TP-DASI Collect’if

Dossier d’analyse

Introduction

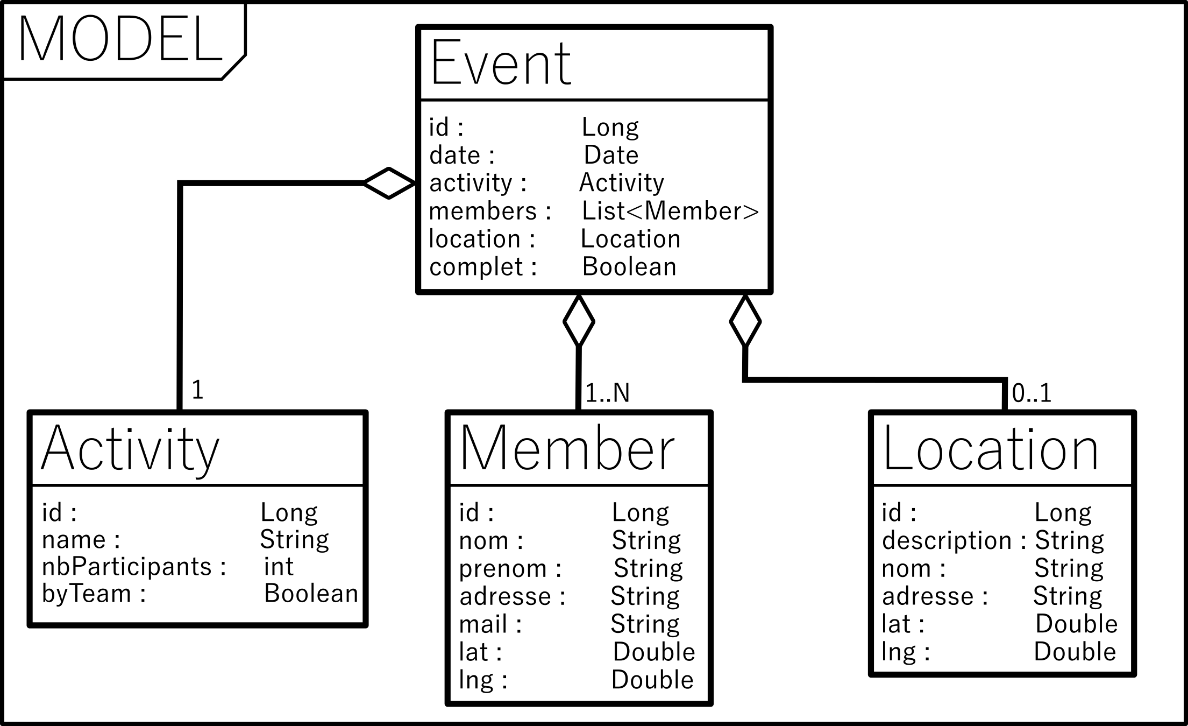
Le but de ce TP est de créer des services metier pour constituer le backend de l’application de COLLECT’IF en java avec de la persistance dans une base de données SQL.

Architecture du projet

Grâce à Maven et git (github), le projet est dévelopable sur plusieur ordinateur avec des IDE différents (IntelliJ et NetBeans). Le projet est divisé en plusieurs packages : le package ‘Utils’ regroupe des classes/méthodes de base qui sont relativement réutilisable. Le package metier.model contient les classes du model de l’application (classes Activity, Location, Event) et le package DAL contient leur Data Access Layer respectifs permettant d’obtenir, mettre à jour et créer des instances du model via la couche de persistance. Le package metier.service contient les services offrant à la vue toute les méthodes nécessaires (Classes Service, ServiceTechnique et ServiceResult). Enfin, le package ‘View’ contient le main et les vues (ConsoleConnexionView, ConsoleMemberView et ConsoleDashboardView).

Analyse du domaine

Après analyse du problème nous avons opté pour model simple composé de 4 classes (Entités de persistance dans le package metier.model) :



Nous avons par exemple choisit de représenter une demande par un évènement dont le lieu n’a pas été assigné (null). Pour chacune de ces classes, une classe de Data Access Layer (DAL) a été implémentée pour obtenir, créer ou modifier des instances des classes du model en conservant la cohérence avec la base de donnée SQL (JPA).

Maquette des IHM

Nous avons dessiné un prototype d’interface web avec inkscape (dessin vectoriel) qui a permis de mieux définir ce qui était nécessaire à la réalisation de l’application au niveau des services.

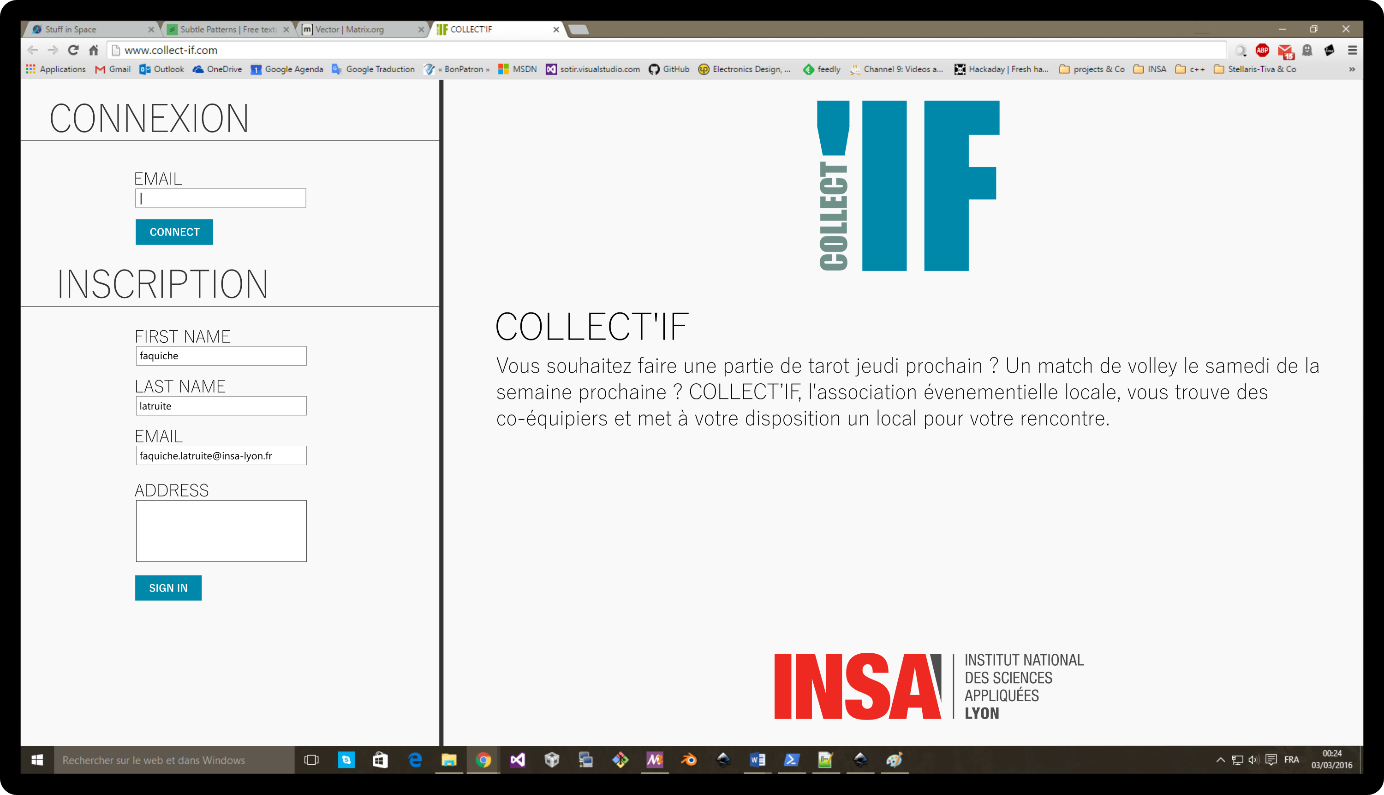
La vue est divisée en trois parties : la page d’accueil où l’utilisateur se connecte ou s’inscrit. L’espace adhérent où un membre connecté peut voir la liste des évènements et rejoindre ou créer un évènement. Et enfin, le dashboard de l’administrateur pouvant assigner un lieu aux événements.

Pour plus de détails sur l’interface ou plus de choix de logos le fichier ‘dessin.svg’ dans le dossier ‘GUI\_prototype’ contient tous les dessins vectoriels de l’interface.

L’interface console illustrant l’utilisation des services a été également été séparé en trois vues similaires.

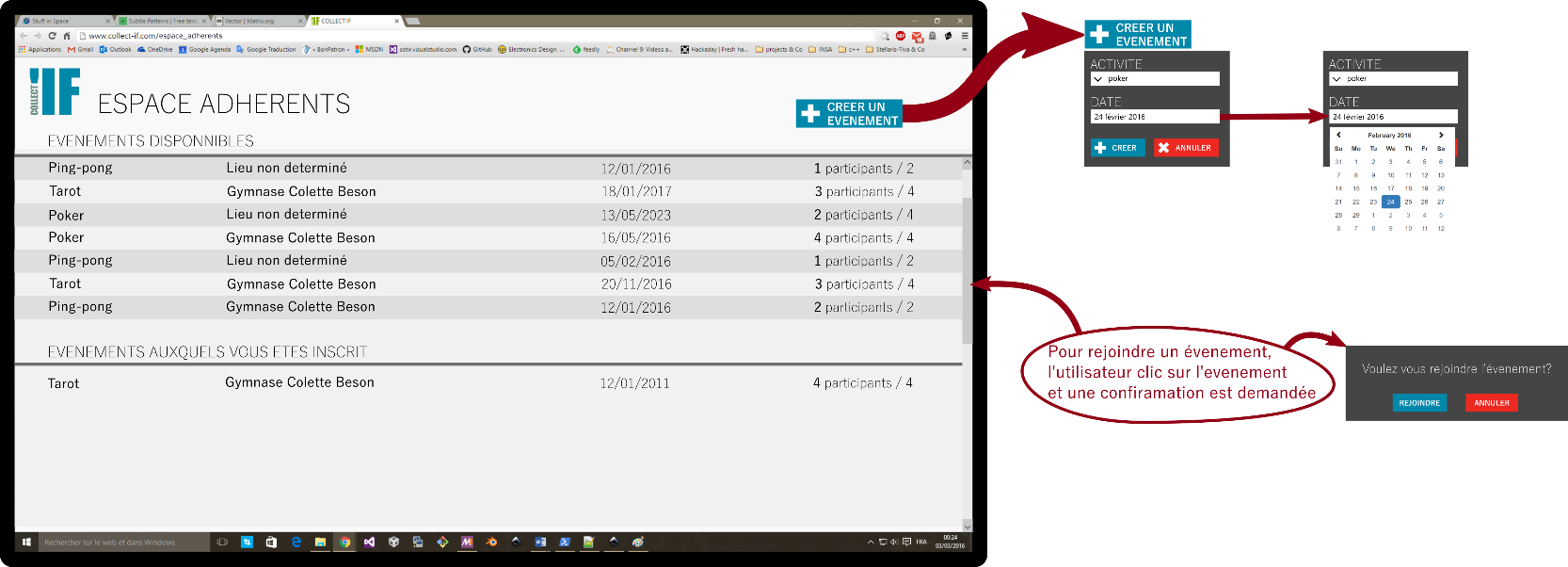
## Page d’accueil

Voici le dessin de la page d’accueil avec les formulaires de connexion et d’inscription à gauche :



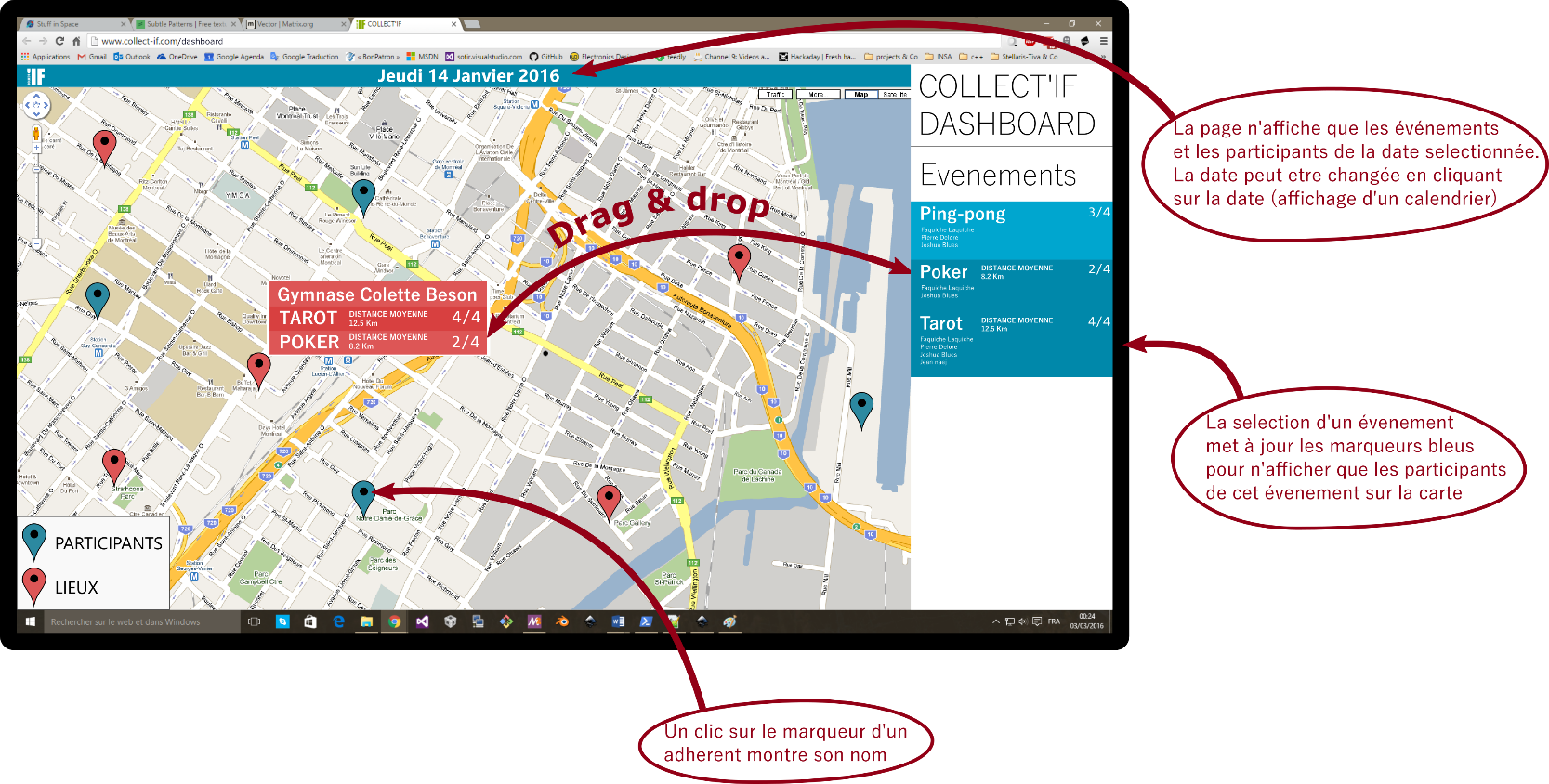
|  |  |
| --- | --- |
| **Bouton** | Action |
| **Connexion** | Permet la connexion d’un utilisateur à partir de son adresse mail. |
| **Inscription** | Permet l’inscription d’un nouvel adhérent en complétant les informations (nom, prénom, mail et adresse). |

# Espace adhérent

Une fois l’utilisateur connecté, il peut visualiser la liste des évènements qui ne sont pas complets ainsi que la liste des événements auxquels il participe sur la vue de l’espace adhérents. La création d’un évènement se fait grâce au bouton en haut à droite :

# Espace administrateur

Si l’email entré lors de la connexion correspond à l’email de l’administrateur (admin.collectif@insa-lyon.fr), l’utilisateur arrive sur le dashboard où est affichée une carte contenant les lieux et participants filtrés en fonction de la date sélectionnée en haut. Un volet à droite liste les événements de la date sélectionnée qui n’ont pas encore été assignés à un lieu. L’administrateur pourrait utiliser le drag and drop pour assigner un événement à un lieu. De plus, il est possible d’améliorer l’expérience utilisateur en permettant la sélection d’un évènement pour n’afficher que les participants à cet évènement ainsi que la distance moyenne de ceux-ci par rapport au lieu assigné à l’évènement :



Spécification des services

La classe *Services.java* fournit les services qui sont utilisés par la vue. Ces services ont été conçus pour être simples à utiliser et donner le plus de liberté possible à l’interface.

Nous avons choisi de ne pas utiliser les exceptions au niveau des services et de mettre toute la sémantique dans des objets serialisables ce qui devrait beaucoup simplifier la création de services REST. En effet, la plupart des méthodes de *Services.java* retournent un objet de type générique ‘ServiceResult<Result\_t, ErrorEnum\_t>’, ‘Result\_t’ étant le type du résultat si l’exécution de la méthode s’est bien déroulée et ‘ErrorEnum\_t’ étant l’énumération d’erreurs utilisée pour détailler les raison d’un éventuel échec.

En pratique, les services utilisent deux énumérations de codes d’erreur. Une première pour les méthodes de connexion et d’inscription (ConnexionError) et une deuxième pour les autres méthodes de ‘Services’ (Request\_Error) :

public enum ConnexionError {  
 *OK*, *EMPTY\_NAME*, *EMPTY\_FIRSTNAME*, *EMPTY\_ADDRESS*, *EMPTY\_EMAIL*, *WRONG\_ADDRESS*, *BAD\_EMAIL*, *DATABASE\_ERROR*}  
public enum Request\_Error {  
 *OK*, *WRONG\_EVENT\_ID*, *WRONG\_LIEU\_ID*, *WRONG\_MEMBER\_ID*, *WRONG\_ACTIVITY\_ID*, *FULL\_EVENT*, *DATABASE\_ERROR*}

Le tableau récapitulatif ci-dessous liste les méthodes disponibles dans la classe *Services.java.*

## Tableau descriptif de l’ensemble des services de Services.java

|  |  |
| --- | --- |
| ServiceResult<Member, ConnexionError> Connexion(String mail) | Méthode permettant la connexion d'un membre à partir de son email.  Retourne le membre connecté ou un code d'erreur détaillant les raisons de l'echec de la connexion (OK, DATABASE\_ERROR ou BAD\_EMAIL) |
| ServiceResult<Member, ConnexionError> Inscription(String nom, String prenom, String mail, String adresse) | Ajoute, si possible, un nouvel adhérent à la base de données avec comme informations son nom, prénom, mail et adresse postale. L’adresse email doit être unique et l’API google maps doit pouvoir trouver les coordonées de l'adresse pour que l'inscription ait lieu.  Retourne le membre connecté ou un code d'erreur détaillant les raisons de l'échec de l'inscription (OK, EMPTY\_FIRSTNAME, EMPTY\_ADDRESS, EMPTY\_EMAIL,WRONG\_ADDRESS, BAD\_EMAIL, DATABASE\_ERROR ou WRONG\_ADDRESS). |
| ServiceResult<List<Event>, Request\_Error> ListAllEvents() | Renvoie la liste de tous les évènements. Retourne DATABASE\_ERROR si la liste ne peut pas être récupérée. |
| ServiceResult<List<Event>, Request\_Error> ListEventsOfDate(Date date) | Renvoie la liste des évènements qui ont lieu à une date donnée en paramètre.  Retourne DATABASE\_ERROR si la liste ne peut pas être récupérée. |
| Request\_Error AssignLocationToEvent(long location\_id, long event\_id) | Permet d’assigner un lieu à un événement à partir de l’id du lieu et de l’événement.  Retourne un code d'erreur indiquant si l'assignation s'est déroulée correctement (DATABASE\_ERROR, WRONG\_LIEU\_ID, WRONG\_EVENT\_ID ou OK). |
| Request\_Error JoinEvent(long member\_id, long event\_id) | Permet à un adhérent de rejoindre un événement (les id de l’adhérent et de l’évènement sont entrés en paramètres).  Retourne un code d'erreur indiquant si le membre a pu rejoindre l'évènement (DATABASE\_ERROR, FULL\_EVENT, WRONG\_EVENT\_ID, WRONG\_MEMBER\_ID ou OK) |
| ServiceResult<Event, Request\_Error> CreateEvent(long member\_id, int activity\_id, Date date) | Permet de créer un événement en donnant en paramètre une date, une activité et l’id de l’adhérent à l’origine de la création de l’événement (l’adhérent participe forcément à l’événement).  Retourne l'évènement crée ou un code d'erreur (OK, WRONG\_ACTIVITY\_ID, WRONG\_MEMBER\_ID ou DATABASE\_ERROR) |
| ServiceResult<List<Event>, Request\_Error> ListEventsOfMember(long member\_id) | Renvoie la liste des évènements associés à un adhérent à partir de son id ou un code d'erreur (OK ou DATABASE\_ERROR).  Retourne également DATABASE\_ERROR si l'id du membre est mauvais. |
| ServiceResult<List<Event>, Request\_Error> ListAvailableEvents() | Renvoie la liste des évènements qui ne sont pas encore complets et un code d’erreur (OK ou DATABASE\_ERROR) |
| ServiceResult<List<Activity>, Request\_Error> ListActivities() | Renvoie la liste des activités possibles et un code d’erreur (OK ou DATABASE\_ERROR) |
| ServiceResult<List<Location>, Request\_Error> ListLocations() | Renvoie la liste des lieux possibles pour un évènement et un code d’erreur (OK ou DATABASE\_ERROR) |

# Tableau descriptif de l’ensemble des services de ServiceTechnique.java

La classe ServiceTechnique regroupe les services permettant l'envoi de mail et la géolocalisation via l'API google maps.

|  |  |
| --- | --- |
| SendSuccessfullInscriptionMail(Adherent adherent) | Simule l’envoie d’un mail pour une inscription à Collect’if réussie. |
| SendFailedInscriptionMail(Adherent adherent) | Simule l’envoie d’un mail pour une inscription échouée à Collect’if. |
| SendEventMail(Event event) | Simule l’envoie d’un mail pour indiquer aux participants d’un événement le lieu de l’événement et les personnes avec qui ils vont jouer. |
| long Distance(LatLng position1, LatLng position2) | Calcule à l’aide de l’Api Google la distance entre deux coordonnées (Latitudes/Longitudes). |
| LatLng GetLatLngFromAddress(String address) | Calcule, à l’aide de l’API Google et à partir d’une adresse postale, les coordonnées (Latitudes/Longitudes) correspondant. |