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](#_Toc134355417)

[2 Analyse à faire complètement avec EA 4](#_Toc134355418)

[2.1 Use case 4](#_Toc134355419)

[2.1.1 Use case client Memory 4](#_Toc134355420)

[2.1.2 Use case client User 5](#_Toc134355421)

[2.1.3 Use case ApiGateway 6](#_Toc134355422)

[2.1.4 Use case Rest Classement 7](#_Toc134355423)

[2.1.5 Use case Rest User 8](#_Toc134355424)

[2.2 Séquence system 9](#_Toc134355425)

[2.2.1 Séquence system AjoutScore 9](#_Toc134355426)

[2.2.2 Séquence system AjoutUtilisateur 10](#_Toc134355427)

[3 Conception à faire complétement avec EA 11](#_Toc134355428)

[3.1 Class Diagrammes 11](#_Toc134355429)

[3.1.1 Architecture ApiGateway 11](#_Toc134355430)

[3.1.2 Architecture RestClassement 11](#_Toc134355431)

[3.1.3 Architecture RestUser 12](#_Toc134355432)

[4 Bases de données 12](#_Toc134355433)

[4.1 Modèles PhpMyAdmin 12](#_Toc134355434)

[4.1.1 BD classement 12](#_Toc134355435)

[4.1.2 BD Utilisateur 13](#_Toc134355436)

[4.2 Requêtes SQL utiles 13](#_Toc134355437)

[4.2.1 BD classement 13](#_Toc134355438)

[4.2.2 BD Utilisateur 13](#_Toc134355439)

[5 Implémentation des applications client Ap1 et client Ap2 14](#_Toc134355440)

[5.1 Application Client 1 14](#_Toc134355441)

[5.1.1 Jeu 14](#_Toc134355442)

[5.1.2 Requêtes 14](#_Toc134355443)

[5.2 Application client 2 16](#_Toc134355444)

[6 Implémentation de l'aplication API Gateway 16](#_Toc134355445)

[6.1 Partie Servlet sur serveur Tomcat commun 16](#_Toc134355446)

[6.2 Gestion session 16](#_Toc134355447)

[7 Implémentation des aplications Service Rest1 et Service Rest2 17](#_Toc134355448)

[7.1 Application Service Rest1 17](#_Toc134355449)

[7.2 Application Service Rest2 17](#_Toc134355450)

[8 Hébergement 17](#_Toc134355451)

[9 Installation du projet complet avec les 5 applications 17](#_Toc134355452)

[10 Outils, langages (versions, définitions, installations) 17](#_Toc134355453)

[11 Tests de fonctionnement du projet 17](#_Toc134355454)

[12 Auto-évaluations et conclusions 17](#_Toc134355455)

[12.1 Nicolas 17](#_Toc134355456)

[12.2 Xavier 17](#_Toc134355457)

# Introduction

Le but de notre projet est de faire un 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 doit être logé.

# Analyse à faire complètement avec EA

## 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

XAVIER REMPLIS

### Use case ApiGateway

Ci-dessous le 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

Xavier Remplis

## 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

XAVIER REMPLIS

# 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 !!

# 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 entree. Une PK, le nom de l’utilisateur, et son score, ceci sont 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 calcule du score initial (sur le client).

La FK user n’est pas non plus utilisé, il aurait servit 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

XAVIER pour mettre le tien va dans php my admin, clique sur la bonne BD et clique sure designer :  
Une image contenant capture d’écran, texte, logiciel, Icône d’ordinateur

Description générée automatiquement



## Requêtes SQL utiles

### BD classement

La requête que j’utilise 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

Xavier stp

# 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 :

J’ai 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

Xavier fait qqchose stp

# 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 gerer 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 aplications Service Rest1 et Service Rest2

## Application Service Rest1

## Application Service Rest2

# Hébergement

# Installation du projet complet avec les 5 applications

# Outils, langages (versions, définitions, installations)

# Tests de fonctionnement du projet

# Auto-évaluations et conclusions

## Nicolas

## Xavier