

DÉVELOPPEMENT DE SITE WEB EATOGETHER

UNITÉ D'ENSEIGNEMENT : DAR

Réalisé par

**THEYRI DARINE & BEN JEMIA MOHAMED YASSINE & HAMIDI
ZAKARIA**

Compte GitHub : <https://github.com/yassinebj3/Eatogether> ; **branche** : master

URL de site web : <https://www.eatogether.ml>

Table des matières

I. Présentation de l'application	3
1. Notre application Eatotogether	3
2. Les services de notre site	3
II. Manuel d'utilisation	4
1. Raffinement de cas d'utilisation « Gérer un compte utilisateur»	4
a. Description textuelle de la tâche coté utilisateur «Créer compte»	4
b. Déroulement de la tâche coté serveur «Créer compte»	5
2. Raffinement de cas d'utilisation « S'authentifier »	5
a. Description textuelle de la tâche «Se connecter avec mon compte Facebook» coté utilisateur	5
b. Déroulement de la tâche «S'authentifier» coté serveur	5
3. Raffinement de cas d'utilisation <i>Consulter les restaurants</i>	6
a. Description textuelle de la tâche «Consulter la liste des restaurants correspondante à mes critères» coté utilisateur	6
b. Déroulement de la tâche «Consulter la liste des restaurants selon mes critères» coté serveur	7
4. Raffinement de cas d'utilisation « Gérer un rendez-vous »	7
a. Description textuelle de la tâche « Créer un rendez-vous » coté utilisateur	7
b. Description textuelle de la tâche coté utilisateur « Matcher un rendez-vous »	8
c. Déroulement de la tâche « Matcher un rendez-vous et Créer un rendez-vous » coté serveur	8
5. Raffinement de cas d'utilisation « Interagir avec mes Matches »	9
a. Description textuelle de la tâche « Discuter avec mon match dans une conversation instantanée » coté utilisateur	9
b. Déroulement de la tâche « Discuter avec mon match dans une conversation instantanée » coté serveur	9
III. Architecture et réalisation de notre application Web	10
1. Architecture Matérielle de l'application	10
2. Architecture Logicielle de l'application	10
3. Les APIs et les technologies utilisées	12
a. Les APIs utilisées	12
b. Le choix des technologies	14
4. Hébergement de l'application	18
IV. Points forts et points faibles de notre application	19
1. Points forts :	19
2. Points faibles :	19
V. Extensions et difficultés rencontrées	19

1. Les extensions possibles.....	19
2. Difficultés rencontrées	19
Conclusion.....	21

Vous ne voulez pas manger seule ? Eatotogether, un réseau social qui vous permet de créer des souvenirs positifs en partageant un bon repas ... en bonne compagnie !

I. Présentation de l'application

1. Notre application Eatotogether

La finalité de l'unité d'enseignement " DAR " est la mise en place d'une application réticulaire en se basant sur une " API REST " et en utilisant les technologies web pour la création des applications. Dans ce contexte, notre projet " Eatotogether " consiste à mettre en place une plateforme de rencontre qui permet aux utilisateurs d'organiser des rendez-vous avec d'autres personnes en partageant un repas dans un cadre conviviale.

Notre application offre un système de recherche qui se base sur l'API "Foursquare" permettant aux utilisateurs de chercher leurs lieux préférés et consulter leurs détails.

Ainsi "Eatotogether" permet à ses utilisateurs de créer leurs propres rendez-vous ou alors, joindre des rendez-vous existants et communiquer entre eux à travers un module de Chat

2. Les services de notre site

Notre site a comme but d'offrir un moyen pour créer des liens entre les différents utilisateurs, dans un contexte convivial. En fait, l'idée de l'application tourne autour de cinq majeur user story :

- A. Comme étant un utilisateur je veux gérer un compte utilisateur
- B. Comme étant un utilisateur je veux m'authentifier sur mon compte afin de consulter les services de sites
- C. Comme étant un utilisateur je veux consulter les restaurants
- D. Comme étant un utilisateur je veux gérer un rendez-vous dans un restaurant
- E. Comme étant un utilisateur je veux interagir avec mes matchs¹

Chaque user story mentionner au-dessus englobe un nombre des taches qui sont également :

A	Créer mon compte utilisateur
	Modifier mon compte utilisateur
	Supprimer mon compte utilisateur
	Consulter mon compte utilisateur
B	S'authentifier à mon compte avec mes identifiants
	S'authentifier avec mon compte Facebook
C	Consulter la liste des restaurants correspondants aux mes critères
	Consulter les détails d'un restaurant choisi
D	Créer un rendez-vous
	Matcher un rendez-vous
	Modifier la date de rendez-vous
	Annuler un rendez-vous
	Consulter tous les rendez-vous
	Consulter mes rendez-vous

¹ Un match c'est la personne avec qui vous allez sortir en rendez vous

E	Discuter avec mon match dans une conversation instantanée
---	---

Tableau 1: Description des users stories

II. Manuel d'utilisation

Dans cette partie nous allons détaillés les user story les plus pertinentes.

1. Raffinement de cas d'utilisation « Gérer un compte utilisateur»

a. Description textuelle de la tâche «Créer compte» coté utilisateur

tâche	Créer compte
Acteurs	Utilisateur
Précondition	Utilisateur n'admet pas de compte avec son adresse mail
Post-condition	Compte utilisateur créé
Scénario principal	<ul style="list-style-type: none"> ▪ L'acteur appuie sur le lien « Create an account » ▪ L'application affiche le formulaire d'inscription ▪ L'acteur remplit les champs avec ses coordonnées ▪ L'acteur clique sur le bouton « Inscription » ▪ L'application redirige l'utilisateur sur l'interface de connexion
Scénario alternatif	L'application affiche un message d'alerte si un des champs de formulaire est vide

Tableau 2:Description détaillées de la tâche "Créer compte"

Figure 1:IHM de la création d'un compte

b. Déroulement de la tâche «Créer compte» coté serveur

La création un nouveau compte utilisateur se fait à travers un formulaire. Les informations seront transmises à la partie Contrôleur définit par le servlet de connexion. Ainsi le contrôleur récupère les informations et invoquent les méthodes nécessaires depuis la couche Business pour la persistance de donnée. Le mot de passe doit être crypté puis inséré dans la Base de données.

2. Raffinement de cas d'utilisation « S'authentifier »

a. Description textuelle de la tâche «Se connecter avec mon compte Facebook» coté utilisateur

tâche	Se connecter avec mon compte Facebook
Acteurs	Utilisateur
Précondition	Utilisateur admet un compte Facebook et connecté sur sa session
Post-condition	Connexion acceptée
Scénario principal	<ul style="list-style-type: none">▪ L'acteur appuie sur le bouton « Continue with Facebook » ou « Continue as ... » s'il est déjà connecté sur son navigateur▪ L'application affiche un pop-up de la connexion vers Facebook▪ L'acteur confirme sa connexion avec son compte Facebook
Scénario alternatif	L'utilisateur ne peut pas se connecter que quand l'administrateur de site a accepté cette connexion

Tableau 3:Description détaillé de la tache "Se connecter avec mon compte utilisateur"



Figure 2:IHM de l'authentification

b. Déroulement de la tâche «S'authentifier» coté serveur

En se connectant avec Facebook, le contrôleur appelle les méthodes du model à fin vérifier l'existence de l'email dans la base de données. En cas d'absence d'information, une requête http est envoyée à l'API Facebook qui renvoie un objet JSON avec quelques informations sur l'utilisateur. Les informations

extraites et décapsulés vont par la suite être insérées dans la base de données. En cas de succès, une session utilisateur sera créée.

3. Raffinement de cas d'utilisation « Consulter les restaurants »

a. Description textuelle de la tâche « Consulter la liste des restaurants correspondante à mes critères » coté utilisateur

tâche	Consulter la liste des restaurants correspondante à mes critères
Acteurs	Utilisateur
Précondition	Utilisateur authentifié
Post-condition	Liste des restaurants selon les choix de l'utilisateur affichée
Scénario principal	<ul style="list-style-type: none"> ▪ L'acteur Choisit un emplacement de restaurant ▪ L'acteur choisit le type de restauration ▪ L'acteur choisit le rayon et le nombre de résultat ▪ L'acteur clique sur le bouton « Recherche » ▪ L'application affiche à l'utilisateur la liste des restaurants correspondant aux critères spécifiés
Scénario alternatif	L'application affiche un message d'alerte si l'utilisateur a oublié de remplir un champ du formulaire

Tableau 4:Description détaillé de la tâche " Consulter la liste des restaurants correspondante à mes critères "

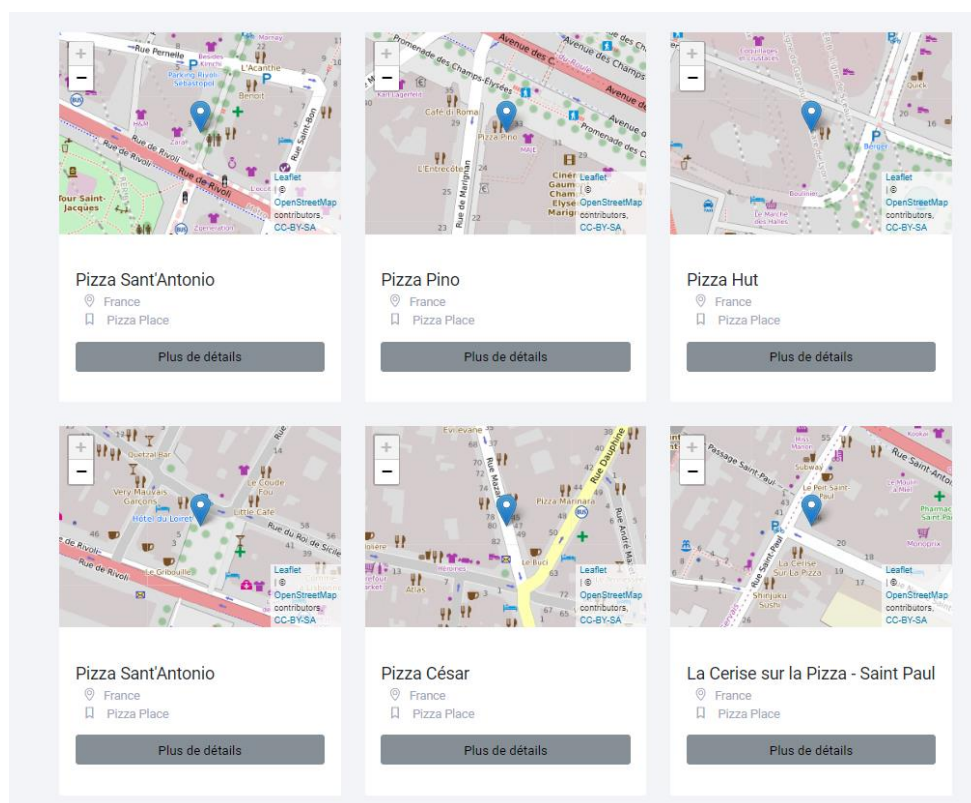


Figure 3:IHM de la page des resultats de ma recherche

b. *Déroulement de la tâche « Consulter la liste des restaurants selon mes critères » coté serveur*

Le formulaire de recherche présenté dans la page principale de l'application permet d'invoquer l'API **Foursquare**, via des requêtes HTTP en précisant des paramètres de la recherche souhaitée. La réponse en JSON est dé-sérialisée dans la partie contrôleur et envoyé à la page JSP pour traiter l'affichage de la liste des restaurants. Nous effectuons aussi un appel à l'api **Openstreetmap**, en passant des informations sur la localisation géographique obtenues de l'api **Foursquare**.

4. Raffinement de cas d'utilisation « *Gérer un rendez-vous* »

a. *Description textuelle de la tâche « Créer un rendez-vous » coté utilisateur*

tâche	Créer un rendez-vous
Acteurs	Utilisateur
Précondition	Utilisateur authentifié et le restaurant est choisi
Post-condition	Rendez-vous créé
Scénario principal	<ul style="list-style-type: none">▪ L'acteur appuie sur le bouton « Plus des détails » pour un restaurant spécifique▪ L'application affiche la page avec les détails du restaurant▪ L'acteur clique sur le bouton « Créer un rendez-vous »▪ L'application affiche un formulaire avec les différents champs à remplir (date, description)▪ L'acteur remplit les champs et clique sur le bouton « Confirmer la création »▪ L'application crée le rendez-vous et redirige l'utilisateur sur l'interface d'accueil
Scénario alternatif	L'application affiche un message d'alerte si l'utilisateur ne précise pas la date de rendez-vous

Tableau 5 : Description détaillée de la tâche " Créer un rendez-vous "

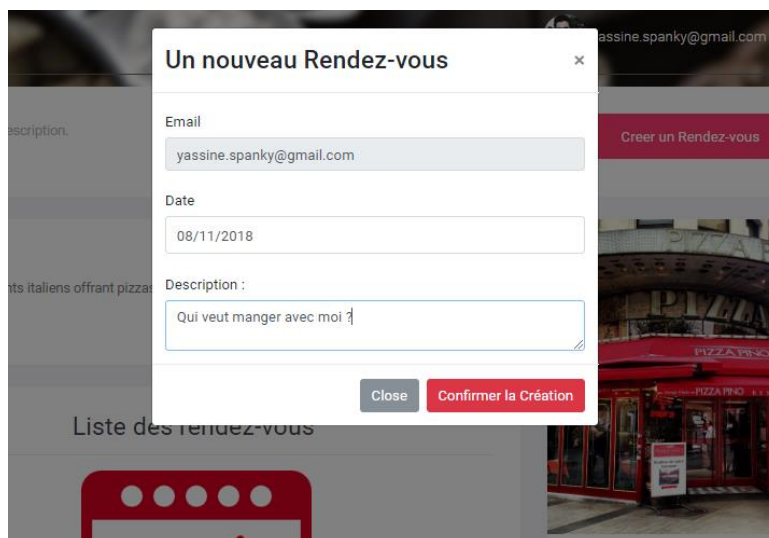


Figure 4: IHM de la création d'un rendez-vous

b. *Description textuelle de la tâche coté utilisateur « Matcher un rendez-vous »*

tâche	Matcher un rendez-vous
Acteurs	Utilisateur
Précondition	Rendez-vous existant
Post-condition	Rendez-vous matché
Scénario principal	<ul style="list-style-type: none"> ▪ L'acteur appuie sur le bouton « Match » pour un rendez-vous spécifique ▪ L'application rajoute ce rendez dans la rubrique « Mes rendez-vous »

Tableau 6 : Description détaillé de la tâche " Matcher un rendez-vous "

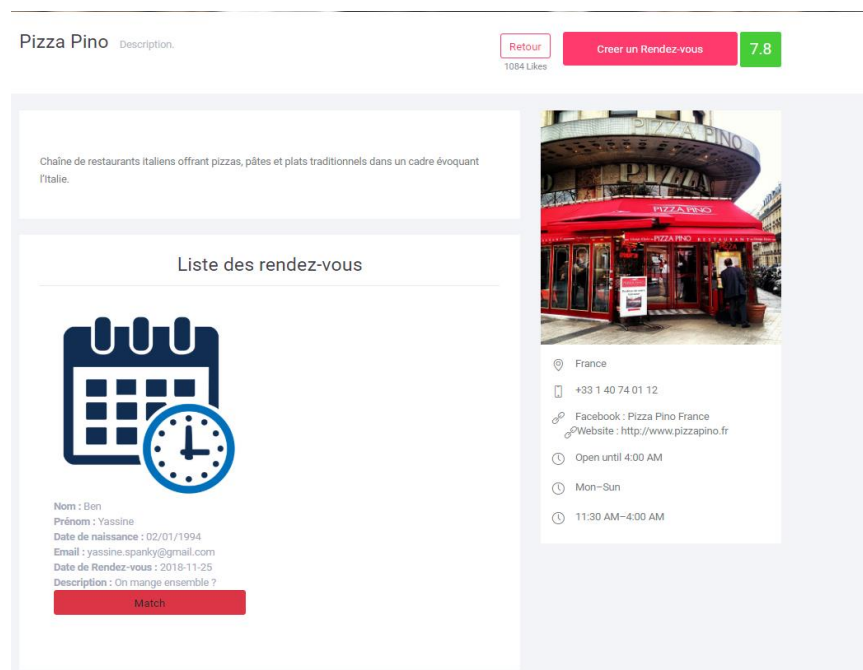


Figure 5: IHM de la création et le matching entre utilisateurs

c. *Déroulement de la tâche « Matcher un rendez-vous et Créer un rendez-vous » coté serveur*

Les informations remplies dans le formulaire de création vont être transmises au contrôleur qui fait appel à la méthode de la création de rendez-vous de la couche Business afin de persister les données. Aussi pour l'évènement matché un rendez-vous la logique consiste à mettre à jour le rendez-vous créé précédemment avec les informations de partenaire en appelant la méthode correspondante encore une fois de la couche business.

5. Raffinement de cas d'utilisation « Interagir avec mes Matches »

- a. Description textuelle de la tâche « Discuter avec mon match dans une conversation instantanée » coté utilisateur

tâche	Discuter avec mon match dans une conversation instantanée
Acteurs	Utilisateur
Précondition	Rendez-vous accepté
Post-condition	Interaction entre les utilisateurs
Scénario principal	<ul style="list-style-type: none">▪ L'application suite à l'acceptation de rendez-vous met les deux utilisateurs dans une liste d'amis▪ L'acteur clique sur l'utilisateur qui veut interagir avec et formule son message▪ L'acteur appuie sur le bouton d'envoi▪ L'application prend en charge la transmission de message

Tableau 7: Description détaillé de la tâche " Discuter avec mon match dans une conversation instantanée »

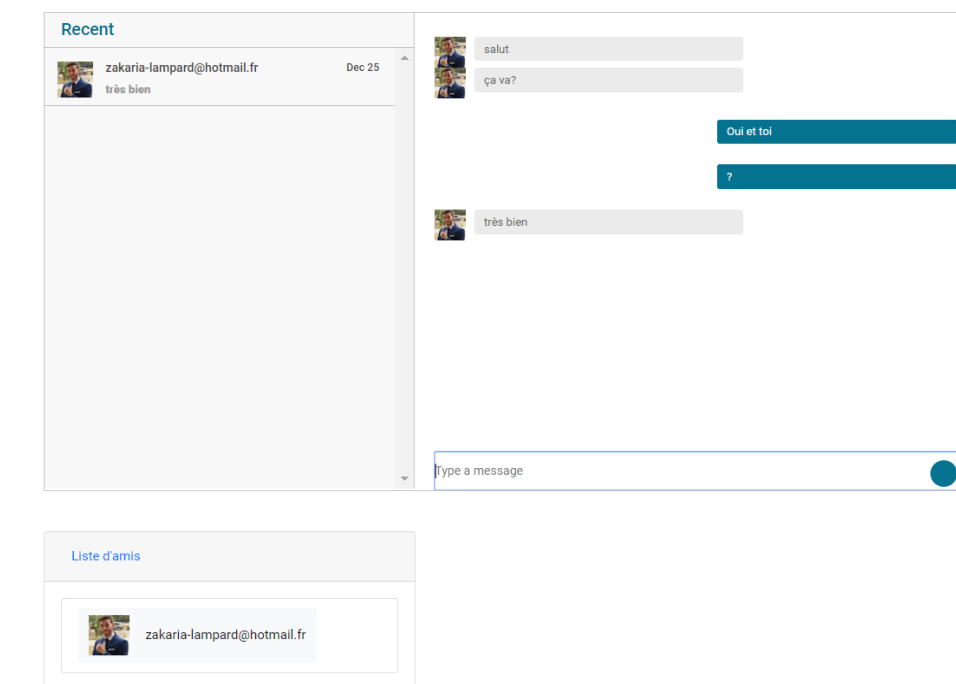


Figure 6: IHM du module Chat

- b. Déroulement de la tâche « Discuter avec mon match dans une conversation instantanée » coté serveur

Les utilisateurs peuvent échanger des messages instantanés entre eux. Afin de s'assurer que le destinataire reçoit les messages, notre chat stocke le message si le destinataire est déconnecté. Nous avons défini un seuil de 3 messages par personnes afin de limiter l'accès à la base de données. Lorsque l'utilisateur consulte le message, une requête AJAX permet d'appeler le

contrôleur responsable de la suppression qui s'en charge à invoquer les méthodes qui permettent la suppression des messages de l'expéditeur. Une fois les deux utilisateurs sont connectés, ils peuvent partager leurs messages à travers le web socket.

III. Architecture et réalisation de notre application Web

1. Architecture Matérielle Client-Serveur

Nous avons utilisé une architecture client-serveur représentée dans la figure ci-dessous. Le client qui représente le navigateur et le serveur qui correspond à la machine virtuelle Bitnami Tomcat. Le client envoie une requête HTTP à notre serveur web qui redirige la requête vers le serveur d'application Tomcat. À la réception de la requête, Tomcat présente au client la vue représentée par les pages JSP de l'application. Les pages JSP servent à définir une interface IHM afin d'interagir avec le client et appeler la couche contrôleurs. Les Servlets reçoivent les requêtes POST ou GET et décident d'invoquer des méthodes qui permettent d'appeler l'API Foursquare afin d'avoir des informations sur les lieux préférés ou bien des méthodes qui permettent d'assurer la persistance de donnée. Les servlets reçoivent la réponse HTTP de l'API sous forme de JSON contenant les détails d'un lieu. Le serveur d'application utilise l'ORM Hibernate afin d'assurer la persistance de donnée dans la base de données MySQL.

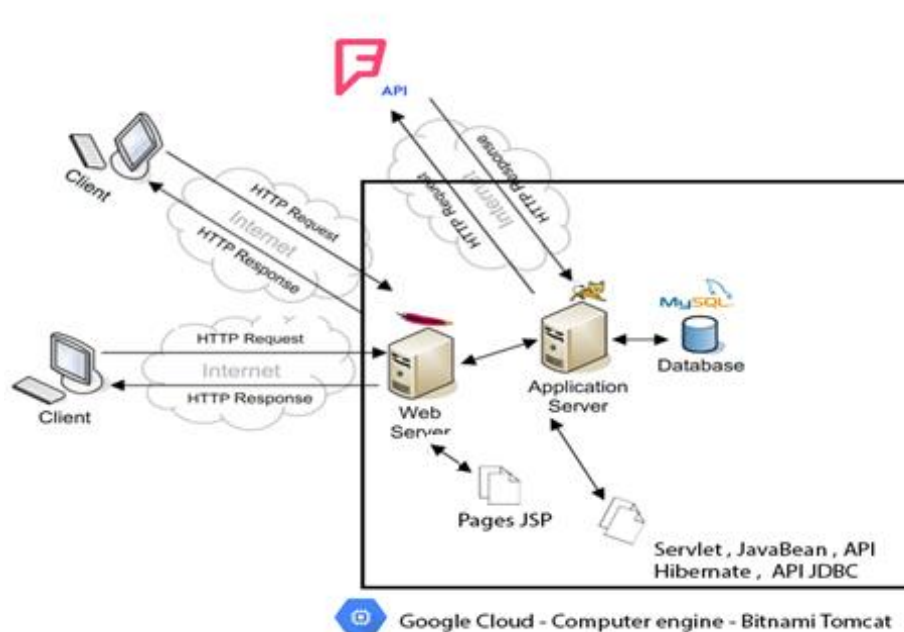


Figure 7: Architecture Matérielle

2. Architecture Logicielle de l'application

Pour la réalisation de notre application web, nous avons adopté une architecture logicielle **N-tiers** avec les principaux modules suivants :

La couche **Interface Utilisateur** qui est l'interface (graphique souvent) qui permet à l'utilisateur de piloter l'application et d'en recevoir des informations.

La couche **Métier** implémente le traitement logique de l'application et la synchronisation entre les différents couches et APIs externes. C'est généralement la couche la plus stable de l'architecture.

Dans notre code correspond au contrôleur « Package Contrôleur » qui est composé de l'ensemble des servlet et de la partie Business qui contient l'ensemble front (Liaison faibles entre les couches) des fonctionnalités qui interagissent avec la couche DAO. « Package Business »

La couche **DAO**² s'occupe de l'accès et la manipulation des données via des méthodes de traitements, le plus souvent des données persistantes au sein d'un SGBD.

Dans notre code correspond aux Packages « Consumer et Repository »

Les couches **Métier** et **DAO** sont normalement utilisées via des interfaces Java (Liaison faibles entre les couches). Ainsi la couche métier ne communique avec la couche dao que via ses interfaces. C'est ce qui assure l'indépendance des couches entre-elles : changer l'implémentation de la couche dao n'a aucune incidence sur la couche métier (tant qu'on ne touche pas à la définition de l'interface de la couche dao). Il en est de même entre les couches **Interface Utilisateur et Métier**.

La couche **ORM** : elle fait le mapping entre les tables relationnelles et les Beans en Java en utilisant le framework Hibernate

Base de données : Notre base de données MySQL avec les différents Tables

User (id,mail,dateofbirth,password,secondname,name,pseudo,sexe,image)

Chat (id,source,destination,message,image)

Rendezvous

(id,idLiker,idTarget,idRestaurant,dateTodate,note,updateIsAccepted,IsAccepted,cancelled Date)

La figure ci-dessous illustre l'architecture globale de notre application web

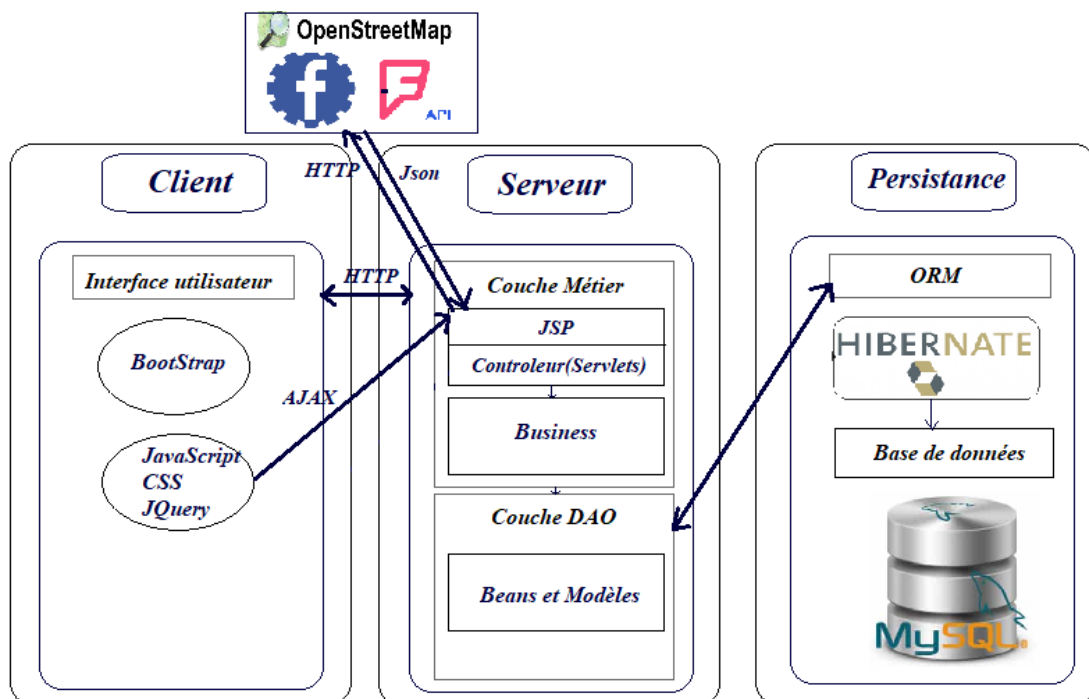


Figure 8: Architecture logicielle et matérielle

² Data Access Object

3. Les APIs et les technologies utilisées

a. Les APIs utilisées

- **L'API Foursquare**

Notre projet se base essentiellement sur l'API REST de « Foursquare». Cette API offre la possibilité d'avoir des informations sur tous les lieux d'une zone géographique donnée. Dans notre cas, nous nous intéressons au domaine de la restauration.

Afin d'avoir les informations sur les restaurants nous utilisons principalement deux API fournissent par Foursquare :

- **Search for Venues** : Cette première API permet d'avoir une réponse JSON contenant principalement le nom, l'adresse et l'identifiant de restaurants qui seront passé par la suite dans l'Api suivante

```
{
  "meta": {
    "response": {
      "venues": [
        {
          "id": "579e332f498e7d497358604c",
          "name": "O'Tacos",
          "location": {
            "address": "5 rue des Innocents",
            "lat": 48.86050541196407,
            "lng": 2.3478661421279465,
            "labeledLatLngs": [
              {
                "label": "display",
                "lat": 48.86050541196407,
                "lng": 2.3478661421279465
              }
            ]
          },
          "postalCode": "75001",
          "cc": "FR",
          "neighborhood": "Les Halles",
          "city": "Paris",
          "state": "Île-de-France",
          "country": "France",
          "formattedAddress": [
            "5 rue des Innocents",
            "75001 Paris",
            "France"
          ]
        }
      ]
    }
  }
}
```

Figure 9: Objet Json retourner de l'API Search for Venues

- **Get Details of a Venue** : Grace aux paramètres passés de l'API « **Search for Venues** ». cette Api retourne un objet JSON complet avec toutes les informations de restaurant tel que l'adresse, Numéro de téléphone, Facebook, Instagram, nombre de j'aime, une description, le site web de restaurant, les horaires de service et une image si ça existe.

Nous avons la possibilité d'exécuter 950/appel par jour pour la première API et 50/appel par jour pour la deuxième.

Sandbox Account

- 950 Regular Calls / Day
- 50 Premium Calls / Day
- 1 Photo per Venue
- 1 Tip per Venue

Looking for more calls or content?
Click below:

Figure 10:SendBox account

- **L'API Openstreetmap**

Nous avons utilisé l'API Openstreetmap afin d'indiquer la localisation géographique des restaurants recherchés. L'API Openstreetmap prend des informations sur la latitude et longitude à l'issu du premier appel de l'API **Search for Venues**, et affiche des cartes avec des marqueurs pour bien cibler l'adresse du lieu.

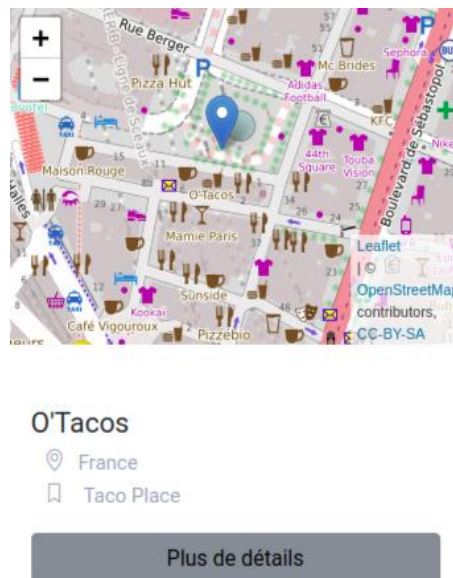


Figure 11:Api Openstreetmap

- **L'API Facebook :**

Notre application fournit la possibilité aux utilisateurs d'éviter de remplir le formulaire d'inscription et se connecter directement via leurs compte Facebook. Notre site récupère les informations suite à un appel GET vers l'Api et les stockent dans notre base de données. Ces informations seront par la suite nécessaires pour afficher les informations d'un profil sur un rendez-vous et dans la partie consacrée à la messagerie instantanée

```

try {
    String acces token= request.getParameter("acces token");
    FacebookClient facebookClient=new DefaultFacebookClient(acces token,Version.VERSION_3_1);
    JsonObject fetchObjectsResults =
        facebookClient.fetchObjects(Arrays.asList("me"),
            JsonObject.class, Parameter.with("fields","name,id,email,picture"));

    JsonObject me = (JsonObject) fetchObjectsResults.get("me");

    String name user= me.get("name").toString().substring(1, me.get("name").toString().length()-1);
    String id user= me.get("id").toString().substring(1, me.get("id").toString().length()-1);
    String email user= me.get("email").toString().substring(1, me.get("email").toString().length()-1);
    JsonObject picture = (JsonObject) me.get("picture");
    JsonObject data = (JsonObject) picture.get("data");
}

```

Figure 12: l'appel à l'API Facebook

b. Le choix des technologies

Coté serveur

- **ORM Hibernate**

Hibernate est une solution open source de type ORM (Object Relational Mapping) qui permet de faciliter le développement de la couche persistance d'une application. Hibernate permet donc de représenter une base de données en objets Java et vice versa.

En effet, nous avons adopté ce framework pour ces différents avantages :

- Facilite la persistance et la recherche de données dans une base de données en réalisant lui-même la création des objets et les traitements de remplissage à travers des méthodes prédéfinies.
- Des bonnes performances, la récupération de données est optimisée, grâce au langage HQL
- La Portabilité du code en cas de changement de la base de données.

```

import ...

@Entity
@Table(name = "users")
public class UtilisateurBean {
    @Id
    @Column(name = "id")
    @GeneratedValue(generator = "incrementator")
    @GenericGenerator(name = "incrementator",strategy = "increment")
    private int id;

    @Column(name = "name")
    private String prenom;

    @Column(name = "secondname")
    private String nom;

    @Column(name = "pseudo")
    private String pseudo;

    @Column(name = "password")
    private String motpasse;
}

```

Figure 13: Bean Users

```

EntityManagerFactory entityManagerfactory = Persistence.createEntityManagerFactory( persistenc

EntityManager entityManager = entityManagerfactory.createEntityManager();

private String USER_DETAILS="Select u from UtilisateurBean u where u.adressemail =:newmail"
private String USER_REGISTRED="Select u from UtilisateurBean u where u.adressemail =:newmai
private String USER_DETAILS_BY_ID="Select u from UtilisateurBean u where u.id =: id";
private String USER_UPDATE="update UtilisateurBean u set u.nom =: newnom ,u.pseudo=: newpse
private String USER_PASS="update UtilisateurBean u set u.motpasse =: password where u.adres
@Override
public void persistuser(UtilisateurBean user) {
    entityManager.getTransaction().begin();
    entityManager.persist(user);
    entityManager.getTransaction().commit();
}

@Override
public void deleteUser(UtilisateurBean user) {

    entityManager.getTransaction().begin();
    entityManager.remove(user);
    entityManager.getTransaction().commit();
}

```

Figure 14 : Manipulation de Bean Users

- **Base de données MySQL**

Nous avons opté à l'utilisation de base de données *MySQL*. Un système de gestion de bases de données relationnelles qui se colle parfaitement avec le framework Hibernate pour différentes raisons :

- Solution très courante en hébergement public
- Très bonne intégration dans l'environnement Apache/PHP
- Facilité de déploiement et de prise en main.
- Connaissances solides et maîtrise des bases de données SQL
- Open Source

- **Junit**

Un framework open source pour le développement et l'exécution de tests unitaires automatisables. Le principal intérêt est de s'assurer que le code répond toujours aux besoins même après d'éventuelles modifications. Plus généralement, ce type de tests est appelé tests unitaires de non-régression


```

class UsersImplementationTest {
    private IUserTransformation iTransformation = new TransformationUser();
    private IUsers iUsers = new UsersImplementation();

    @Test
    public void TestGetUser()
    {

        User user = iTransformation.fromUserBeanToUser(iUsers.getUserDetails( mail: "theyri.dari
        System.out.println(user.getNom());
    }
}

```

Figure 15:Code de test : JUnit

Coté client

Pour la mise en forme notre application, nous nous sommes servis principalement des technologies web standard à savoir CSS3, JQuery, JavaScript et framework Bootstrap.

• AJAX

Notre application exécute plusieurs appels AJAX en utilisant le Framework JavaScript/ JQuery. Nous pouvons prendre en exemple la page dédiée à la messagerie instantanée. Au chargement de la page, une requête http est envoyée au serveur afin d'avoir la liste des amis et les messages dans la boîte des messages. En cliquant sur un ami, la discussion se charge suite à autre appel AJAX qui permet d'invoquer les contrôleurs afin de récupérer la discussion.

```

1 var test = false;
2 var compteurId= 0;
3 $(document).ready(function(){
4     $("#messageText").prop('disabled', true);
5     $("#send").prop('disabled',true);
6     $.ajax({
7         url : 'Chatchargement',
8         type : 'GET',
9         dataType : 'json',
10        success : function(data){
11            $("#websocket").append("<script>websocket = new WebSocket(\"ws://"+window.location.host+"/chatroomServerEndpoint\");</script>");
12            websocketonmessage1();
13            $.each(data.Chat, function(i, obj) {
14                $.each(data.Ami, function(j, obj2){
15                    if(obj.source==obj2.adressemail){
16                        $(".inbox chat").append("<div class='chat list' id='chat"+obj.source+'><div class='chat people'><div class='chat img'> <img id='imglien'+i+'\" s
17                    }
18                });
19                compteurId++;
20            });
21            $.each(data.Ami, function(i, obj1) {
22                $(".list-group").append("<li class='list-group-item'><button type='button' class='btn btn-light \" id='ami\" value='\""+obj1.adressemail+'\"><img id='
23            });
24        },
25        error : function(resultat, statut, erreur){
26        },
27        complete : function(resultat, statut){
28        }
29    });
30    });
31    });
32    });
33    });
34    });
35    });
36    });
37    });
38    });

```

Figure 16:Code AJAX

• Template Bootstrap

Notre site web Eatotogether utilise le framework Bootstrap qui permet d'avoir un contenu responsif. Ainsi notre Template est basé sur ce framework. Il présente principalement 3 parties :

- Header qui est défini dans une page JSP « header.jsp »
- La corp de la page
- Footer qui est défini dans une page JSP « Footer.jsp »

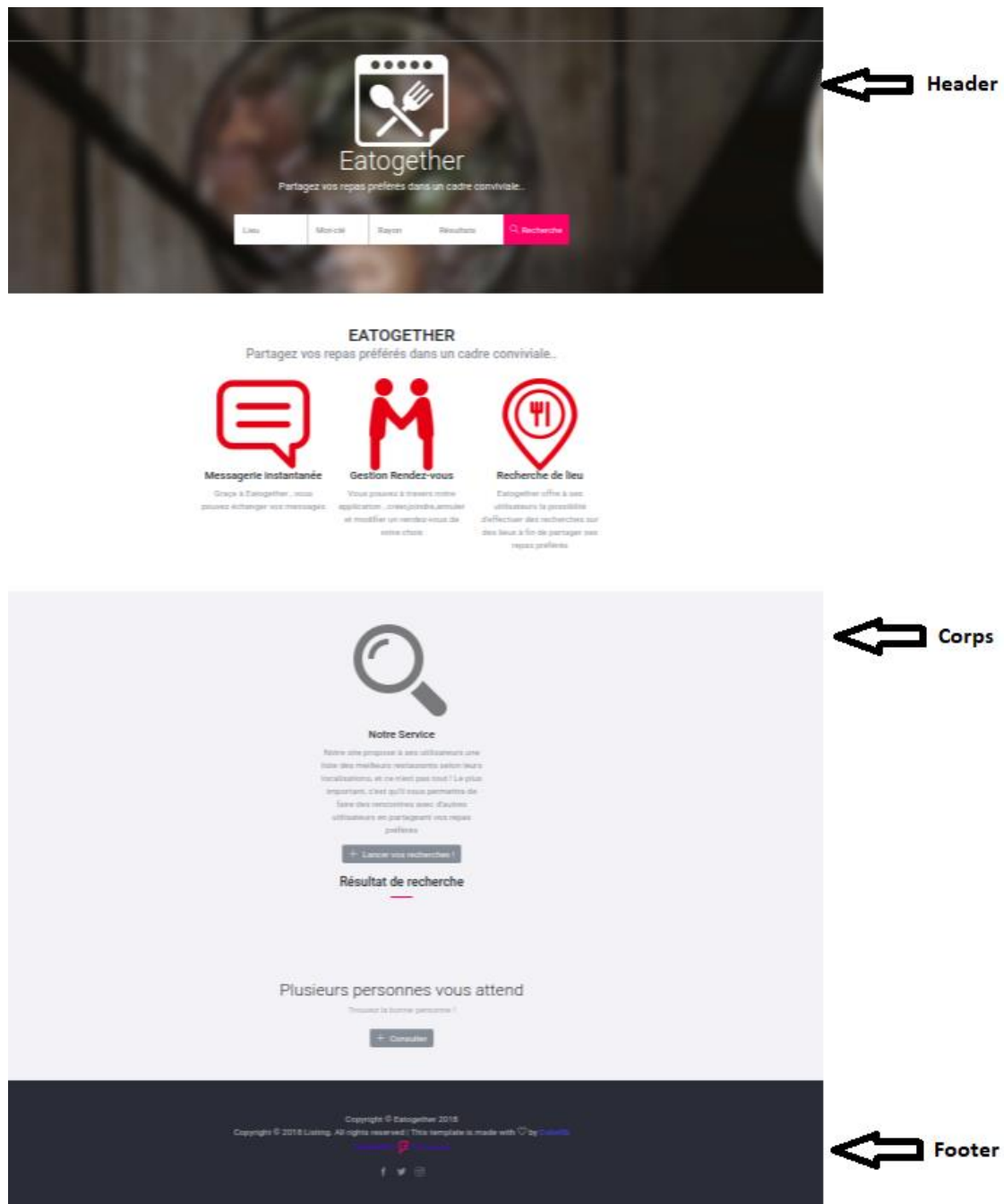


Figure 17: Template Bootstrap

Web Socket

Notre application implémente un serveur de socket sous java. À l'ouverture de la page de messagerie instantanée, Un socket est créé avec la session de l'utilisateur. La session contient deux attributs principaux source et destination afin de permettre au serveur de socket d'effectuer le bon acheminement des messages. Au début, l'utilisateur a une session avec la source qui correspond à son adresse email et une destination aléatoire pour le but de recevoir les messages en provenance des autres utilisateurs dans le système. En choisissant un utilisateur ou bien le message correspond à un utilisateur, l'attribut destination prend la valeur de l'email de l'utilisateur destinataire. À l'issue de cette étape, notre utilisateur peut communiquer à travers les sockets.

```

23 @ServerEndpoint(value="/chatroomServerEndpoint", configurator=ChatroomServerConfigurator.class)
24 public class ChatroomServerEndpoint {
25     static Set<Session> chatroomUsers = Collections.synchronizedSet(new HashSet<Session>());
26
27
28     @OnOpen
29     public void handleOpen(EndpointConfig endpointConfig, Session userSession) {
30         userSession.getUserProperties().put("login", endpointConfig.getUserProperties().get("login"));
31         userSession.getUserProperties().put("destination", endpointConfig.getUserProperties().get("destination"));
32         boolean exist = false;
33         String src = userSession.getUserProperties().get("login").toString();
34         String dest = userSession.getUserProperties().get("destination").toString();
35         for (Session peer : chatroomUsers) {
36             if(peer.getUserProperties().get("login").toString().equals(src) && peer.getUserProperties().get("destination").toString().equals(dest)){
37                 exist = true;
38             }
39         }
40         if(exist==false) {
41             chatroomUsers.add(userSession);
42         }
43     }
44
45     @OnMessage
46     public void handleMessage(String message, Session userSession) throws IOException {
47         String username = (String)userSession.getUserProperties().get("login");
48         if(username==null) {
49             chatroomUsers.stream().forEach(x -> {
50                 try {x.getBasicRemote().sendText(buildJsonData(username,message));}
51                 catch (Exception e) {
52                     e.printStackTrace();
53                 }
54             });
55             userSession.getUserProperties().put("login", message);
56             userSession.getBasicRemote().sendText(buildJsonData("System","you are connected as "+message));
57         } else {
58             boolean exist = false;
59             boolean exist1 = false;
60             for (Session peer : chatroomUsers) {
61                 if(userSession.getUserProperties().get("destination").toString().equals(peer.getUserProperties().get("login").toString())){
62                     if(peer.getUserProperties().get("destination").toString().equals(username)) {
63                         exist1 = true;
64                     }
65                     exist = true;
66                     peer.getBasicRemote().sendText(buildJsonData(username,message));
67                 }
68             }
69             if(exist==false) {
70                 Chat chat = new Chat();
71                 exist1=true;
72                 try {
73                     if(chat.numberOfmsg(username, userSession.getUserProperties().get("destination").toString())<=3) {
74                         chat.insertDiscussion(username, userSession.getUserProperties().get("destination").toString(), message);
75                     }
76                 } catch (Exception e) {
77                     e.printStackTrace();
78                 }
79             }
80         }
81     }
82 }

```

Figure 18: Code de web Socket

4. Hébergement de l'application

- **Google Computer engine - Bitnami Tomcat**

Nous avons choisi de créer une machine virtuelle Bitnami Tomcat sur Google cloud, qui offre un serveur d'application Tomcat préconfigurée afin d'héberger notre site. Nous avons configuré le serveur tomcat pour qu'il supporte le protocole "ws" qui permet de créer un canal de communication en se basant sur TCP. Notre serveur supporte le protocole HTTPS, grâce à un certificat SSL.

The screenshot displays the Google Cloud Platform console for a Bitnami Tomcat instance. The 'Application Info' section shows the instance name 'yassineb3@bitnami-tomcat-dm-cecf', the image 'Tomcat 8.5.35-0', and the OS 'Debian GNU/Linux 9'. The 'Server Info' section shows the IP address '130.211.57.141', the machine type 'G1-SMALL', the disk 'MAGNETIC DISK', and the region 'EUROPE-WEST1-B'. The 'LAUNCH SSH CONSOLE' button is highlighted. Below the console, the terminal output shows the Bitnami Tomcat 8.5.35-0 welcome message and documentation links.

Figure 19: Google Computer engine - Bitnami Tomcat

IV. Points forts et points faibles de notre application

Comme pour chaque application, notre site web présente des points forts et quelques points faibles

1. Points forts :

- Un nom de domaine simple
- Notre serveur possède un certificat SSL ce qui permet d'avoir une connexion sécurisée grâce au protocole HTTPS. Ce certificat nous a permis d'avoir l'approbation dans notre application par Facebook afin d'utiliser son API.
- Notre application offre aux utilisateurs de choisir le mode d'authentification.
- L'API de Foursquare offre une base de données énorme de lieux.
- L'API Openstreetmap est un API « Opensource » qui permet de bien localiser les lieux.
- Eatotogether offre à ses utilisateurs sa propre messagerie grâce aux « Websocket » afin d'échanger des discussions instantanées.

2. Points faibles :

- La limite de nombre d'appel de L'API Foursquare présente un point faible dans l'application. Après 50 appels par jour, Notre application ne peut plus afficher correctement **les détails** d'un lieu.
- L'API Foursquare offre une base de données riche, le nombre énorme de lieu peut diminuer la probabilité d'avoir deux utilisateurs qui choisissent le même lieu.

V. Extensions et difficultés rencontrées

1. Les extensions possibles

- L'utilisation d'une API qui permet de faire la localisation géographique de l'utilisateur sera pertinente.
- La localisation géographique permettra de se limiter sur un rayon indiqué et avoir que les rendez-vous sur une zone donnée.
- Eatotogether peut bénéficier d'autres API de Foursquare afin d'avoir plus d'informations tels que : les commentaires des utilisateurs, les statistiques et plus d'images le lieu.
- L'utilisation d'un API de météo peut aider les utilisateurs à choisir la date d'un rendez-vous.
- La recherche par catégorie permettra aux utilisateurs de découvrir les différentes catégories sans avoir une idée sur les mots clés.

Plusieurs autres extensions et amélioration reste envisageables.

2. Difficultés rencontrées

• Web socket

Au début, nous avons choisi d'héberger notre site sur la plateforme Heroku. Cependant, nous nous sommes rendu compte que le serveur Apache Tomcat utilisé par la plateforme ne supporte pas les « web socket ». L'importance de la fonctionnalité de messagerie dans notre application nous a poussé à tester plusieurs autres solutions comme AWS et Google cloud.

Afin de pallier ce problème, nous avons créé une machine virtuelle Bitnami Tomcat sur Google cloud. Ainsi plusieurs configurations sur le serveur Tomcat nous ont mené à avoir un module de chat fonctionnelle.

- **Problème avec Facebook**

Nous avons rencontré des difficultés pour approuver notre application par Facebook et bénéficier des informations fournies par son API. Dans sa nouvelle version, Facebook exige une application sécurisé accessible à travers le protocole (HTTPS). Afin de résoudre ce problème, nous avons été amenés à installer et configurer un certificat SSL dans notre serveur Tomcat.

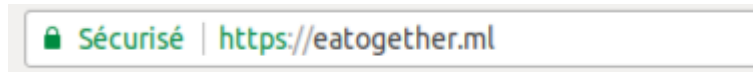


Figure 20 : Protocole Https

Conclusion

Ce projet a été une bonne expérience pour pratiquer et se familiariser avec les technologies «web». Tout au long de l'élaboration du projet, nous avons rencontré plusieurs difficultés aussi bien sur le niveau conceptuel que sur le niveau de la réalisation. Mais, nous avons réussi à surpasser les difficultés afin de présenter une application fonctionnelle et qui respecte les exigences. Plusieurs améliorations dans notre application restent envisageables afin d'avoir un projet complet. Certes, Cette expérience bénéfique pour nous été un milieu d'autoévaluation et d'apprentissage des techniques de développement web.