# CODE TP1 SI : B3330

# SOTIR Paul-Emmanuel

# CHAPELLE Victoire

Main.java

package TP1\_SI.View;  
  
import TP1\_SI.DAL.\*;  
  
*/\*\*  
 \* Fonction principale démarant la vue de connexion  
 \** ***@author*** *B3330  
 \*/*public class Main {  
  
 public static void main(String[] arg) {  
 JpaUtil.*init*();  
 JpaUtil.*creerEntityManager*();  
  
 ConsoleConnexionView connexion\_view = new ConsoleConnexionView();  
 connexion\_view.run();  
  
 JpaUtil.*fermerEntityManager*();  
 JpaUtil.*destroy*();  
 }  
}

Callalbe.java (utils)

package TP1\_SI.Utils;  
  
*/\*\*  
 \* Interface représentant une fonction ne prenant pas de paramètres et retournant void.  
 \* Cette interface permet de créer un mécaniqme similaire aux pointeurs de fonctions.  
 \** ***@author*** *B3330  
 \*/*public interface Callable {  
 public void call();  
}

ConsoleUtils (Utils)

package TP1\_SI.Utils;  
  
import java.util.List;  
import java.io.IOException;  
import java.io.BufferedReader;  
import java.io.InputStreamReader;  
  
*/\*\*  
 \* Classe regourpant quelques fonction utiles à l'interface console  
 \** ***@author*** *B3330  
 \*/*public class ConsoleUtils {  
  
 public static String lireChaine(String invite) {  
 String chaineLue = null;  
 System.*out*.print(invite);  
 try {  
 InputStreamReader isr = new InputStreamReader(System.*in*);  
 BufferedReader br = new BufferedReader(isr);  
 chaineLue = br.readLine();  
 } catch (IOException exception) {  
 exception.printStackTrace(System.*err*);  
 }  
 return chaineLue;  
  
 }  
  
 public static Integer lireInteger(String invite) {  
 Integer valeurLue = null;  
 try {  
 valeurLue = new Integer(*lireChaine*(invite));  
 } catch (java.lang.NumberFormatException e) {  
 System.*out*.println("erreur de saisie");  
 valeurLue = *lireInteger*(invite);  
 }  
 return valeurLue;  
 }  
  
 public static Integer lireInteger(String invite, List<Integer> valeursPossibles) {  
 Integer valeurLue = null;  
 try {  
 valeurLue = new Integer(*lireChaine*(invite));  
 if (!(valeursPossibles.contains(valeurLue))) {  
 throw new Exception();  
 }  
 } catch (Exception e) {  
 System.*out*.println("erreur de saisie");  
 valeurLue = *lireInteger*(invite, valeursPossibles);  
 }  
 return valeurLue;  
 }  
}

ConsoleMenu (Utils)

package TP1\_SI.Utils;  
  
import java.io.InputStreamReader;  
import java.io.BufferedReader;  
import java.util.HashMap;  
import java.util.Map;  
  
*/\*\*  
 \* Classe aidant à la création d'un menu sur console.  
 \* Un choix est ajouté au menu en fournissant un nom de l'option et une fonction à appeler  
 \* si cette option est choisie (class implémentant l'interface 'Callable').  
 \** ***@author*** *B3330  
 \*/*public class ConsoleMenu {  
  
 */\*\*  
 \* Constructeur de la classe 'ConsoleMenu'  
 \** ***@param*** *menu\_name nom du menu  
 \*/* public ConsoleMenu(String menu\_name) {  
 menu\_choices = new HashMap<>();  
 this.menu\_name = menu\_name;  
 this.exit\_function = null;  
 }  
  
 */\*\*  
 \* Constructeur de la classe permmettant de specifier une fonction à appeler pour quitter le menu  
 \** ***@param*** *menu\_name nom du menu  
 \** ***@param*** *exit\_function fonction à appeler si l'utilisateur veux quitter le menu  
 \*/* public ConsoleMenu(String menu\_name, Callable exit\_function) {  
 menu\_choices = new HashMap<>();  
 this.menu\_name = menu\_name;  
 this.exit\_function = exit\_function;  
 }  
  
 */\*\*  
 \* Ajoute un choix au menu  
 \** ***@param*** *function fonction à appeler si le l'option est choisie  
 \** ***@param*** *choice\_name nom de l'option affiché dans le menu  
 \*/* public void addChoice(Callable function, String choice\_name) {  
 menu\_choices.put(menu\_choices.size(), new menu\_choice(choice\_name, function));  
 }  
  
 */\*\*  
 \* Affiche le menu  
 \*/* public void runMenu() {  
 while (true) {  
 System.*out*.println("\n############## " + menu\_name + " ##############");  
 int i = 0;  
 for (Map.Entry<Integer, menu\_choice> entry : menu\_choices.entrySet()) {  
 System.*out*.println("\t" + i + ":\t" + entry.getValue().name);  
 i++;  
 }  
  
 if(exit\_function != null)  
 System.*out*.println("\t \tType any other number to exit menu");  
  
 while(true)  
 {  
 int num = ConsoleUtils.*lireInteger*("Enter your choice: ");  
 if(num >= menu\_choices.size() || num < 0)  
 {  
 if(exit\_function != null)  
 {  
 System.*out*.println();  
 exit\_function.call();  
 return;  
 }  
 }  
 else  
 {  
 menu\_choice choice = menu\_choices.get(num);  
 System.*out*.println();  
 choice.callback.call();  
 break;  
 }  
 }  
 }  
 }  
  
 private static class menu\_choice  
 {  
 public menu\_choice(String name, Callable callback)  
 {  
 this.name = name;  
 this.callback = callback;  
 }  
  
 public String name;  
 public Callable callback;  
 }  
  
 private HashMap<Integer, menu\_choice> menu\_choices;  
 private Callable exit\_function;  
 private String menu\_name;  
}

Location (métier.model)

package TP1\_SI.metier.model;  
  
import com.google.maps.model.LatLng;  
  
import javax.persistence.Id;  
import java.io.Serializable;  
import javax.persistence.Entity;  
import javax.persistence.GeneratedValue;  
import javax.persistence.GenerationType;  
  
*/\*\*  
 \* Classe représentant un lieux définit par des coordonées, une addresse, un nom et une*

*\* description.  
 \** ***@author*** *B3330  
 \*/*@Entity  
public class Location implements Serializable {  
 @Id  
 @GeneratedValue(strategy = GenerationType.*AUTO*)  
 private Long id;  
 private String address;  
 private String description;  
 private String denomination;  
 private Double latitude;  
 private Double longitude;  
  
 public Location() { }  
  
 public Location(String denomination, String description, String address) {  
 this.denomination = denomination;  
 this.description = description;  
 this.address = address;  
 }  
  
 public Long getId() {  
 return id;  
 }  
  
 public String getDenomination() {  
 return denomination;  
 }  
  
 public String getDescription() {  
 return description;  
 }  
  
 public String getAddress() {  
 return address;  
 }  
  
 public Double getLongitude() {  
 return longitude;  
 }  
  
 public Double getLatitude() {  
 return latitude;  
 }  
  
 public void setCoordonnees(LatLng latLng) {  
 this.longitude = latLng.lng;  
 this.latitude = latLng.lat;  
 }  
  
 public void setDenomination(String denomination) {  
 this.denomination = denomination;  
 }  
  
 public void setDescription(String description) {  
 this.description = description;  
 }  
  
 public void setAddress(String adresse) {  
 this.address = adresse;  
 }  
  
 public void setLongitude(Double longitude) {  
 this.longitude = longitude;  
 }  
  
 public void setLatitude(Double latitude) {  
 this.latitude = latitude;  
 }  
  
 @Override  
 public int hashCode() {  
 return (id != null ? id.hashCode() : 0);  
 }  
  
 @Override  
 public boolean equals(Object object) {  
 if (!(object instanceof Location))  
 return false;  
  
 Location other = (Location) object;  
 if ((this.id == null && other.id != null) || (this.id != null && !this.id.equals(other.id)))  
 return false;  
 return true;  
 }  
  
 @Override  
 public String toString() {  
 return "Location{" + "id=" + id + ", denomination=" + denomination + ", description=" + description + ", address=" + address + ", longitude=" + longitude + ", latitude=" + latitude + '}';  
 }  
}

Member (métier.model)

package TP1\_SI.metier.model;  
  
import com.google.maps.model.LatLng;  
  
import javax.persistence.Id;  
import java.io.Serializable;  
import javax.persistence.Entity;  
import javax.persistence.GeneratedValue;  
import javax.persistence.GenerationType;  
  
*/\*\*  
 \* Classe représentant un membre de l'association pouvant rejoindre ou créer des événements  
 \** ***@author*** *B3330  
 \*/*@Entity  
public class Member implements Serializable {  
 @Id  
 @GeneratedValue(strategy = GenerationType.*AUTO*)  
 private Long id;  
 private String address;  
 private Double lat;  
 private Double lng;  
 private String mail;  
 private String nom;  
 private String prenom;  
  
 public Member() { }  
  
 public Member(String nom, String prenom, String address, String mail, LatLng coordonnees) {  
 this.nom = nom;  
 this.prenom = prenom;  
 this.mail = mail;  
 this.address = address;  
 this.lat = coordonnees.lat;  
 this.lng = coordonnees.lng;  
 }  
  
 public Long getId() {  
 return id;  
 }  
  
 public String getNom() {  
 return nom;  
 }  
  
 public String getPrenom() {  
 return prenom;  
 }  
  
 public String getMail() {  
 return mail;  
 }  
  
 public String getAddress() {  
 return address;  
 }  
  
 public Double getLatitude() {  
 return lat;  
 }  
  
 public Double getLongitude() {  
 return lng;  
 }  
  
 public void setNom(String nom) {  
 this.nom = nom;  
 }  
  
 public void setPrenom(String prenom) {  
 this.prenom = prenom;  
 }  
  
 public void setMail(String mail) {  
 this.mail = mail;  
 }  
  
 public void setAddress(String adresse) {  
 this.address = adresse;  
 }  
  
 public void setCoordonnees(LatLng latLng) {  
 this.lat = latLng.lat;  
 this.lng = latLng.lng;  
 }  
  
 @Override  
 public int hashCode() {  
 return (id != null ? id.hashCode() : 0);  
 }  
  
 @Override  
 public boolean equals(Object object) {  
 if (!(object instanceof Member))  
 return false;  
  
 Member other = (Member) object;  
 if ((this.id == null && other.id != null) || (this.id != null && !this.id.equals(other.id)))  
 return false;  
 return true;  
 }  
  
 @Override  
 public String toString() {  
 return "Member{" + "id=" + id + ", nom=" + nom + ", prenom=" + prenom + ", mail=" + mail + ", adresse=" + address + ", longitude=" + lng + ", latitude=" + lat + '}';  
 }  
}

Event (métier.model)

package TP1\_SI.metier.model;  
  
import java.sql.Date;  
import java.util.ArrayList;  
import java.util.List;  
import java.io.Serializable;  
import javax.persistence.Id;  
import javax.persistence.Entity;  
import javax.persistence.GeneratedValue;  
import javax.persistence.GenerationType;  
  
*/\*\*  
 \* Classe représentant un événement constitué de membres voulant participer à cet événement.  
 \* Un événement peut ne pas être encore assigné à un lieu (location == null)  
 \*  
 \** ***@author*** *B3330  
 \*/*@Entity  
public class Event implements Serializable {  
 @Id  
 @GeneratedValue(strategy = GenerationType.*AUTO*)  
 private Long id;  
 private Date date;  
 private Activity activity;  
 private List<Member> members;  
 private Location location;  
 private boolean complet;  
  
 public Event() {  
 }  
  
 public Event(Date date, Activity activity, Member creator) {  
 this.activity = activity;  
 this.date = date;  
 this.members = new ArrayList<>();  
 this.members.add(creator);  
 }  
  
 public Long getId() {  
 return id;  
 }  
  
 public void setId(Long id) {  
 this.id = id;  
 }  
  
 public Date getDate() {  
 return date;  
 }  
  
 public void setDate(Date date) {  
 this.date = date;  
 }  
  
 public Activity getActivity() {  
 return activity;  
 }  
  
 public void setActivity(Activity activity) {  
 this.activity = activity;  
 }  
  
 public List<Member> getMembers() {  
 return members;  
 }  
  
 public void setMembers(List<Member> members) {  
 this.members = members;  
 }  
  
 public Location getLocation() {  
 return location;  
 }  
  
 public void setLocation(Location location) {  
 this.location = location;  
 }  
  
 public boolean getComplet() {  
 return this.complet;  
 }  
  
 public void setComplet() {  
 this.complet = true;  
 }  
  
 public List<Member> getAdherentsEquipe1() {  
 if (activity.isByTeam()) {  
 List<Member> equipe\_1 = null;  
 for (int i = 0; i < this.members.size(); i += 2) {  
 equipe\_1.add(members.get(i));  
 }  
 return equipe\_1;  
 }  
 return null;  
 }  
  
 public List<Member> getAdherentsEquipe2() {  
 if (activity.isByTeam()) {  
 List<Member> equipe\_2 = null;  
 for (int i = 1; i < this.members.size(); i += 2) {  
 equipe\_2.add(members.get(i));  
 }  
 return equipe\_2;  
 }  
 return null;  
 }  
  
 @Override  
 public int hashCode() {  
 return (id != null ? id.hashCode() : 0);  
 }  
  
 @Override  
 public boolean equals(Object object) {  
 // *TODO: Warning - this method won't work in the case the id fields are not set* if (!(object instanceof Event)) {  
 return false;  
 }  
 Event other = (Event) object;  
 if ((this.id == null && other.id != null) || (this.id != null && !this.id.equals(other.id))) {  
 return false;  
 }  
 return true;  
 }  
  
 @Override  
 public String toString() {  
 return "Event{ id=" + id + ", date=" + date + ", location=" + location + ", is\_complet=" +

complet + " }";  
 }  
  
}

Activity (métier.model)

package TP1\_SI.metier.model;  
  
import java.io.Serializable;  
import javax.persistence.Id;  
import javax.persistence.Entity;  
import javax.persistence.GeneratedValue;  
import javax.persistence.GenerationType;  
  
*/\*\*  
 \* Classe représentatn une activité.  
 \* Une activité peut être partiquée par une ou deux équipes (booléen 'byTeam').  
 \** ***@author*** *B3330  
 \*/*@Entity  
public class Activity implements Serializable {  
 @Id  
 @GeneratedValue(strategy = GenerationType.*AUTO*)  
 private Long id;  
 private String name;  
 private Integer nbParticipants;  
 private Boolean byTeam;  
  
 public Activity() { }  
  
 public Activity(String name, Boolean byTeam, Integer nbParticipants) {  
 this.name = name;  
 this.byTeam = byTeam;  
 this.nbParticipants = nbParticipants;  
 }  
  
 public Long getId() {  
 return id;  
 }  
  
 public String getName() {  
 return name;  
 }  
  
 public Boolean isByTeam() {  
 return byTeam;  
 }  
  
 public Integer getNbParticipants() {  
 return nbParticipants;  
 }  
  
 public void setName(String denomination) {  
 this.name = denomination;  
 }  
  
 public void setByTeam(Boolean parEquipe) {  
 this.byTeam = parEquipe;  
 }  
  
 public void setNbParticipants(Integer nbParticipants) {  
 this.nbParticipants = nbParticipants;  
 }  
  
 @Override  
 public int hashCode() {  
 return (id != null ? id.hashCode() : 0);  
 }  
  
 @Override  
 public boolean equals(Object object) {  
 if (!(object instanceof Activity))  
 return false;  
  
 Activity other = (Activity) object;  
 if ((this.id == null && other.id != null) || (this.id != null &&

!this.id.equals(other.id)))  
 return false;  
 return true;  
 }  
  
 @Override  
 public String toString() {  
 return "Activity{" + "id=" + id + ", denomination=" + name + ", byTeam=" + byTeam +

", nbParticipants=" + nbParticipants + '}';  
 }  
}

LocationDAL (DAL)

package TP1\_SI.DAL;  
  
import java.util.List;  
import javax.persistence.Query;  
import javax.persistence.EntityManager;  
  
import TP1\_SI.metier.model.Location;  
  
*/\*\*  
 \* Data Access Layer permettant d'obtenir, de créer et modifier des instances de la classe 'Location'.  
 \* Utilise JPA pour persiter les Lieux.  
 \** ***@author*** *B3330  
 \*/*public class LocationDAL {  
  
 public void create(Location location) throws Throwable {  
 EntityManager em = JpaUtil.*obtenirEntityManager*();  
 em.persist(location);  
 }  
  
 public Location update(Location location) throws Throwable {  
 EntityManager em = JpaUtil.*obtenirEntityManager*();  
 return em.merge(location);  
 }  
  
 public Location findById(long id) throws Throwable {  
 EntityManager em = JpaUtil.*obtenirEntityManager*();  
 return em.find(Location.class, id);  
 }  
  
 public List<Location> findAll() throws Throwable {  
 EntityManager em = JpaUtil.*obtenirEntityManager*();  
 Query q = em.createQuery("SELECT l FROM Location l");  
 return (List<Location>) q.getResultList();  
 }  
}

MemberDAL (DAL)

package TP1\_SI.DAL;  
  
import java.util.List;  
import javax.persistence.Query;  
import javax.persistence.EntityManager;  
  
import TP1\_SI.metier.model.Member;  
  
*/\*\*  
 \* Data Access Layer permettant d'obtenir, de créer et modifier des instances de la classe 'Member'.  
 \* Utilise JPA pour persiter les Membres.  
 \** ***@author*** *B3330  
 \*/*public class MemberDAL {  
  
 public void create(Member member) throws Throwable {  
 EntityManager em = JpaUtil.*obtenirEntityManager*();  
 em.persist(member);  
 }  
  
 public Member update(Member member) throws Throwable {  
 EntityManager em = JpaUtil.*obtenirEntityManager*();  
 return em.merge(member);  
 }  
  
 public Member findById(long id) throws Throwable {  
 EntityManager em = JpaUtil.*obtenirEntityManager*();  
 return em.find(Member.class, id);  
 }  
  
 public Member findByMail(String mail) throws Throwable {  
 EntityManager em = JpaUtil.*obtenirEntityManager*();  
 Query q = em.createQuery("SELECT a FROM Member a WHERE a.mail='" + mail + "'");  
 List<Member> members = (List<Member>) q.getResultList();  
  
 if (members.size() >= 1)  
 return members.get(0);  
 return null;  
 }  
  
 public List<Member> findAll() throws Throwable {  
 EntityManager em = JpaUtil.*obtenirEntityManager*();  
 Query q = em.createQuery("SELECT a FROM Member a");  
 return (List<Member>)q.getResultList();  
 }  
}

EventDAL (DAL)

package TP1\_SI.DAL;  
  
import javax.persistence.EntityManager;  
import javax.persistence.Query;  
import java.util.List;  
import java.sql.Date;  
  
import TP1\_SI.metier.model.Member;  
import TP1\_SI.metier.model.Event;  
  
*/\*\*  
 \* Data Access Layer permettant d'obtenir, de créer et modifier des instances de la classe 'Event'.  
 \* Utilise JPA pour persiter les Evenements.  
 \** ***@author*** *B3330  
 \*/*public class EventDAL {  
  
 public void create(Event event) throws Throwable {  
 EntityManager em = JpaUtil.*obtenirEntityManager*();  
 em.persist(event);  
 }  
  
 public Event update(Event event) throws Throwable {  
 EntityManager em = JpaUtil.*obtenirEntityManager*();  
 return em.merge(event);  
 }  
  
 public Event findById(long id) throws Throwable {  
 EntityManager em = JpaUtil.*obtenirEntityManager*();  
 return em.find(Event.class, id);  
 }  
  
 public void AddAdherentToEvent(Event event, Member member) throws Throwable {  
 event.getMembers().add(member);  
 update(event);  
 if (event.getMembers().size() == event.getActivity().getNbParticipants())  
 event.setComplet();  
 }  
  
 public List<Event> findAll() throws Throwable {  
 EntityManager em = JpaUtil.*obtenirEntityManager*();  
 Query q = em.createQuery("SELECT a FROM Event a");  
 return (List<Event>)q.getResultList();  
 }  
  
 public List<Event> findDispo() throws Throwable {  
 EntityManager em = JpaUtil.*obtenirEntityManager*();  
 Query q = em.createQuery("SELECT a FROM Event a WHERE a.complet = FALSE");  
 return (List<Event>)q.getResultList();  
 }  
  
 public List<Event> findByMember(long member\_id) throws Throwable {  
 EntityManager em = JpaUtil.*obtenirEntityManager*();  
 Query q = em.createQuery("SELECT e FROM Event e INNER JOIN e.members event\_member WHERE event\_member.id = " + member\_id);  
 return (List<Event>)q.getResultList();  
 }  
  
 public List<Event> findByDate(Date date) throws Throwable {  
 EntityManager em = JpaUtil.*obtenirEntityManager*();  
 Query q = em.createQuery("SELECT a FROM Event a WHERE a.date = " + date);  
 return (List<Event>)q.getResultList();  
 }  
  
}

ActivityDAL (DAL)

package TP1\_SI.DAL;  
  
import java.util.List;  
import javax.persistence.Query;  
import javax.persistence.EntityManager;  
  
import TP1\_SI.metier.model.Activity;  
  
*/\*\*  
 \* Data Access Layer permettant d'obtenir, de créer et modifier des instances de la classe 'Activity'.  
 \* Utilise JPA pour persiter les Activitées.  
 \** ***@author*** *B3330  
 \*/*public class ActivityDAL {  
  
 public void create(Activity activity) throws Throwable {  
 EntityManager em = JpaUtil.*obtenirEntityManager*();  
 em.persist(activity);  
 }  
  
 public Activity update(Activity activity) throws Throwable {  
 EntityManager em = JpaUtil.*obtenirEntityManager*();  
 return em.merge(activity);  
 }  
  
 public Activity findById(long id) throws Throwable {  
 EntityManager em = JpaUtil.*obtenirEntityManager*();  
 return em.find(Activity.class, id);  
 }  
  
 public List<Activity> findAll() throws Throwable {  
 EntityManager em = JpaUtil.*obtenirEntityManager*();  
 Query q = em.createQuery("SELECT a FROM Activity a");  
 return (List<Activity>) q.getResultList();  
 }  
}

ServiceResult (metier.service)

package TP1\_SI.metier.service;  
  
import java.io.Serializable;  
  
*/\*\*  
 \* Classe générique représentant le résultat d'un appel à une méthode d'un service.  
 \* Cette classe peut être utilisé comme type de retour pour fournir à la fois un résultat de*

*\* type 'Result\_t'  
 \* et un éventuel code d'erreur appartenant à l'énumératrion 'ErrorEnum\_t'.  
 \* Ce méchanisme peut être préférable à l'utilisation d'exeptions car ficilite l'ajout*

*\* d'informations sur une erreur  
 \* et peut être sérialisé (ce qui peut faciliter la création de services REST en serialisant*

*\* directement en JSON cette  
 \* classe par exemple ;) )  
 \** ***@param*** <*Result\_t*> *type du résultat de l'appel  
 \** ***@param*** <*ErrorEnum\_t*> *type des codes d'erreur  
 \** ***@author*** *B3330  
 \*/*public class ServiceResult<Result\_t, ErrorEnum\_t> implements Serializable {  
  
 public ServiceResult(Result\_t result, ErrorEnum\_t error) {  
 this.result = result;  
 this.error = error;  
 }  
  
 @Override  
 public int hashCode() {  
 return (result != null ? result.hashCode() : 0);  
 }  
  
 @Override  
 public boolean equals(Object object) {  
 if (!(object instanceof ServiceResult))  
 return false;  
  
 ServiceResult other = (ServiceResult)object;  
  
 if ((this.result == null && other.result != null) || (this.result != null && !this.result.equals(other.result)))  
 return false;  
 if ((this.error == null && other.error != null) || (this.error != null && !this.error.equals(other.error)))  
 return false;  
 return true;  
 }  
  
 @Override  
 public String toString() {  
 if(result != null)  
 return "ServiceResult{ result = '" + result + "', error\_code = '" + error + "'}";  
 else  
 return "ServiceResult{ error\_code = '" + error + "'}";  
 }  
  
 public Result\_t result;  
 public ErrorEnum\_t error;  
}

ServiceTechnique (metier.service)

package TP1\_SI.metier.service;  
  
import TP1\_SI.metier.model.Member;  
import TP1\_SI.metier.model.Event;  
  
import com.google.maps.DistanceMatrixApi;  
import com.google.maps.GeoApiContext;  
import com.google.maps.GeocodingApi;  
import com.google.maps.model.DistanceMatrix;  
import com.google.maps.model.GeocodingResult;  
import com.google.maps.model.LatLng;  
  
import java.util.List;  
  
*/\*\*  
 \* Classe regroupant les méthodes des services techniques (services permettant l'envoi de mail et  
 \* la géolocalisation via l'API google maps).  
 \** ***@author*** *B3330  
 \*/*public class ServiceTechnique {  
  
 /\*private static final String SMTP\_HOST1 = "";  
 private static final String LOGIN\_SMTP1 = "paul-emmanuel.sotir@insa-lyon.fr";  
 private static final String IMAP\_ACCOUNT1 = "";\*/  
 private static final String *GeoAPIKey* = "AIzaSyDcVVJjfmxsNdbdUYeg9MjQoJJ6THPuap4";  
  
 */\*\*  
 \* Envoie un mail de confirmation d'inscription d'un membre  
 \** ***@param*** *member membre venant de s'inscrire  
 \*/* public static void SendSuccessfullInscriptionMail(Member member) {  
 /\*Properties properties = new Properties();   
 properties.setProperty("mail.transport.protocol", "smtp");   
 properties.setProperty("mail.smtp.host", SMTP\_HOST1);   
 properties.setProperty("mail.smtp.user", LOGIN\_SMTP1);   
 properties.setProperty("mail.from", IMAP\_ACCOUNT1);   
 Session session = Session.getInstance(properties);\*/  
 String corps = "Bonjour " + member.getPrenom() + ",\n" +  
 "Nous vous confirmons votre adhésion à l'association COLLECT’IF. Votre numéro d'adhérent est : " + member.getId();  
  
 System.*out*.println(corps);  
 }  
  
 */\*\*  
 \* Envoie un mail expliquant que l'inscription d'un membre a échouée  
 \** ***@param*** *member membre voulant s'inscrire  
 \*/* public static void SendFailedInscriptionMail(Member member) {  
 String corps = "Bonjour " + member.getPrenom() + ",\n" +  
 "Votre adhésion à l'association COLLECT’IF a malencontreusement échoué... Merci de recommencer ultérieurement.";  
  
 System.*out*.println(corps);  
 }  
  
 */\*\*  
 \* Envoie un mail pour notifier les membres d'un evenement que l'évenement a été assigné à un lieu  
 \** ***@param*** *event Evenement ayant été assigné à un lieu  
 \*/* public static void SendEventMail(Event event) {  
 final LatLng coord = new LatLng(event.getLocation().getLatitude(), event.getLocation().getLongitude());  
 List<Member> members = event.getMembers();  
  
 for (int i = 0; i < members.size(); i++) {  
 try {  
 long distance = *Distance*(coord, new LatLng(members.get(i).getLatitude(), members.get(i).getLongitude()));  
  
 String corps = "Bonjour" + members.get(i).getPrenom() + ",\n" +  
 "Comme vous l'aviez souhaité, COLLECT’IF organise un évènement de " + event.getActivity().getName() + " le " + event.getDate() + ".\n" +  
 "Vous trouverez ci-dessous les détails de cet évènement.\n" +  
 "Associativement vôtre,\n" +  
 "Le Responsable de l'Association\n" +  
 "Evénement : " + event.getActivity().getName() + "\n" +  
 "Date : " + event.getDate() + "\n" +  
 "Location : " + event.getLocation().getDenomination() + ", " + event.getLocation().getAddress() + "\n" +  
 "(à " + distance + " km de chez vous)\n" +  
 "Vous jouerez avec :\n";  
 if (event.getActivity().isByTeam()) {  
 List<Member> equipe\_1 = event.getAdherentsEquipe1();  
 for (int j = 0; j < equipe\_1.size(); j++) {  
 if (equipe\_1.get(j) != members.get(i))  
 corps += equipe\_1.get(j).getPrenom() + " " + equipe\_1.get(j).getNom() + "\n";  
 }  
 corps += "Contre :\n";  
  
 List<Member> equipe\_2 = event.getAdherentsEquipe2();  
 for (int j = 0; j < equipe\_2.size(); j++) {  
 if (equipe\_2.get(j) != members.get(i))  
 corps += equipe\_2.get(j).getPrenom() + " " + equipe\_2.get(j).getNom() + "\n";  
 }  
  
 } else {  
 for (int j = 0; j < members.size(); j++) {  
 if (j != i)  
 corps += members.get(j).getPrenom() + " " + members.get(j).getNom() + "\n";  
 }  
 }  
  
 // *TODO: afficher plus d'infos avec une mise en forme* System.*out*.println(corps);  
 } catch (Exception e) {  
 //*TODO: gerer ca* }  
 }  
 }  
  
 */\*\*  
 \* Donne la distance entre deux positions  
 \** ***@param*** *position1 coordonnées de la première position  
 \** ***@param*** *position2 coordonnées de la seconde position  
 \** ***@return*** *la distance entre 'position1' et 'position2'  
 \** ***@throws*** *Exception pouvant être lancée par l'API google maps  
 \*/* public static long Distance(LatLng position1, LatLng position2) throws Exception {  
 GeoApiContext context = new GeoApiContext().setApiKey(*GeoAPIKey*);  
  
 String[] origins = {position1.toUrlValue()};  
 String[] dest = {position2.toUrlValue()};  
 DistanceMatrix result = DistanceMatrixApi.*getDistanceMatrix*(context, origins, dest).await();  
  
 return result.rows[0].elements[0].distance.inMeters;  
 }  
  
 */\*\*  
 \* Retourne les coordonées d'un adresse donnée grâce à l'API google maps.  
 \** ***@param*** *address Addresse sous forme de chaine de charactères  
 \** ***@return*** *les coordonées de l'adresse  
 \** ***@throws*** *Exception pouvant être lancée par l'API google maps  
 \*/* public static LatLng GetLatLngFromAddress(String address) throws Exception {  
 GeoApiContext context = new GeoApiContext().setApiKey(*GeoAPIKey*);  
 GeocodingResult[] results = GeocodingApi.*geocode*(context, address).await();  
  
 return results[0].geometry.location;  
 }  
}

Services (metier.service)

package TP1\_SI.metier.service;  
  
import java.sql.Date;  
import java.util.List;  
  
import TP1\_SI.DAL.\*;  
import com.google.maps.model.LatLng;  
  
import TP1\_SI.metier.model.Event;  
import TP1\_SI.metier.model.Member;  
import TP1\_SI.metier.model.Activity;  
import TP1\_SI.metier.model.Location;  
  
*/\*\*  
 \* Classe regroupant les services metier de l'application COLLECT'IF.  
 \** ***@author*** *B3330  
 \*/*public class Services {  
  
 */\*\*  
 \* Enumération des codes d'erreur pouvant se produire lors de la connexion/inscription*

*\* d'un member  
 \*/* public enum ConnexionError {  
 *OK*, *EMPTY\_NAME*, *EMPTY\_FIRSTNAME*, *EMPTY\_ADDRESS*, *EMPTY\_EMAIL*, *WRONG\_ADDRESS*,

*BAD\_EMAIL*, *DATABASE\_ERROR* }  
  
 */\*\*  
 \* Enumération des codes d'erreur pouvant se produire lors de l'appel aux autres services*

*\* (voir type de retour dez méthodes ci-dessous)  
 \*/* public enum Request\_Error {  
 *OK*, *WRONG\_EVENT\_ID*, *WRONG\_LIEU\_ID*, *WRONG\_MEMBER\_ID*, *WRONG\_ACTIVITY\_ID*, *FULL\_EVENT*,

*DATABASE\_ERROR* }  
  
 */\*\*  
 \* Methode permettant la connexion d'un membre à partir de son email  
 \** ***@param*** *mail email du membre  
 \** ***@return*** *retourne le membre connecté ou un code d'erreur détaillant les raisons de*

*\* l'echec de la connexion (OK, DATABASE\_ERROR ou BAD\_EMAIL)  
 \*/* public static ServiceResult<Member, ConnexionError> Connexion(String mail) {  
 try {  
 MemberDAL adherent\_dao = new MemberDAL();  
 Member member = adherent\_dao.findByMail(mail);  
 if (member == null)  
 return new ServiceResult<Member, ConnexionError>(null,

ConnexionError.*BAD\_EMAIL*);  
 return new ServiceResult<Member, ConnexionError>(member, ConnexionError.*OK*);  
 } catch (Throwable e) {  
 return new ServiceResult<Member, ConnexionError>(null,

ConnexionError.*DATABASE\_ERROR*);  
 }  
 }  
  
 */\*\*  
 \* Inscrit un membre si possible.  
 \** ***@param*** *nom nom du membre  
 \** ***@param*** *prenom prénom du membre  
 \** ***@param*** *mail email du nouveau membre (doit être unique)  
 \** ***@param*** *address addresse du nouveau membre. L'API google maps doit pouvoir trouver les*

*\* coordonées de l'addresse pour que l'inscription ait lieu  
 \** ***@return*** *retourne le membre connecté ou un code d'erreur détaillant les raisons de*

*\* l'echec de l'inscription (OK, EMPTY\_FIRSTNAME, EMPTY\_ADDRESS,  
 \* EMPTY\_EMAIL, WRONG\_ADDRESS, BAD\_EMAIL, DATABASE\_ERROR ou WRONG\_ADDRESS).  
 \*/* public static ServiceResult<Member, ConnexionError> Inscription(String nom, String

prenom, String mail, String address) {  
 if(prenom == null || prenom.isEmpty())  
 return new ServiceResult<Member, ConnexionError>(null,

ConnexionError.*EMPTY\_FIRSTNAME*);  
 if(mail == null || mail.isEmpty())  
 return new ServiceResult<Member, ConnexionError>(null,

ConnexionError.*EMPTY\_EMAIL*);  
 if(nom == null || nom.isEmpty())  
 return new ServiceResult<Member, ConnexionError>(null,

ConnexionError.*EMPTY\_NAME*);  
 if(address == null || address.isEmpty())  
 return new ServiceResult<Member, ConnexionError>(null,

ConnexionError.*EMPTY\_ADDRESS*);  
  
 try {  
 LatLng coords = ServiceTechnique.*GetLatLngFromAddress*(address);  
 Member member = new Member(nom, prenom, address, mail, coords);  
 JpaUtil.*ouvrirTransaction*();  
 MemberDAL adherent\_dao = new MemberDAL();  
  
 try {  
 if (adherent\_dao.findByMail(mail) != null) {  
 JpaUtil.*annulerTransaction*();  
 return new ServiceResult<Member, ConnexionError>(null,

ConnexionError.*BAD\_EMAIL*);  
 }  
 adherent\_dao.create(member);  
 JpaUtil.*validerTransaction*();  
 ServiceTechnique.*SendSuccessfullInscriptionMail*(member);  
 return new ServiceResult<Member, ConnexionError>(member, ConnexionError.*OK*);  
 } catch (Throwable e) {  
 ServiceTechnique.*SendFailedInscriptionMail*(member);  
 JpaUtil.*annulerTransaction*();  
 return new ServiceResult<Member, ConnexionError>(null,

ConnexionError.*DATABASE\_ERROR*);  
 }  
 } catch (Exception e) {  
 JpaUtil.*annulerTransaction*();  
 return new ServiceResult<Member, ConnexionError>(null,

ConnexionError.*WRONG\_ADDRESS*);  
 }  
 }  
  
 */\*\*  
 \* Liste les événements d'une date données  
 \** ***@param*** *date date, au jour près, à la quelle ont cherche les événements  
 \** ***@return*** *la liste des événements ou un code d'erreur détaillant les raison pour les*

*\* quelles la méthode a échouée (OK, DATABASE\_ERROR)  
 \*/* public static ServiceResult<List<Event>, Request\_Error> ListEventsOfDate(Date date) {  
 EventDAL event\_dao = new EventDAL();  
 try {  
 return new ServiceResult<List<Event>, Request\_Error>(event\_dao.findByDate(date),

Request\_Error.*OK*);  
 } catch (Throwable e) {  
 return new ServiceResult<List<Event>, Request\_Error>(null,

Request\_Error.*DATABASE\_ERROR*);  
 }  
 }  
  
 */\*\*  
 \* Liste tout les évenements  
 \** ***@return*** *la liste des événements ou un code d'erreur détaillant les raison pour les*

*\* quelles la méthode a échouée (OK, DATABASE\_ERROR)  
 \*/* public static ServiceResult<List<Event>, Request\_Error> ListAllEvents() {  
 try {  
 EventDAL event\_dao = new EventDAL();  
 return new ServiceResult<List<Event>, Request\_Error>(event\_dao.findAll(),

Request\_Error.*OK*);  
 } catch (Throwable e) {  
 return new ServiceResult<List<Event>, Request\_Error>(null,

Request\_Error.*DATABASE\_ERROR*);  
 }  
 }  
  
 */\*\*  
 \* Assigne un lieu (location) à un évenement  
 \** ***@param*** *location\_id id du lieu  
 \** ***@param*** *event\_id id de l'évenement au quel assigner le lieu  
 \** ***@return*** *Un code d'erreur indiquant si l'assignation s'est déroullée correctement*

*\* (DATABASE\_ERROR, WRONG\_LIEU\_ID, WRONG\_EVENT\_ID ou OK)  
 \*/* public static Request\_Error AssignLocationToEvent(long location\_id, long event\_id) {  
 try {  
 EventDAL event\_dao = new EventDAL();  
 Event event = event\_dao.findById(event\_id);  
 if (event == null)  
 return Request\_Error.*WRONG\_EVENT\_ID*;  
  
 LocationDAL lieu\_dao = new LocationDAL();  
 Location location = lieu\_dao.findById(location\_id);  
 if (location == null)  
 return Request\_Error.*WRONG\_LIEU\_ID*;  
  
 JpaUtil.*ouvrirTransaction*();  
 event.setLocation(location);  
 event\_dao.update(event);  
 if (event.getComplet())  
 ServiceTechnique.*SendEventMail*(event);  
  
 JpaUtil.*validerTransaction*();  
 return Request\_Error.*OK*;  
 } catch (Throwable e) {  
 JpaUtil.*annulerTransaction*();  
 return Request\_Error.*DATABASE\_ERROR*;  
 }  
 }  
  
 */\*\*  
 \* Methode permettant à un membre de rejoindre un évenement  
 \** ***@param*** *member\_id id du membre  
 \** ***@param*** *event\_id id de l'évenement à rejoindre  
 \** ***@return*** *Un code d'erreur indiquant si le membre a pû rejoindre l'évenement*

*\* (DATABASE\_ERROR, FULL\_EVENT, WRONG\_EVENT\_ID, WRONG\_MEMBER\_ID ou OK)  
 \*/* public static Request\_Error JoinEvent(long member\_id, long event\_id) {  
 try {  
 EventDAL event\_dao = new EventDAL();  
 Event event = event\_dao.findById(event\_id);  
 if (event == null)  
 return Request\_Error.*WRONG\_EVENT\_ID*;  
 if (event.getComplet())  
 return Request\_Error.*FULL\_EVENT*;  
  
 MemberDAL member\_dao = new MemberDAL();  
 Member member = member\_dao.findById(member\_id);  
 if (member == null)  
 return Request\_Error.*WRONG\_MEMBER\_ID*;  
  
 JpaUtil.*ouvrirTransaction*();  
 event\_dao.AddAdherentToEvent(event, member);  
 if (event.getComplet())  
 ServiceTechnique.*SendEventMail*(event);  
  
 JpaUtil.*validerTransaction*();  
 return Request\_Error.*OK*;  
 } catch (Throwable e) {  
 JpaUtil.*annulerTransaction*();  
 return Request\_Error.*DATABASE\_ERROR*;  
 }  
 }  
  
 */\*\*  
 \* Crée un évenement au quel participera son créateur  
 \** ***@param*** *member\_id id du créateur de l'évenement  
 \** ***@param*** *activity\_id id de l'activité de l'évenement à creer  
 \** ***@param*** *date data à laquelle auras lieu l'evenement  
 \** ***@return*** *L'évenement crée ou un code d'erreur (OK, WRONG\_ACTIVITY\_ID, WRONG\_MEMBER\_ID*

*\* ou DATABASE\_ERROR)  
 \*/* public static ServiceResult<Event, Request\_Error> CreateEvent(long member\_id, long

activity\_id, Date date) {  
 try {  
 ActivityDAL activity\_dao = new ActivityDAL();  
 Activity activity = activity\_dao.findById(activity\_id);  
 if (activity == null)  
 return new ServiceResult<Event, Request\_Error>(null,

Request\_Error.*WRONG\_ACTIVITY\_ID*);  
  
 MemberDAL member\_dao = new MemberDAL();  
 Member member = member\_dao.findById(member\_id);  
 if (member == null)  
 return new ServiceResult<Event, Request\_Error>(null,

Request\_Error.*WRONG\_MEMBER\_ID*);  
  
 JpaUtil.*ouvrirTransaction*();  
 EventDAL event\_dao = new EventDAL();  
 Event event = new Event(date, activity, member);  
 event\_dao.create(event);  
 JpaUtil.*validerTransaction*();  
  
 return new ServiceResult<Event, Request\_Error>(event, Request\_Error.*OK*);  
 } catch (Throwable e) {  
 JpaUtil.*annulerTransaction*();  
 return new ServiceResult<Event, Request\_Error>(null,

Request\_Error.*DATABASE\_ERROR*);  
 }  
 }  
 */\*\*  
 \* Liste les évenements aux quels participe un membre donné en paramètre  
 \** ***@param*** *member\_id id du membre  
 \** ***@return*** *La liste des évnenements ou un code d'erreur (OK ou DATABASE\_ERROR) (renvoie*

*\* DATABASE\_ERROR si l'id du membre est mauvais)  
 \*/* public static ServiceResult<List<Event>, Request\_Error> ListEventsOfMember(long

member\_id) {  
 try {  
 EventDAL event\_dao = new EventDAL();  
 List<Event> events = event\_dao.findByMember(member\_id);  
 return new ServiceResult<List<Event>, Request\_Error>(events, Request\_Error.*OK*);  
 } catch (Throwable e) {  
 return new ServiceResult<List<Event>, Request\_Error>(null,

Request\_Error.*DATABASE\_ERROR*);  
 }  
 }  
  
 */\*\*  
 \* Liste les évenements qui ne sont pas complets (évenements pouvant acceuillir encore*

*\* des participants)  
 \** ***@return*** *La liste des évnenements ou un code d'erreur (OK ou DATABASE\_ERROR)  
 \*/* public static ServiceResult<List<Event>, Request\_Error> ListAvailableEvents() {  
 try {  
 EventDAL event\_dao = new EventDAL();  
 return new ServiceResult<List<Event>, Request\_Error>(event\_dao.findDispo(),

Request\_Error.*OK*);  
 } catch (Throwable e) {  
 return new ServiceResult<List<Event>, Request\_Error>(null,

Request\_Error.*DATABASE\_ERROR*);  
 }  
 }  
  
 */\*\*  
 \* Donne la liste des activitées possibles  
 \** ***@return*** *La liste de toutes les activitées ou un code d'erreur (OK ou DATABASE\_ERROR)  
 \*/* public static ServiceResult<List<Activity>, Request\_Error> ListActivities() {  
 try {  
 ActivityDAL activity\_dao = new ActivityDAL();  
 return new ServiceResult<List<Activity>, Request\_Error>(activity\_dao.findAll(),

Request\_Error.*OK*);  
 } catch (Throwable e) {  
 return new ServiceResult<List<Activity>, Request\_Error>(null,

Request\_Error.*DATABASE\_ERROR*);  
 }  
 }  
  
 */\*\*  
 \* Donne la liste de tout les lieux possibles  
 \*/* public static ServiceResult<List<Location>, Request\_Error> ListLocations() {  
 try {  
 LocationDAL Location\_dao = new LocationDAL();  
 return new ServiceResult<List<Location>, Request\_Error>(Location\_dao.findAll(),

Request\_Error.*OK*);  
 } catch (Throwable e) {  
 return new ServiceResult<List<Location>, Request\_Error>(null,

Request\_Error.*DATABASE\_ERROR*);  
 }  
 }  
}

ConsoleConnexionView (View)

package TP1\_SI.View;  
  
import TP1\_SI.Utils.Callable;  
import TP1\_SI.Utils.ConsoleMenu;  
import TP1\_SI.Utils.ConsoleUtils;  
import TP1\_SI.metier.model.Member;  
import TP1\_SI.metier.service.Services;  
import TP1\_SI.metier.service.ServiceResult;  
import TP1\_SI.metier.service.Services.ConnexionError;  
  
import java.util.Scanner;  
  
*/\*\*  
 \* Classe gèrant la vue console de connexion.  
 \* L'utilisateur peut y creer un compte ou se connecter avec une compte existant.  
 \** ***@author*** *B3330  
 \*/*public class ConsoleConnexionView {  
 static final String *admin\_email* = "admin.collectif@insa-lyon.fr";  
  
 */\*\*  
 \* Lance la vue  
 \*/* public void run() {  
 ConsoleMenu connexion\_menu = new ConsoleMenu("CONNEXION MENU", new Quit());  
 connexion\_menu.addChoice(new Login(), "Log in");  
 connexion\_menu.addChoice(new Signin(), "Sign in");  
  
 connexion\_menu.runMenu();  
 }  
  
 */\*\*  
 \* Callback gèrant la connexion avezc un coimpte existant  
 \*/* private static class Login implements Callable  
 {  
 @Override  
 public void call() {  
 System.*out*.println("### Connexion ###");  
 String email = ConsoleUtils.*lireChaine*("Entrez votre mail : ");  
 if(!*admin\_email*.equals(email))  
 {  
 ServiceResult<Member, ConnexionError> newConnexion =

Services.*Connexion*(email);  
 if (newConnexion.error == ConnexionError.*OK*)  
 *goto\_espace\_adherents*(newConnexion.result);  
 else  
 // *TODO: gèrer l'erreur en fonction de 'newConnexion.error'* System.*out*.println("Erreur de connexion : " + newConnexion.error);  
 }  
 else  
 *goto\_dashboard*();  
 }  
 }  
  
 */\*\*  
 \* Callback gèrant la création de comptes  
 \*/* private static class Signin implements Callable  
 {  
 @Override  
 public void call() {  
 System.*out*.println("### Inscription ###");  
 String nom = ConsoleUtils.*lireChaine*("Entrez votre Nom : ");  
 String prenom = ConsoleUtils.*lireChaine*("Entrez votre Prenom : ");  
 String email = ConsoleUtils.*lireChaine*("Entrez votre Email : ");  
 String address = ConsoleUtils.*lireChaine*("Entrez votre Adresse : ");  
 if(!*admin\_email*.equals(email))  
 {  
 ServiceResult<Member, ConnexionError> newInscription =

Services.*Inscription*(nom, prenom, email, address);  
 if (newInscription.error == ConnexionError.*OK*)  
 *goto\_espace\_adherents*(newInscription.result);  
 else  
 // *TODO: gèrer l'erreur en fonction de 'newInscription.error'* System.*out*.println("Erreur d'inscription : " + newInscription.error);  
 }  
 else  
 System.*out*.println("Erreur d'inscription : " + ConnexionError.*BAD\_EMAIL*);  
 }  
 }  
  
 */\*\*  
 \* Callback appelée si l'utilisateur veux quitter l'application  
 \*/* private static class Quit implements Callable {  
 @Override  
 public void call() {  
 System.*out*.println("\n\nExiting application...");  
 }  
 }  
  
 */\*\*  
 \* Navigue vers la vue 'espace adherent' (MemberView)  
 \** ***@param*** *logged\_member Membre s'étant coinnecté avec succès  
 \*/* private static void goto\_espace\_adherents(Member logged\_member) {  
 System.*out*.println("##########################################\n");  
 ConsoleMemberView new\_view = new ConsoleMemberView(logged\_member);  
 new\_view.run();  
 }  
  
 */\*\*  
 \* Navigue vers la vue dashboard de l'administrateur  
 \*/* private static void goto\_dashboard() {  
 System.*out*.println("##########################################\n");  
 ConsoleDashboardView new\_view = new ConsoleDashboardView();  
 new\_view.run();  
 }  
}

ConsoleMemberView (View)

package TP1\_SI.View;  
  
import TP1\_SI.Utils.Callable;  
import TP1\_SI.Utils.ConsoleMenu;  
import TP1\_SI.Utils.ConsoleUtils;  
import TP1\_SI.metier.model.Activity;  
import TP1\_SI.metier.model.Event;  
import TP1\_SI.metier.model.Member;  
import TP1\_SI.metier.service.Services;  
import TP1\_SI.metier.service.ServiceResult;  
import TP1\_SI.metier.service.Services.Request\_Error;  
  
import java.sql.Date;  
import java.util.List;  
  
*/\*\*  
 \** ***@author*** *B3330  
 \*/*public class ConsoleMemberView {  
 public ConsoleMemberView(Member logged\_member) {  
 this.logged\_member = logged\_member;  
 }  
  
 */\*\*  
 \* Lance la vue  
 \*/* public void run() {  
 ConsoleMenu connexion\_menu = new ConsoleMenu("ESPACE ADHERENT", new Logout());  
  
 connexion\_menu.addChoice(new SeeEventsList(), "Voir la liste des événements");  
 connexion\_menu.addChoice(new CreateEvent(), "Créer un événement");  
 connexion\_menu.addChoice(new JoinEvent(), "Rejoindre un événement");  
  
 connexion\_menu.runMenu();  
 }  
  
 */\*\*  
 \* Callback permettant d'afficher la liste des événements qui ne sont pas complets et la liste des*

*\* événements aux quels participe le membre courant  
 \*/* private class SeeEventsList implements Callable {  
 @Override  
 public void call() {  
 ServiceResult<List<Event>, Request\_Error> available\_events\_rslt =

Services.*ListAvailableEvents*();  
 ServiceResult<List<Event>, Request\_Error> member\_events\_rslt =

Services.*ListEventsOfMember*(logged\_member.getId());  
  
 if(available\_events\_rslt.error == Request\_Error.*OK* && member\_events\_rslt.error ==

Request\_Error.*OK*) {  
 System.*out*.println("### Evenements que vous avez rejoint ###");  
 List<Event> member\_events = member\_events\_rslt.result;  
 for(int i = 0; i < member\_events.size(); ++i)  
 System.*out*.println("\t- " + member\_events.get(i));  
  
 System.*out*.println("### Evenements disponnibles (pas encore complets) ###");  
 List<Event> available\_events = available\_events\_rslt.result;  
 for(int i = 0; i < available\_events.size(); ++i)  
 System.*out*.println("\t- " + available\_events.get(i));  
 }  
 else  
 // *TODO: gèrer l'erreur en fonction de 'available\_events\_rslt.error' ou*

*// 'member\_events\_rslt.error'* System.*out*.println("Erreur lors de l'execution du service : " +

(member\_events\_rslt.error == Request\_Error.*OK* ? available\_events\_rslt.error : member\_events\_rslt.error));  
 }  
 }  
  
 */\*\*  
 \* Callback permetant au membre de crer un événement  
 \*/* private class CreateEvent implements Callable {  
 @Override  
 public void call() {  
 System.*out*.println("### Création d'événement ###");  
  
 // Affiche la liste des activités disponnibles  
 ServiceResult<List<Activity>, Request\_Error> activities\_list\_rslt =

Services.*ListActivities*();  
 if(activities\_list\_rslt.error == Request\_Error.*OK*)  
 {  
 List<Activity> activities = activities\_list\_rslt.result;  
 for(int i = 0; i < activities.size(); ++i)  
 System.*out*.print((i == 0 ? "\tActivités disponnibles : " : ", \n\t\t") +

activities.get(i));  
 }  
 else  
 // *TODO: gèrer l'erreur en fonction de 'activities\_list\_rslt.error'* System.*out*.println("Erreur lors de l'execution du service : " +

activities\_list\_rslt.error);  
  
 // Demmande l'activité de l'événement et crée l'évenement  
 long activity\_id = ConsoleUtils.*lireInteger*("\nEntrez l'id de l'activité : ");  
 Date today = new java.sql.Date((new java.util.Date()).getTime()); // La date de l'évenement

// est la date actuelle  
 ServiceResult<Event, Request\_Error> event\_creation\_rslt =

Services.*CreateEvent*(logged\_member.getId(), activity\_id, today);  
  
 if(event\_creation\_rslt.error == Request\_Error.*OK*)  
 System.*out*.println("Evenement créé avec succès : " + event\_creation\_rslt.result);  
 else  
 // *TODO: gèrer l'erreur en fonction de 'event\_creation\_rslt.error'* System.*out*.println("Erreur lors de l'execution du service:"+ event\_creation\_rslt.error);  
 }  
 }  
  
 */\*\*  
 \* Callback permettant au membre de rejoindre un événement  
 \*/* private class JoinEvent implements Callable {  
 @Override  
 public void call() {  
 long id = ConsoleUtils.*lireInteger*("Entrez l'id de l'évenement que vous voulez rejoindre:");  
  
 Request\_Error join\_event\_rslt = Services.*JoinEvent*(logged\_member.getId(), id);  
  
 if(join\_event\_rslt == Request\_Error.*OK*)  
 System.*out*.println("Evenement rejoint avec succès.");  
 else  
 // *TODO: gèrer l'erreur en fonction de 'join\_event\_rslt'* System.*out*.println("Erreur lors de l'execution du service : " + join\_event\_rslt);  
 }  
 }  
  
 */\*\*  
 \* Callback appelée si l'utilisateur veux se déconnecter (retourner à la vue de connexion)  
 \*/* private class Logout implements Callable {  
 @Override  
 public void call() {  
  
 }  
 }  
  
 private Member logged\_member;  
}

ConsoleDashboardView (View)

package TP1\_SI.View;  
  
import TP1\_SI.Utils.Callable;  
import TP1\_SI.Utils.ConsoleMenu;  
import TP1\_SI.Utils.ConsoleUtils;  
import TP1\_SI.metier.model.Event;  
import TP1\_SI.metier.model.Location;  
import TP1\_SI.metier.service.ServiceResult;  
import TP1\_SI.metier.service.Services;  
  
import java.util.List;  
  
*/\*\*  
 \* Vue de l'administrateur assignat les lieux aux evenements  
 \** ***@author*** *B3330  
 \*/*public class ConsoleDashboardView {  
  
 */\*\*  
 \* Lance la vue  
 \*/* public void run() {  
 ConsoleMenu connexion\_menu = new ConsoleMenu("DASHBOARD", new Logout());  
 connexion\_menu.addChoice(new AssignLocationToEvent(), "Assign location to event");  
  
 connexion\_menu.runMenu();  
 }  
  
 */\*\*  
 \* Callback gèrant l'assingation d'un lieu à un evenement  
 \*/* private static class AssignLocationToEvent implements Callable {  
 @Override  
 public void call() {  
 ServiceResult<List<Event>, Services.Request\_Error> events\_rslt = Services.*ListAllEvents*();  
 ServiceResult<List<Location>, Services.Request\_Error> locations\_rslt =

Services.*ListLocations*();  
  
 if(events\_rslt.error == Services.Request\_Error.*OK* && locations\_rslt.error ==

Services.Request\_Error.*OK* ) {  
 System.*out*.println("### Liste de tout les évenements ###");  
 List<Event> events = events\_rslt.result;  
 for(int i = 0; i < events.size(); ++i)  
 System.*out*.println("\t- " + events.get(i));  
  
 System.*out*.println("\n### Liste de tout les lieux ###");  
 List<Location> locations = locations\_rslt.result;  
 for(int i = 0; i < locations.size(); ++i)  
 System.*out*.println("\t- " + locations.get(i));  
  
 long event\_id = ConsoleUtils.*lireInteger*("\nEntrez l'id de l'évenement : ");  
 long lieu\_id = ConsoleUtils.*lireInteger*("Entrez l'id du lieu : ");  
  
 Services.Request\_Error error\_code = Services.*AssignLocationToEvent*(lieu\_id, event\_id);  
 if(error\_code == Services.Request\_Error.*OK*)  
 System.*out*.println("Assignation réussie.");  
 else  
 System.*out*.println("Assignation échouée.");  
 }  
 else  
 // *TODO: gèrer l'erreur en fonction de 'events\_rslt.error' ou 'locations\_rslt.error'* System.*out*.println("Erreur lors de l'execution du service : " + (events\_rslt.error ==

Services.Request\_Error.*OK* ? locations\_rslt.error : events\_rslt.error) );  
 }  
 }  
  
 */\*\*  
 \* Callback appelé si l'utilisateur veux retourner à la vue de connexion  
 \*/* private static class Logout implements Callable {  
 @Override  
 public void call() {  
  
 }  
 }  
}

FIN