mory

Projet module 133

Date de création : 30.03.2023  
Version 1 du 05.05.2023

Nicolas Schwander, Xavier Rey

|  |  |  |  |
| --- | --- | --- | --- |
| S:\EMF\CPROF\Identite_EMF\2015-Identites\Logos des sections COULEURS\Logo_EMF-Informatique_FR_RVB_25.jpg |  |  | Module du 23.03.2023 au 05.05.2023 |

Table des matières

[1 Introduction 4](#_Toc134370025)

[2 Analyse 4](#_Toc134370026)

[2.1 Architecture globale 4](#_Toc134370027)

[2.2 Use case 5](#_Toc134370028)

[2.2.1 Use case client Memory 5](#_Toc134370029)

[2.2.2 Use case client User 6](#_Toc134370030)

[2.2.3 Use case ApiGateway 7](#_Toc134370031)

[2.2.4 Use case Rest Classement 8](#_Toc134370032)

[2.2.5 Use case Rest User 9](#_Toc134370033)

[2.3 Séquence system 10](#_Toc134370034)

[2.3.1 Séquence system AjoutScore 10](#_Toc134370035)

[2.3.2 Séquence system AjoutUtilisateur 11](#_Toc134370036)

[3 Conception à faire complétement avec EA 12](#_Toc134370037)

[3.1 Class Diagrammes 12](#_Toc134370038)

[3.1.1 Architecture ApiGateway 12](#_Toc134370039)

[3.1.2 Architecture RestClassement 12](#_Toc134370040)

[3.1.3 Architecture RestUser 13](#_Toc134370041)

[4 Bases de données 13](#_Toc134370042)

[4.1 Modèles PhpMyAdmin 13](#_Toc134370043)

[4.1.1 BD classement 13](#_Toc134370044)

[4.1.2 BD Utilisateur 13](#_Toc134370045)

[4.2 Requêtes SQL utiles 14](#_Toc134370046)

[4.2.1 BD classement 14](#_Toc134370047)

[4.2.2 BD Utilisateur 14](#_Toc134370048)

[5 Implémentation des applications client Ap1 et client Ap2 15](#_Toc134370049)

[5.1 Application Client 1 15](#_Toc134370050)

[5.1.1 Jeu 15](#_Toc134370051)

[5.1.2 Requêtes 15](#_Toc134370052)

[5.2 Application client 2 17](#_Toc134370053)

[6 Implémentation de l'aplication API Gateway 19](#_Toc134370054)

[6.1 Partie Servlet sur serveur Tomcat commun 19](#_Toc134370055)

[6.2 Gestion session 19](#_Toc134370056)

[7 Implémentation des applications Service Rest1 et Service Rest2 20](#_Toc134370057)

[7.1 Application Service Rest1 20](#_Toc134370058)

[7.2 Application Service Rest2 22](#_Toc134370059)

[8 Hébergement 26](#_Toc134370060)

[9 Installation du projet complet avec les 5 applications 27](#_Toc134370061)

[10 Outils, langages (versions, définitions, installations) 28](#_Toc134370062)

[10.1 Client memory 28](#_Toc134370063)

[10.2 Java 28](#_Toc134370064)

[10.3 IDE Clients 28](#_Toc134370065)

[11 Tests de fonctionnement du projet 28](#_Toc134370066)

[12 Auto-évaluations et conclusions 29](#_Toc134370067)

[12.1 Nicolas 29](#_Toc134370068)

[12.2 Xavier 29](#_Toc134370069)

# Introduction

Le but de notre projet est de faire un jeu memory. Nous avons deux clients, le premier client inclut le jeu et le classement des scores des joueurs. Le deuxième client permet de créer des user ainsi que de rechercher un user. Pour participer au classement le joueur devra être logé.

# Analyse

## Architecture globale

Voici l’architecture globale du projet :

Une image contenant texte, capture d’écran, diagramme, conception

Description générée automatiquement

## Use case

### Use case client Memory

Ci-dessous, le use case du client Memory :

Une image contenant diagramme, texte, capture d’écran, ligne

Description générée automatiquement

Les actions que le client peut effectuer envers l’apiGateway sont celles pour :

* + Voir le classement, même les visiteurs sans comptes doivent pouvoir voir le classement des scores mondiaux.
  + Se connecter, cette action permet simplement d’ouvrir une session sur l’apiGateway pour pouvoir ensuite ajouter son score
  + Envoyer le score obtenu, une fois connecté, l’orsque l’on fini une partie de memory le score pourra être ajouté au classement.

### Use case client User

Une image contenant diagramme, capture d’écran, texte, ligne

Description générée automatiquement

Voici les interactions que fait l’application client user avec l’API Gateway.

* Le visiteur peut rechercher un user
* Le visiteur peut créer un user.

### Use case ApiGateway

Ci-dessous l’use case de l’api Gateway.

Une image contenant texte, capture d’écran, diagramme, cercle

Description générée automatiquement

Le use case permet aux applications d’effectuer des opérations sur les micro-services restClassement et restUser à un seul endroit, tout en gérant la session.

Operations possibles :

* + SendScore, permet d’envoyer un score au service RestClassement pour qu’il soit ajouté dans la base de données.
  + GetClassement, permet de récupérer tout le classement.
  + Check login, Permet de vérifier si un utilisateur et son mot de passe correspondant existent bien dans la base de données derrière le service RestUser
  + AddUser, Permet d’envoyer un utilisateur et son mot de passe au service Rest User pour qu’il soit ajouté à la base de données
  + GetUser, permet de récupérer un utilisateur spécifique.

### Use case Rest Classement

Ci-dessous le use case du service RestClassement.

Une image contenant texte, diagramme, capture d’écran, cercle

Description générée automatiquement

Le service Rest classement permet d’effectuer des actions sur la base de données.

* + GetClassement, sert à récupérer le classement en entier, retour en format json.
  + SetScore, sert à ajouter un score a la base de données

### Use case Rest User

Une image contenant diagramme, capture d’écran, texte, cercle

Description générée automatiquement

Voici les interaction entre du serveur Rest\_User

* Get User permet de rechercher un user dans la base de données.
* Le Login permet de se loger.
* AddUser permet de crée un nouveau user.

## Séquence system

### Séquence system AjoutScore

Ci-dessous le diagramme de séquence système pour l’ajout d’un score dans le classement

Une image contenant texte, diagramme, ligne, Parallèle

Description générée automatiquement

Un utilisateur doit d’abord terminer une partie pour déclencher l’ajout d’un score au classement, ensuite l’apiGateway reçoit le score et vérifie si la session contient un nom d’utilisateur logué. Si oui le résultat sera envoyé au service rest classement avec le nom d’utilisateur logué et ceci sera ajouté à la base de données.

### Séquence system AjoutUtilisateur

Une image contenant texte, diagramme, Parallèle, ligne

Description générée automatiquement

Dans l’application user, on peut créer un user. L’application va check les donnée puis va transmettre à L’APIGateway. Celui-ci va transmettre les données au bon rest, dans notre cas au Service Rest\_User. Lr service Rest\_User va transmettre les données à la db\_User. Si le nom du user est déjà utiliser le cela va donner redonnée un message d’erreur.

# Conception à faire complétement avec EA

## Class Diagrammes

### Architecture ApiGateway

Une image contenant texte, capture d’écran, Police, carte de visite

Description générée automatiquement

L’architecture de l’api Gateway est plutôt simple, il y a un servlet pour traiter les requêtes entrantes et un Workeur qui contient les méthodes pour faire des requêtes http vers les servers rest spécifiques.

Au niveau du servlet API Gateway c’est dans la méthode processRequest que seront traitées les différentes requêtes.

### Architecture RestClassement

Une image contenant texte, capture d’écran, Police, Rectangle

Description générée automatiquement

Le rest Classement est simple dans son fonctionnement, c’est juste un petit server rest qui appelle les méthodes d’ajout a la base de donnes qui se trouvent dans le worker pour chaque opération.

### Architecture RestUser

Une image contenant texte, capture d’écran, Police, Rectangle

Description générée automatiquement

XAVIER

Voici l’architecture du RestUser. On retrouve les éléments principale, openConnexion, closeConnexion. Et les éléments nécessaires à l’application, addUser, getUser, checkUser.

# Bases de données

## Modèles PhpMyAdmin

### BD classement

Une image contenant texte, capture d’écran, Police, nombre

Description générée automatiquement

La base de donnes classement est très simple, elle contient une seule table classement avec les infos pour chaque entrée. Une PK, le nom de l’utilisateur, et son score, ceci est les données que l’on utilise.

Nous avions initialement pensé ajouter le temps (durée de la partie de memory) mais j’ai finalement décidé de simplement prendre en compte le temps dans le calcul du score initial (sur le client).

La FK user n’est pas non plus utilisé, il aurait servi de lien entre les deux bases de données mais il est inutile car le nom est aussi unique et peut plus facilement être utilisé a sa place.

### BD Utilisateur

Voici la base de donnes user. La base de données contient une PK, un string user, un autre string password. Initialement nous avions prévu d’utiliser un admin. Mais finalement nous ne l’avons pas utilisé c’est pour ca qu’il y a encore admin.

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

## Requêtes SQL utiles

### BD classement

La requête que nous utilisons le plus est celle trouvées dans le worker de mon restClassement pour l’ajout d’une entrée.

ps = dbConnexion.prepareStatement("INSERT INTO t\_classement (score, nom, fk\_user) VALUES (?, ?, ?)");

ps.setInt(1, score);

ps.setString(2, nom);

ps.setInt(3, fk\_user);

### BD Utilisateur

Voici une requête que nous utilisons pour la base de données user. Elle permet d’ajouter un nouveau user dans celle-ci.

ps = dbConnexion.prepareStatement("INSERT INTO t\_user (user, password) VALUES (?, ?)");

# Implémentation des applications client Ap1 et client Ap2

## Application Client 1

### Jeu

Cette application est un jeu memory sur une base de 4 cartes sur 4.

Aspect visuel :

Nous avons utilisé des emoji pour les faces des cartes pour ne pas devoir utiliser des images :

const generateGame = () => {  
 const dimensions = ***selectors***.board.getAttribute('data-dimension')  
  
 if (dimensions % 2 !== 0) {  
 throw new Error("The dimension of the board must be an even number.")  
 }  
  
 const emojis = ['🥔', '🍒', '🥑', '🌽', '🥕', '🍇', '🍉', '🍌', '🥭', '🍍']  
 const picks = pickRandom(emojis, (dimensions \* dimensions) / 2)   
 const items = shuffle([...picks, ...picks])  
 const cards = `  
 <div class="board" style="grid-template-columns: repeat(${dimensions}, auto)">  
 ${items.map(item => `  
 <div class="card">  
 <div class="card-front"></div>  
 <div class="card-back">${item}</div>  
 </div>  
 `).join('')}  
 </div>  
 `  
   
 const parser = new DOMParser().parseFromString(cards, 'text/html')  
  
 ***selectors***.board.replaceWith(parser.querySelector('.board'))  
}

### Requêtes

L’ajout d’un score dans la base de données se fait lorsque’ une partie est terminée, avec le score calculé :

// If there are no more cards that we can flip, we won the game  
 if (!***document***.querySelectorAll('.card:not(.flipped)').length) {  
 score = calculateScore(***state***.totalTime,***state***.totalFlips)  
 setTimeout(() => {  
 ***selectors***.boardContainer.classList.add('flipped')  
 ***selectors***.win.innerHTML = `  
 <span class="win-text">  
 You won!<br />  
 with <span class="highlight">${***state***.totalFlips}</span> moves<br />  
 under <span class="highlight">${***state***.totalTime}</span> seconds<br/>  
 Your calculated scor is <span class="highlight">${score}</span>  
 </span>  
 `  
 //score = calculateScore(state.totalTime,state.totalFlips)  
 sendScore("addEntree",score)  
 chargerClassement(chargerClassementSuccess, chargerClassementError);  
  
 clearInterval(***state***.loop)  
 }, 1000)  
 }  
}

Et dans mon serviceshttp.js :

function chargerClassement(successCallback, errorCallback) {  
 $.ajax({  
 type: "GET",  
 dataType: "json",  
 url: "http://localhost:8080/ApiGateway/servletGateway?type=getclassement",  
 success: function(response) {  
 successCallback(response.test);  
 },  
 xhrFields: {  
 withCredentials: true  
 },  
 async: false,  
 crossDomain: true,  
 error: errorCallback  
 });  
}  
  
  
  
function sendScore(type, score, successCallback, errorCallback) {  
 $.ajax({  
 type: "POST",  
 dataType: "json",  
 url: "http://localhost:8080/ApiGateway/servletGateway",  
 data: {  
 type: type,  
 score: score  
 },  
 xhrFields: {  
 withCredentials: true  
 },  
 async: false,  
 crossDomain: true,  
 success: successCallback,  
 error: errorCallback  
  
 });  
  
}

A noter que la partie surlignée en bleu est très importantes et permet la gestion de la session sur l’api Gateway :

* + **xhrFields** permet de définir des propriétés supplémentaires pour l'objet XHR. Dans ce cas, **withCredentials** est défini à **true**, ce qui permet d'envoyer des cookies et des en-têtes d'authentification avec la requête XHR.
  + **async** est une propriété booléenne qui détermine si la requête XHR est asynchrone ou non. Dans ce cas, elle est définie à **false**, ce qui signifie que la requête est synchrone et que le navigateur attendra une réponse avant de continuer à exécuter la ligne de code suivante.
  + **crossDomain** est une propriété booléenne qui indique si la requête XHR est une requête entre domaines différents. Si elle est définie à **true**, cela signifie que la requête est envoyée vers un domaine différent de celui qui a servi la page web.

## Application client 2

Voici le code du userCtrl.js.

const myButton = document.getElementById('adduser');

const myButton2 = document.getElementById('recherche');

myButton.addEventListener('click', function() {

    addUser(document.getElementById("user").value,document.getElementById("password").value, successCallback, errorCallback)

});

myButton2.addEventListener('click', function() {

    chercherUser(document.getElementById("search").value, successCallback2, errorCallback2)

});

function errorCallback2(request, status, error) {

    alert("Erreur lors de la recherche");

}

function successCallback2(data, text, jqXHR) {

    console.log(data)

}

function errorCallback(request, status, error) {

    alert("Erreur lors de l'enregistrement");

}

function successCallback(data, text, jqXHR) {

    if (data.includes("OK"))

    {

        alert("Sign-Up ok");

    }

    else{

        alert("Erreur lors de l'enregistrement");

    }

}

$(document).ready(function () {

    $.getScript("./services/servicesHttp.js", function () {

        console.log("servicesHttp.js chargé !");

    });

});

Voici le code du service.js

function addUser(user, mdp, successCallback, errorCallback) {

    $.ajax({

        type: "POST",

        dataType: "text",

        url: "https://schwandern.emf-informatique.ch/javaApiGateway/servletGateway",

        data: {

            type: "Adduser",

            user: user,

            password: mdp

        },

        xhrFields: {

            withCredentials: true

        },

        async: false,

        crossDomain: true,

        success: successCallback,

        error: errorCallback

    });

}

function chercherUser(username ,successCallback, errorCallback) {

    $.ajax({

        type: "GET",

        dataType: "json",

        url: "https://schwandern.emf-informatique.ch/javaApiGateway/servletGateway?type=getUser&user="+username,

        success: function(response) {

            successCallback(response.test);

        },

        xhrFields: {

            withCredentials: true

        },

        async: false,

        crossDomain: true,

        error: errorCallback

    });

}

Pour communiquer les données à l’API Gateway. On utilise un post et en get. Le post est utiliser pour la ajouter un user et le get pour rechercher un user.

# Implémentation de l'aplication API Gateway

## Partie Servlet sur serveur Tomcat commun

Dans mon servlet la gestion de toutes les requêtes se passe dans la methode process Request, chaque type de requête est traité par l’attribut obligatoire de type :

protected void processRequest(HttpServletRequest request, HttpServletResponse response)

throws ServletException, IOException {

response.setContentType("text/json;charset=UTF-8");

response.addHeader("Access-Control-Allow-Origin", request.getHeader("Origin"));

response.addHeader("Access-Control-Allow-Credentials", "true");

String requestType = new String(request.getParameter("type"));

if (requestType.equals("getclassement")) {

try ( PrintWriter out = response.getWriter()) {

out.print(wrk.getclassement());

out.flush();

}

}

if (requestType.equals( … ETCETERA

## Gestion session

Pour gérer la session tout se passe lors du login d’un client :

if (requestType.equals("checkLogin")) {

String nom = new String(request.getParameter("user"));

String password = new String(request.getParameter("password"));

HttpSession session = request.getSession();

try ( PrintWriter out = response.getWriter()) {

String s = wrk.checkLogin(nom, password);

if (s.equals("OK")) {

request.getSession().setAttribute("user", nom);

}

out.print(s);

out.flush();

}

}

Ainsi on peut savoir si un client est logué ou non en vérifiant si la session contient une variable avec le nom de l’utilisateur connecté :

if (requestType.equals("addEntree") && (request.getSession().getAttribute("user")!=null)) {

String score = new String(request.getParameter("score"));

String s = (String) request.getSession().getAttribute("user");

try ( PrintWriter out = response.getWriter()) {

out.print(wrk.addEntree(score, s));

out.flush();

}

}

# Implémentation des applications Service Rest1 et Service Rest2

## Application Service Rest1

La partie qui reçoit les requêtes et les traite :

@Path("db")

public class DB {

@Context

private UriInfo context;

public DB() {

}

Wrk wrkDB = new Wrk("3306", "schwandern\_133\_memory");

@GET

@Path("getClassement")

@Produces(MediaType.APPLICATION\_JSON)

public String getClassement() {

Gson builder = new Gson();

String toJson = builder.toJson(wrkDB.getClassement());

return "{\"test\":" + toJson + "}";

}

@POST

@Path("addEntree")

@Produces(MediaType.TEXT\_PLAIN)

@Consumes(MediaType.APPLICATION\_FORM\_URLENCODED)

public String addEntree(@FormParam("name") String name, @FormParam("score") int score, @FormParam("fk\_user") int fkUser) {

String result;

if (wrkDB.insertClassementEntry(score, name, fkUser)) {

result = "OK";

} else {

result = "NOK";

}

return result;

}

}

Et la partie dans la partie worker il y a toutes les méthodes nécessaires pour effectuer des requêtes sur la BD (connexion, déconnexion, et les requêtes spécifiques). Exemple :

public ArrayList<String> getClassement() {

ArrayList<String> lstClassement = null;

boolean result = openConnexion();

if (result) {

System.out.println("Connection successful.");

PreparedStatement ps = null;

String entree = "";

String score = "";

String nom = "";

String fk\_user = "";

lstClassement = new ArrayList<String>();

try {

ps = dbConnexion.prepareStatement("SELECT \* FROM t\_classement");

ResultSet rs = ps.executeQuery();

while (rs.next()) {

entree = Integer.toString(rs.getInt("PK\_entree"));

score = rs.getString("score");

nom = rs.getString("nom");

fk\_user = rs.getString("fk\_user");

lstClassement.add(entree + ", " + score + ", " + nom + ", " + fk\_user);

}

rs.close();

result = true;

System.out.println("Query executed successfully.");

} catch (Exception ex) {

System.out.println("Error executing query: " + ex.getMessage());

}

if (result) {

result = closeConnexion();

}

}

return lstClassement;

}

## Application Service Rest2

Voici la partie db du Rest\_User. On a les différentes requetés à traité.

@Path("db")

public class DB {

@Context

private UriInfo context;

Wrk wrkDB = new Wrk("3306", "reyx\_133\_G4\_Admin");

/\*\*

\* Creates a new instance of DB

\*/

public DB() {

}

@GET

@Path("Getusers")

@Produces(MediaType.APPLICATION\_JSON)

public String Getusers(@QueryParam("user") String user) {

Gson builder = new Gson();

String toJson = builder.toJson(wrkDB.getUsers(user));

return "{\"user\":" + toJson + "}";

}

@POST

@Path("Addusers")

@Produces(MediaType.TEXT\_PLAIN)

@Consumes(MediaType.APPLICATION\_FORM\_URLENCODED)

public String addUsers(@FormParam("user") String user, @FormParam("password") String password) {

String s;

if (wrkDB.addUsers(user, BCrypt.hashpw(password, BCrypt.gensalt()))) {

s = "OK";

} else {

s = "KO";

}

return s;

}

@POST

@Path("checkLogin")

@Produces(MediaType.TEXT\_PLAIN)

@Consumes(MediaType.APPLICATION\_FORM\_URLENCODED)

public String checkLogin(@FormParam("user") String user, @FormParam("password") String password) {

String s;

if (BCrypt.checkpw(password, wrkDB.GetMdp(user))) {

s = "OK";

} else {

s = "KO";

}

return s;

}

}

Voici la partie wrk du Rest\_User. Il y les requête adduser, getUser et checkLogin.

public String GetMdp(String user) {

boolean resultLogin = false;

ArrayList<String> lstUsers = null;

boolean result = openConnexion();

if (result) {

System.out.println("connection ok");

PreparedStatement ps = null;

String passwordDB = "";

try {

ps = dbConnexion.prepareStatement("SELECT password FROM t\_user WHERE user like (?) ");

ps.setString(1, user);

ResultSet rs = ps.executeQuery();

while (rs.next()) {

passwordDB = (String) rs.getString("password");

}

rs.close();

} catch (Exception ex) {

return "error";

}

if (result) {

result = closeConnexion();

}

return passwordDB;

}

return "error";

}

public ArrayList<String> getUsers(String user) {

ArrayList<String> lstUsers = null;

boolean result = openConnexion();

if (result) {

System.out.println("connection ok");

PreparedStatement ps = null;

String pk\_user = "";

String password = "";

lstUsers = new ArrayList<String>();

try {

ps = dbConnexion.prepareStatement("SELECT \* FROM t\_user WHERE user like (?) ");

ps.setString(1, user);

ResultSet rs = ps.executeQuery();

while (rs.next()) {

pk\_user = (String) rs.getString("PK\_User");

user = (String) rs.getString("user");

password = (String) rs.getString("password");

lstUsers.add(pk\_user + ", " + user + ", " + password);

}

rs.close();

result = true;

} catch (Exception ex) {

System.out.println(ex.getMessage());

}

if (result) {

result = closeConnexion();

}

}

return lstUsers;

}

public boolean addUsers(String user, String password) {

boolean success = false;

boolean result = openConnexion();

if (result) {

PreparedStatement ps = null;

try {

ps = dbConnexion.prepareStatement("INSERT INTO t\_user (user, password) VALUES (?, ?)");

ps.setString(1, user);

ps.setString(2, password);

int rowsAffected = ps.executeUpdate();

if (rowsAffected > 0) {

success = true;

System.out.println("New entry added to t\_user table.");

} else {

System.out.println("No rows affected when adding new entry to t\_user table.");

}

} catch (Exception ex) {

System.out.println("Error executing insert statement: " + ex.getMessage());

}

if (success) {

result = closeConnexion();

}

}

return success;

}

# Hébergement

Pour notre hébergement, nous avons dû configurer notre compte Tizoo. Pour cela, nous avons modifié le fichier XML nommé "server.xml" dans les outils ea-tomcat de notre compte. Nous avons configuré la valeur de "unpackWARs" à true pour que le serveur puisse extraire et traiter les fichiers .war que nous luis envoyons.

Une image contenant texte, logiciel, Page web, Site web

Description générée automatiquement

Ensuite il a fallu aller sur la console du cpanel et y mettre cette commande :

export PATH=$(dirname $(readlink/usr/local/cpanel/3rdparty/bin/perl)):$PATH

ubic restart ea-tomcat85

Une image contenant texte, Appareils électroniques, capture d’écran, logiciel

Description générée automatiquement[[1]](#footnote-1)

Ensuite, ul faut simplement se connecter avec FileZilla ou coreFTP et mettre son projet .war pour que le server fasse le reste du travail.

# Installation du projet complet avec les 5 applications

Nous avons mis l’api GateWay sur le mon compte Tizoo (nicolas Schwander) :

<http://schwandern.emf-informatique.ch/javaApiGateway/>

Pour le server Rest du classement il est sur le comte tizoo suivant :

<http://schwandern.emf-informatique.ch/java-Rest_Classement/>

Pour le client memory il est sur le comte tizoo suivant :

<http://schwandern.emf-informatique.ch/client_memory/>

Pour le server Rest User il est sur le compte tizoo de Rey Xavier

<https://reyx.emf-informatique.ch/javaRest_Admin02/>

Pour le client User il est sur le compte tizoo de Rey Xavier

<https://reyx.emf-informatique.ch/133_User_REY/User.html>

# Outils, langages (versions, définitions, installations)

## Client memory

Pour l’implémentation du client memory je me suis basé sur une explication sur le site suivant :

<https://webtips.dev/memory-game-in-javascript>

## Java

Pour la partie java – server Rest/servlet nous sommes resté sur l’IDE NetBeans car c’est celui qu’on utilise le plus souvent en classe et avec lequel nous sommes le plus à l’aise.

## IDE Clients

Personnellement (nicolas schwander) j’ai utilisé l’IDE WebStorm car j’ai une license étudiante d’un an et je suis très à l’aise avec, ayant déjà créé plusieurs sites dynamiques sur cet IDE.  
  
Xavier est resté sur Visual studio code car c’est celui que nous avons utilisé dans d’autres modules et il est simple a utiliser

# Tests de fonctionnement du projet

|  |  |  |
| --- | --- | --- |
| **No test** | **Description** | **Résultats attendus** |
| **1** | Un visiteur peut créer un compte sur le client User | Le visiteur peut créer son compte. |
|  | **Résultats obtenus** | **Conclusion** |
|  | Le visiteur a pu créer un compte | Une image contenant texte, clipart  Description générée automatiquement |

|  |  |  |
| --- | --- | --- |
| **No test** | **Description** | **Résultats attendus** |
| **2** | Un visiteur Peut se loguer avec son compte crée sur Le client User, sur le Client Memory | Le visiteur peut se loguer avec ce même compte. |
|  | **Résultats obtenus** | **Conclusion** |
|  | Le visiteur a pu se connecter à son compte. | Une image contenant texte, clipart  Description générée automatiquement |

|  |  |  |
| --- | --- | --- |
| **No test** | **Description** | **Résultats attendus** |
| **3** | Un utilisateur logué qui a fini sa partie et obtenu un score supérieur au dernier sur le classement voit apparaitre son score | Le score est visible sur le classement |
|  | **Résultats obtenus** | **Conclusion** |
|  | Le score est visible sur le classement | Une image contenant texte, clipart  Description générée automatiquement |

# Auto-évaluations et conclusions

## Nicolas

Dans son ensemble j’ai très bien aimé ce module car le projet choisi m’intéressait et j’avais beaucoup de motivation pour faire quelque chose de bien. Après avoir compris le principe de base du fonctionnement de java web j’ai pris beaucoup de plaisir à coder mon application. Ce module m’a apporté beaucoup de connaissances autant en termes de la méthode de travail que de connaissances spécifiques. J’ai trouvé bien d’aborder une autre manière de faire des serveurs webs car pour l’instant je connaissais uniquement PHP, j’ai aussi beaucoup appris sur la notion de micro-services et la séparation des différentes parties en plusieurs unités indépendantes.

## Xavier

Durant ce module, j’ai appris de nouvelle chose comme par exemple à configurer Tomcat, à faire un API Gateway en java et de les faire communiquer avec un client. J’ai aimé travailler sur ce module, travailler sur le projet ma beaucoup plus. J’ai éprouvé du plaisir à travailler sur ce module. J’ai aussi pu travailler avec GitHub et Postman ce qui m’a permis d’être plus à l’aise avec ces outils. J’ai aussi apprécié de gérer et planifier notre projet.

1. [↑](#footnote-ref-1)