    - /restaurants/{id}/horaires : retourne les horaires d'un restaurant par semaine
    - /restaurants : en requête POST et les paramètres 'nomResto', 'adr', 'coordonnees' et 'note' permet la création d'un restaurant
- /reservation :
    - /reservation?idResto=...&nbConviv=...&date=... : récupérer les heures où il est possible de réserver (date en format 'yyyy-MM-dd')
    - /reservation : en requête POST et les paramètres 'idResto', 'nbConviv', 'date', 'nom', 'prenom' et 'numTel' permet de réserver un table (date au format 'yyyy-MM-dd hh:mm:ss'

Accès RMI

- BinderInterface : interface permettant au object distant de s'enregistrer sur le serveur central implémenté deux fois (restaurant et travaux)
- DataRequesterInterface : interface de l'objet distant qui est enregistré sur le serveur central
- RestaurantDataRequesterInterface : interface spécifique aux requêtes de données des restaurants implémentant DataRequesterInterface
- TravauxDataResquesterInterface : interface spécifique aux requêtes de données des urls protégés par CORS implémentant DataRequesterInterface

Sécurisation des requêtes

L'ensemble des données obtenus par HTTP sont vérifiés autrement une réponse d'erreur est envoyée indiquant s'il manque des paramètres ou si un paramètre est invalide ou encore si le format d'un paramètre est incorrect.

Du côté RMI, les remotes exceptions sont également traités pour retourner une erreur HTTP sans crasher le serveur.

Finalement, c'est l'ensemble des erreurs qui sont traitées pour permettre une meilleure compréhension des problèmes que ce soit du côté backend mais également pour l'interface web.

restaurant_rmi

Accès RMI

Après la récupération du Binder, le restaurant enregistre son implémentation de l'interface RestaurantDataRequesterInterface. Celle-ci permet de réaliser diverse requête :

- getPossibleReservation(idResto, nbConviv, date) : retourne les heures de réservation possible
- postReservation(idResto, nom, prenom, nbConviv, numTel, date) : ajoute une réservation au restaurant en choisissant la plus petite table disponible, et retourne la réservation
- getAllRestaurantPosition() : retourne l'ensemble des restaurants sous la forme {id: …, coordonnees: …}
- getRestaurantById(idResto) : retourne l'ensemble des données d'un restaurant
- getRestaurantHoraires(idResto) : retourne les horaires d'un restaurant
- postRestaurant(nomResto, adr, coordonnees, note) : créer un restaurant et le retourne

Accès JDBC

Afin de réaliser facilement les requêtes auprès de la base de données, nous avons implémenté des actives records. Il s'agit ensuite de réaliser les bonnes conversions de données, pour réaliser les insert et select.

Les erreurs sont également gérer au maximum, pour retourner des données vides si nécessaires, et envoyer une RemoteException en cas d'erreur exceptionnelle.

Sécurisation des requêtes

La sécurisation a été faite au niveau du http_server. Il s'agit ici de gérer les Exceptions et de retourner des JSON vides si nécessaire.

Un maximum de log sont fait pour permettre de suivre les erreurs en temps réels

travaux_rmi

Récupération des données

Fonctionne grâce à une simple méthode getData(url) qui permet de récupérer les données de l'URL et retourner le JSON obtenu. D'abord en essayant le fetch simple puis en faisant fetch via le proxy de l'IUT si cela ne marche pas.

Sécurisation des requêtes

Est-ce responsable d'ainsi contourner la politique de sécurité de votre navigateur ?

Contourner la politique de sécurité de votre navigateur n'est généralement pas recommandé car ces politiques sont en place pour protéger les utilisateurs contre diverses menaces telles que les attaques de type cross-site scripting (XSS) et le vol de données. Utiliser un proxy pour accéder à des données bloquées peut être vu comme une tentative de contourner ces mesures de sécurité. Cependant, dans un contexte de développement ou d'un projet académique où les données sont nécessaires pour le bon fonctionnement de l'application, et où les accès sont contrôlés et sécurisés, cette pratique peut être tolérée à des fins d'apprentissage et de démonstration.

1) Dossier Affichage

1. aff_meteo.js

Ce fichier gère l'affichage des informations météorologiques. Il utilise le moteur de template Handlebars pour générer le HTML correspondant à la météo. Voici quelques fonctionnalités clés :

- **titleToPicto** : Une correspondance entre les conditions météorologiques et les icônes.
- **displayMeteo** : Fonction principale pour afficher les données météorologiques actuelles, les heures et les jours suivants.
- **setupDate, getCurrentDay, getNextHours, getNextDays** : Fonctions auxiliaires pour formater et préparer les données météorologiques à afficher.

2. aff_popup.js

Ce fichier est responsable de la création et de la gestion des contenus des popups sur la carte. Il utilise également Handlebars pour les templates. Voici les fonctionnalités principales :

- **createVeloPopupContent, createIncidentPopupContent, createSchoolPopupContent** : Fonctions pour créer le contenu des popups pour les stations de vélo, les incidents et les écoles respectivement.
- **createRestaurantPopupContent** : Gère le clic sur un marqueur de restaurant pour afficher les détails du restaurant.
- **createDateSelectionArea** : Génère la zone de sélection de date pour les réservations.

3. aff_restaurant.js

Ce fichier gère l'affichage des détails des restaurants et la gestion des réservations. Les fonctionnalités clés incluent :

- **displayRestaurantDetails** : Afficherd les détails d'un restaurant, y compris sa note, ses horaires d'ouverture, et permet de faire une réservation.

- **initDateValue, nextDateValue, previousDateValue, displayDateSelectionArea, displayDateAndAnimate** : Gèrent la sélection des dates disponibles pour les réservations.

- **generateStars** : Génère le HTML pour afficher la note sous forme d'étoiles.

4. `map.js`

Ce fichier initialise la carte avec Leaflet et ajoute des marqueurs avec des popups. Les fonctionnalités principales sont :

- **initMap** : Initialise la carte centrée sur Nancy avec les tuiles d'OpenStreetMap.

- **addMarkersToMap** : Ajoute des marqueurs à la carte avec des popups personnalisés.

Ces fichiers travaillent ensemble pour afficher une carte interactive avec des popups pour différents types de points d'intérêt (météo, restaurants, incidents, etc.), et pour gérer les réservations dans les restaurants.

2) Dossier Tools

1. `config.js`

Ce fichier contient des configurations de base utilisées par d'autres parties de l'application. Voici les principales constantes définies :

- **baseURL** : L'URL de base pour les requêtes API, ici définie comme http://localhost:8080.

- **semaine** : Un tableau contenant les noms des jours de la semaine en français, de "Dimanche" à "Samedi".

2. `mapIcons.js`

Ce fichier gère la création des icônes pour les marqueurs sur la carte en utilisant Leaflet. Voici les différentes icônes définies :

- **redIcon, blueIcon, yellowIcon, greenIcon** : Ces variables exportent des icônes de différentes couleurs (rouge, bleu, jaune, vert) avec leurs propriétés respectives comme l'URL de l'icône et de son ombre, la taille de l'icône, le point d'ancrage de l'icône et le point d'ancrage de la popup par rapport à l'icône.

3. `RestaurantManager.js`

Ce fichier contient une classe pour gérer l'ajout et l'affichage des restaurants sur la carte. Les fonctionnalités clés incluent :

- **RestaurantManager** : Une classe qui initialise la carte avec des marqueurs de restaurants et gère les événements de clic pour ajouter de nouveaux restaurants.
- **init** : Initialise l'écouteur d'événements pour les clics sur la carte.
- **openAddRestaurantForm** : Ouvre le formulaire pour ajouter un nouveau restaurant lorsque l'utilisateur clique sur la carte.
- **closeAddRestaurantForm** : Ferme le formulaire d'ajout de restaurant.
- **addRestaurant** : Ajoute un nouveau restaurant à la carte et envoie les informations du restaurant à l'API backend. Si la requête est réussie, un nouveau marqueur est ajouté à la carte avec les détails du restaurant.

Ces fichiers sont essentiels pour la configuration, la gestion des icônes des marqueurs sur la carte, et l'ajout dynamique de nouveaux restaurants, ce qui améliore l'interactivité et l'expérience utilisateur de l'application.

3) Dossier Récupération

1. `recup_ecoles.js`

Ce fichier est responsable de la récupération des données des établissements scolaires. Voici les principales fonctionnalités :

- **fetchSchoolData** : Fonction pour récupérer les données des écoles via une API. Elle filtre les établissements situés dans certaines communes et de certaines natures (collège, lycée général et technologique, lycée professionnel). Les données récupérées incluent l'ID, la latitude, la longitude, le nom et l'adresse des écoles.

2. `recup_incidents.js`

Ce fichier gère la récupération des données d'incidents. Voici les principales fonctionnalités :

- **fetchIncidentData** : Fonction pour récupérer les données des incidents via une API. Les données récupérées incluent l'ID, la latitude, la longitude, la description, la description courte, l'heure de début et de fin, ainsi qu'une description de l'emplacement.

3. `recup_meteo.js`

Ce fichier est utilisé pour récupérer les données météorologiques. Voici les principales fonctionnalités :

- **fetchMeteo** : Fonction pour récupérer les données météorologiques via une API. Elle retourne les données météorologiques récupérées ou un tableau vide en cas d'erreur.

4. `recup_restaurants.js`

Ce fichier gère la récupération et la manipulation des données des restaurants. Voici les principales fonctionnalités :

- **fetchRestaurantList** : Fonction pour récupérer la liste des restaurants avec leurs coordonnées.
- **fetchRestaurantDetails** : Fonction pour récupérer les détails d'un restaurant par son ID et ajouter une image provenant de l'API Unsplash.
- **fetchRestaurantHours, fetchAllRestaurantHours** : Fonctions pour récupérer les horaires d'ouverture d'un restaurant pour un jour spécifique ou pour toute la semaine.
- **postRestaurant** : Fonction pour ajouter un nouveau restaurant en envoyant une requête POST à l'API backend.
- **fetchReservation, postReservation** : Fonctions pour récupérer et poster des réservations de restaurant.

5. `recup_velo.js`

Ce fichier est utilisé pour récupérer les données des stations de vélos. Voici les principales fonctionnalités :

- **fetchVeloData** : Fonction pour récupérer les données des stations de vélos via une API. Les données récupérées incluent l'adresse, la capacité, la latitude et la longitude des stations.

Ces fichiers sont essentiels pour la récupération des données nécessaires à l'application, couvrant les domaines des écoles, des incidents, de la météo, des restaurants et des stations de vélos. Ils permettent d'obtenir et de manipuler les informations requises pour afficher ces données sur la carte et dans les différentes interfaces de l'application.

Diagramme de séquence du fonctionnement du serveur :

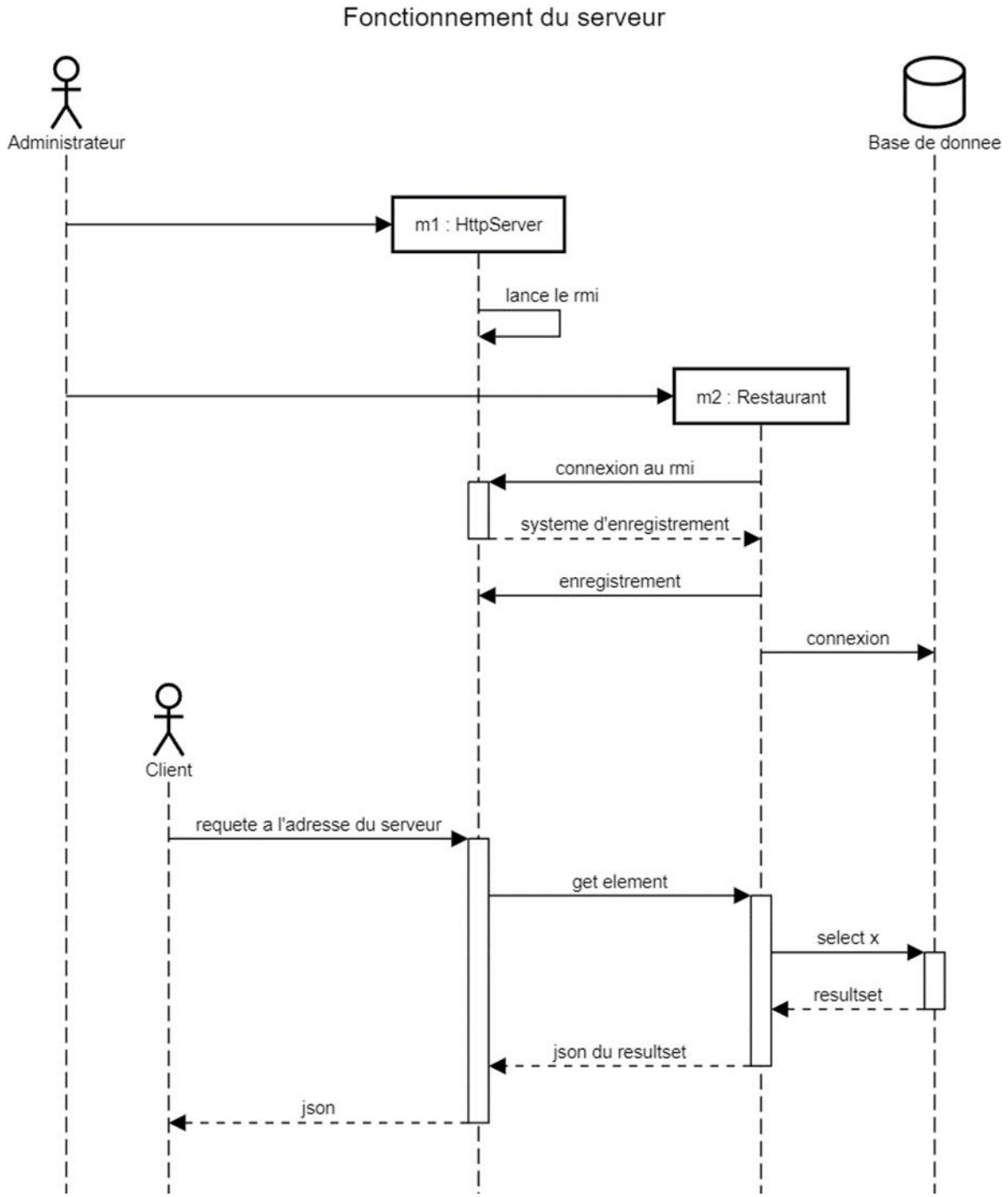
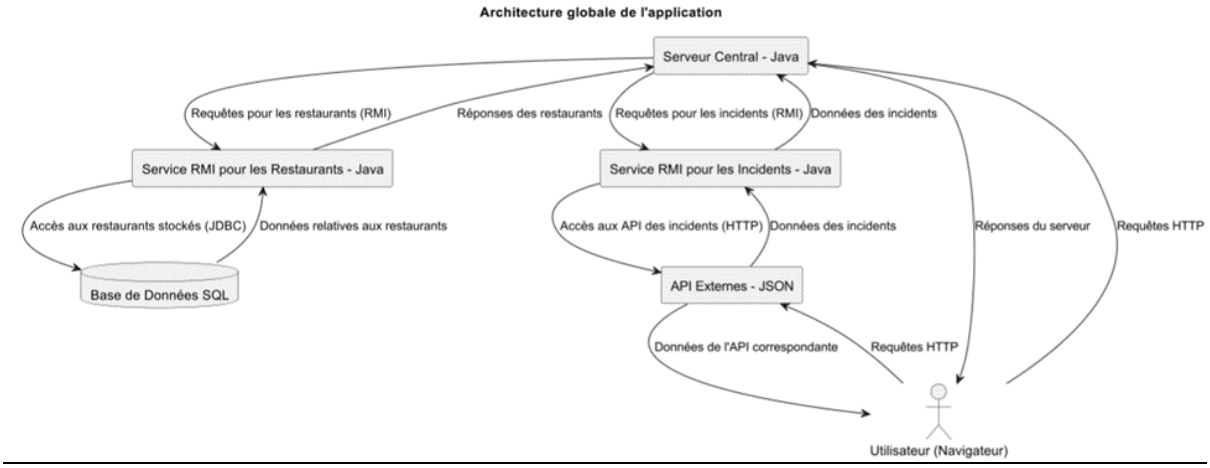


Schéma du fonctionnement global de l'application :



Rôles des différents acteurs :

Utilisateur (Navigateur)

- Envoie des requêtes HTTP au Serveur Central.

Serveur Central (Java)

- Reçoit les requêtes HTTP de l'utilisateur.
- Transmet les requêtes aux services RMI appropriés.
- Retourne les réponses HTTP à l'utilisateur.

Service RMI pour les Restaurants

- Reçoit les requêtes pour les restaurants du Serveur Central.
- Accède à la Base de Données SQL pour récupérer les informations des restaurants.
- Retourne les données relatives aux restaurants au Serveur Central.

Service RMI pour les Incidents

- Reçoit les requêtes pour les incidents du Serveur Central.
- Accède aux API Externes pour récupérer les données des incidents.
- Retourne les données des incidents au Serveur Central.

API Externes

- Fournissent les données des incidents via HTTP.

Base de Données SQL

- Stocke les informations des restaurants.
- Accède aux données pour le Service RMI pour les Restaurants.

Le schéma montre les flux de données entre les différents composants de l'application, illustrant comment les requêtes de l'utilisateur sont traitées par le Serveur Central, puis transmises aux services RMI et aux bases de données ou API externes, avant de retourner les réponses à l'utilisateur.

Résumé complet des fonctionnalités de l’application :

-Affichage d’une map fixée sur Nancy

-Affichage depuis un bouton des incidents de voirie sur Nancy

-Affichage depuis un bouton des parcs vélib de Nancy

-Affichage depuis un bouton, des écoles primaire, collège et lycée de Nancy

-Pour chaque affichage, un popUp présente des informations suplémentaire sur ce dernier

-Affichage depuis un bouton, de la météo du jour et du reste de semaine

-Affichage depuis un bouton des restaurants de nancy (vrai restaurants mais pas tous présent)

-Possibilité de consulter les heures REEL d’ouverture des restaurants

-Possibilités de consulter des informations suplémentaire et les notes des restaurant

-Possibilité de réserver un restaurants selon une date et un nombres de personnes

-Possibilité de créer un restaurant avec un CTRL + Clic droit n’importe où sur la carte. Il faudra lui ajouter un nom, une adresse et une note.