133 – Réaliser des applications Web en Session-Handling

Documentation de projet

Version 1 du avril 2025

Crée le 24.03.25

|  |  |
| --- | --- |
|  | Module du 17.03.25 au 15.04.25 |

Table des matières

[1 Introduction et contexte du projet 4](#_Toc195605842)

[2 Analyse à faire complètement avec EA 5](#_Toc195605843)

[2.1 Use case client et use case Rest 5](#_Toc195605844)

[2.1.1 Use case client 1 5](#_Toc195605845)

[2.1.2 Use case client 2 6](#_Toc195605846)

[2.1.3 Use case Rest 1 7](#_Toc195605847)

[2.1.4 Use case Rest 2 8](#_Toc195605848)

[2.1.5 Use case GW 9](#_Toc195605849)

[2.2 Activity Diagram d'un cas complet 10](#_Toc195605850)

[2.2.1 API Rest 1 (ouvrir compétition) 10](#_Toc195605851)

[2.2.1 API Rest 2 (login) 10](#_Toc195605852)

[2.3 Maquettes du projet 11](#_Toc195605853)

[2.3.1 Login 11](#_Toc195605854)

[2.3.2 Admin 11](#_Toc195605855)

[2.3.3 Utilisateur 12](#_Toc195605856)

[2.4 Séquence System global entre les applications 13](#_Toc195605857)

[2.4.1 API Rest 1 (ouvrir compétition) 13](#_Toc195605858)

[2.4.2 API Rest 2 (login) 14](#_Toc195605859)

[3 Conception 16](#_Toc195605860)

[3.1 Diagramme de classes de chaque application 16](#_Toc195605861)

[3.1.1 Application 1 16](#_Toc195605862)

[3.1.2 Application 2 16](#_Toc195605863)

[3.1.3 API Gateway 18](#_Toc195605864)

[3.1.4 API Rest 1 19](#_Toc195605865)

[3.1.5 API Rest 2 20](#_Toc195605866)

[3.2 Navigation Diagram complète des applications 20](#_Toc195605867)

[3.2.1 Application 1 20](#_Toc195605868)

[3.2.2 Application 2 22](#_Toc195605869)

[4 Bases de données 23](#_Toc195605870)

[4.1 Modèles ER 23](#_Toc195605871)

[4.2 Modèles Workbench MySQL 23](#_Toc195605872)

[4.3 DB 1 23](#_Toc195605873)

[4.4 DB 2 24](#_Toc195605874)

[5 Implémentation des applications 25](#_Toc195605875)

[5.1 Une descente de code globale (depuis client 2 vers Rest 1 en passant par la GW) 25](#_Toc195605876)

[6 Outils, langages (versions, définitions, installations) 28](#_Toc195605877)

[7 Tests de fonctionnement du projet 29](#_Toc195605878)

[8 Auto-évaluations et conclusions 32](#_Toc195605879)

[8.1 Auto-évaluation et conclusion de client 1 32](#_Toc195605880)

[8.2 Auto-évaluation et conclusion de client 2 32](#_Toc195605881)

# Introduction et contexte du projet

Dans le cadre du module 133, nous devons mettre en place, par binôme de deux, un site WEB utilisant un backend en java via Spring Boot. Ce backend sera constitué de plusieurs Endpoint et utilisant un système de session.

Notre projet portera sur des concours d’horlogerie où il sera possible, pour l’administrateur, de créer des compétitions et de les gérer ainsi que leurs participants. L’utilisateur pourra voter pour un participant d’une compétition et y participer s’il le souhaite.

# Analyse à faire complètement avec EA

## Use case client et use case Rest

### Use case client 1

Voici le diagramme use case du client 1

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

Le contenu généré par l’IA peut être incorrect.

|  |  |
| --- | --- |
| Acteurs | Description |
| Visiteur | **Login :** Le visiteur peut se connecter à un compte administrateur. |
| Admin | **getCompetitions :** Il pourra récupérer toute la liste des compétitions |
| **Ouvrir competition :** Il pourra ouvrir une nouvelle compétition |
| **Supprimer compétition :** Il pourra supprimer une compétition |
| **Désinscrire un participant :** Il pourra désinscrire un participant d’une compétition |
| **Changer état compétition : il pourra modifier l’état d’une compétition (inscriptions, votes, terminé)** |
| **Logout :** Il pourra se déconnecter |

### Use case client 2

Voici le diagramme de use case du client 2 :

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

Le contenu généré par l’IA peut être incorrect.

Voici les actions que peuvent faire un visiteur et un utilisateur :

|  |  |
| --- | --- |
| Acteurs | Actions |
| Visiteur | Login : le visiteur peut se connecter à son compte s’il en possède déjà un. |
| S’enregistrer : le visiteur peut s’enregistrer pour créer un nouveau compte. |
| Utilisateur | Se déconnecter : l’utilisateur peut se déconnecter de son compte. |
| Get compétitions : l’utilisateur obtient la liste des compétitions en cours dès sa connexion/inscription. |
| Participer à une compétition : l’utilisateur peut choisir une ou plusieurs compétitions dans laquelle il veut participer. Il n’est pas obligé de participer à une compétition. |
| Voter : l’utilisateur peut voter pour un autre participant pour chaque compétition |

### Use case Rest 1

Voici le diagramme de use case de l’API Rest 1 :

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

Le contenu généré par l’IA peut être incorrect.

Voici un tableau récapitulatif des acteurs et de leurs actions :

|  |  |
| --- | --- |
| Acteurs | Actions |
| API Gateway | Désinscrire un participant : cette action contactera la db car pour chaque compétitions, leurs participants sont enregistrés. |
| Ouvrir une compétition : cette action ajoutera une nouvelle compétition dans la db. |
| Supprimer une compétition : cette action supprimera la compétition de la db. |
| Changer l’état d’une compétition : cette action met à jour une compétition dans la db en changeant la valeur de la colonne « état ». |

### Use case Rest 2

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

Le contenu généré par l’IA peut être incorrect.

|  |  |
| --- | --- |
| Acteurs | Actions |
| ApiGateway | Login : il peut vérifier les informations de login indiquées. |
| S’enregistrer : il peut ajouter un nouvel utilisateur avec les informations de login indiquées. |
| Utilisateur | Get compétitions : il peut récupérer la liste des compétitions dans la DB |
| Voter : il peut ajouter le vote d’un utilisateur pour le participant de la compétition indiqué. |
| Participer à compétition : il peut inscrire l’utilisateur indiqué à une compétition. |

### Use case GW

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

Le contenu généré par l’IA peut être incorrect.

Ce diagramme reprend les éléments des use-case précédent. Il fait simplement le pont entre le client et l’API adéquate.

## Activity Diagram d'un cas complet

### API Rest 1 (ouvrir compétition)

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

Le contenu généré par l’IA peut être incorrect.

Ce diagramme représente l’ouverture d’une compétition par un administrateur.

### API Rest 2 (login)

Ce diagramme d’activités représente le login d’un visiteur :

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

Le contenu généré par l’IA peut être incorrect.

Les actions effectuées dans ce diagramme sont les mêmes que celles dans le diagramme de séquences.

## Maquettes du projet

### Login

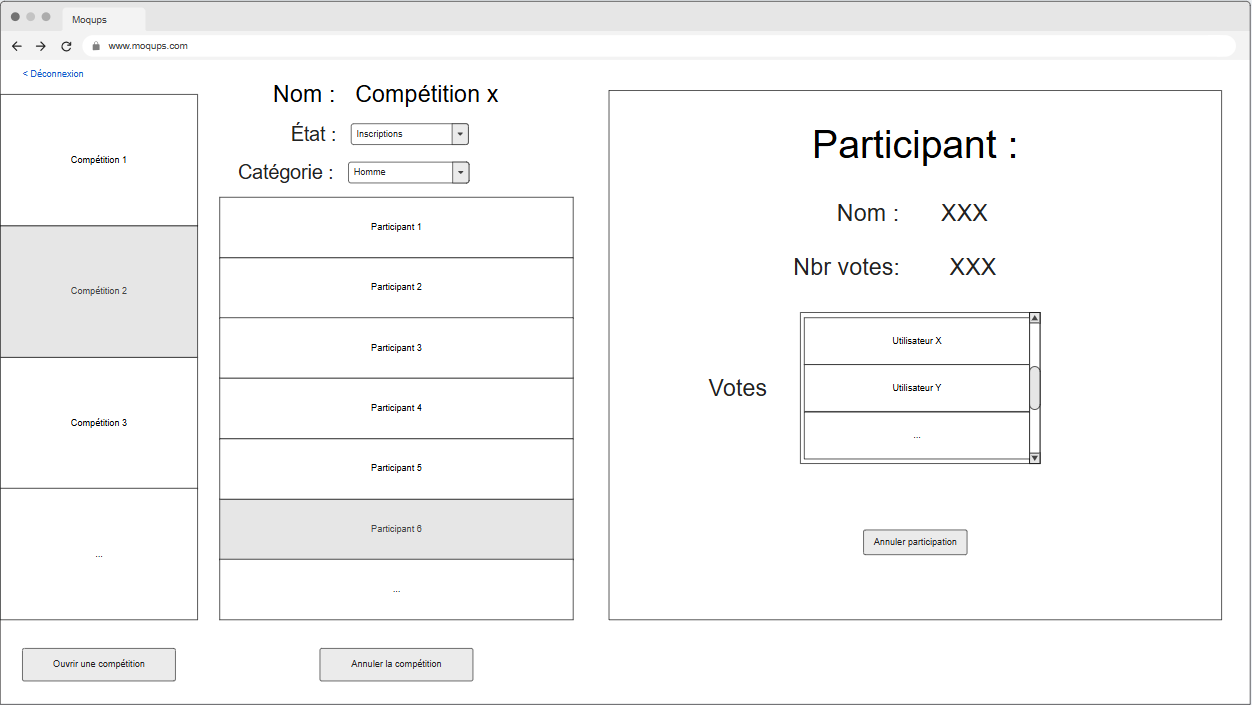
Voici la page de login pour tous les visiteurs :

Une image contenant texte, logiciel, Icône d’ordinateur, Logiciel multimédia

Le contenu généré par l’IA peut être incorrect.

Le visiteur peut se connecter ou s’enregistrer s’il n’a pas de compte.

### Admin



L’administrateur pourra ajouter de nouvelles compétitions, supprimer les compétitions et modifier leur état ou leur catégorie. L’administrateur peut aussi annuler la participation d’un utilisateur.

### Utilisateur

Voici la maquette pour la vue utilisateur :

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

Le contenu généré par l’IA peut être incorrect.

L’utilisateur peut naviguer entre les compétitions pour voir leur catégorie, leur état et leurs participants.

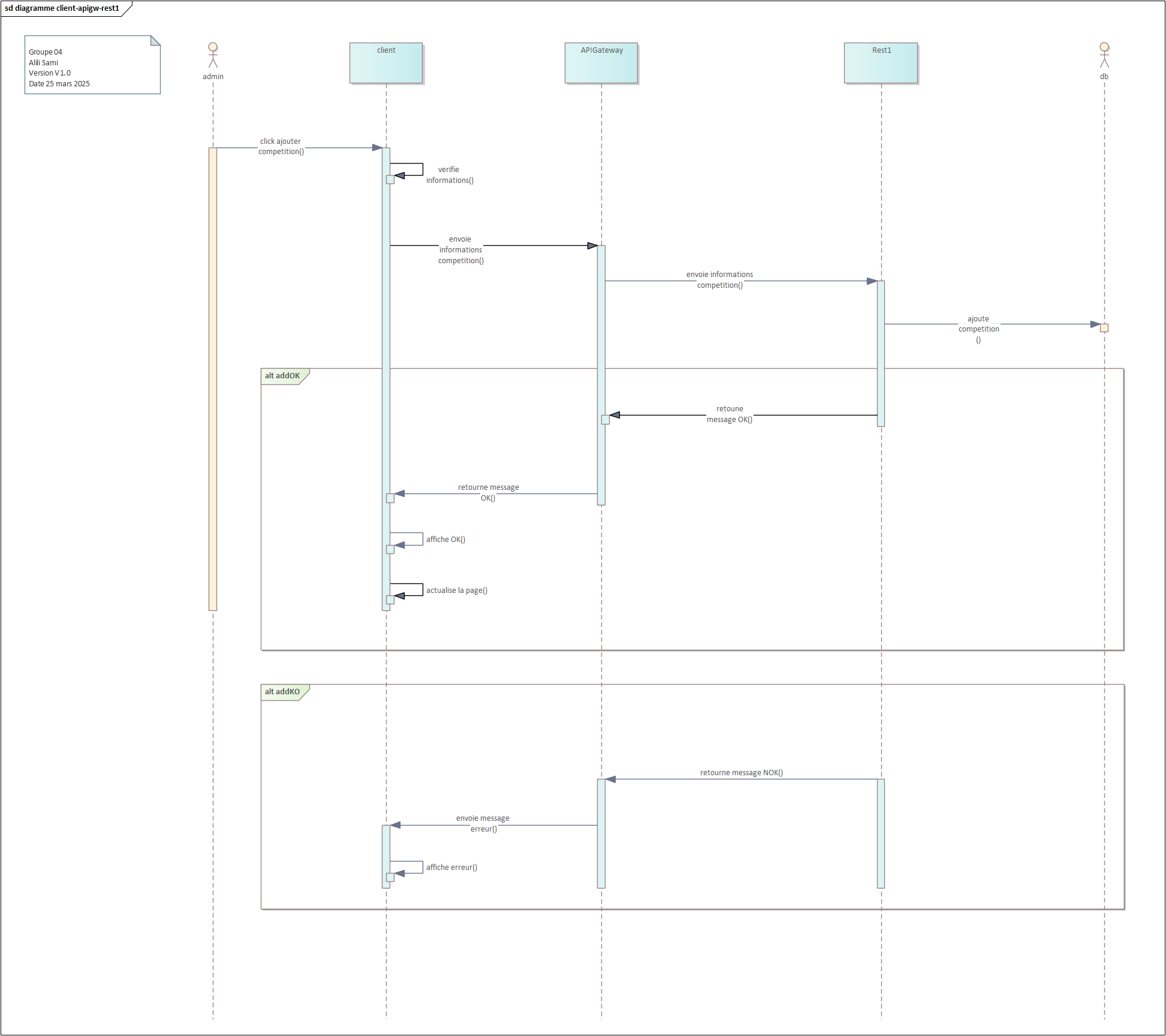
Il peut participer à une compétition si l’état de cette dernière le lui permet.

Il peut également naviguer entre les participants d’une compétition pour voir leur nom, nombres de votes et les utilisateurs ayant voté pour ce dernier.

Il peut voter pour un participant par compétition et annuler son vote s’il veut voter un autre participant.

## Séquence System global entre les applications

### API Rest 1 (ouvrir compétition)



Ce diagramme représente l’ouverture d’une compétition par un administrateur. Ça se déroulera de la manière suivante :

* L’administrateur appuie sur le bouton pour ajouter une compétition.
* L’application client vérifie si les données sont valides.
* L’application client envoie ensuite les informations de la nouvelle compétition à l’ApiGateway.
* L’ApiGateway envoie ensuite les informations au Rest 1.
* Le Rest 1 ajoute la compétition à la DB.
* L’ajout est OK :
  + Retourne un message de succès à l’ApiGateway.
  + L’ApiGateway renvoie le message au client.
  + L’application client affiche le message de succès.
  + L’application rafraichi la liste des compétitions.
* L’ajout est NOK :
  + Le Rest 1 envoie un message d’erreur à l’ApiGateway.
  + L’ApiGateway renvoie le message d’erreur au client.
  + Le client affiche un message d’erreur.

### API Rest 2 (login)

Voici le diagramme de séquences de l’API Rest 2 qui reprend l’action « login ».

Une image contenant texte, diagramme, Parallèle, Plan

Le contenu généré par l’IA peut être incorrect.

* Le visiteur clique sur le bouton du login.
* L’application cliente vérifie les informations entrées dans les champs nécessaires au login.
* Si les champs ne sont pas vides, l’application cliente envoie les informations à l’ApiGateway qui les envoie au Rest 2.
* Le Rest 2 récupère l’utilisateur en question auprès de la DB.
* Le Rest 2 teste si le mot de passe est correct.
* Le login est OK :
  + Retourne un message de succès à l’ApiGateway.
  + L’ApiGateway ouvre une nouvelle session.
  + L’ApiGateway renvoie le message au client.
  + L’application cliente ouvre la page de l’utilisateur.
* L’ajout est NOK :
  + Le Rest 2 envoie un message d’erreur à l’ApiGateway.
  + L’ApiGateway renvoie le message d’erreur au cliente.
  + Le client affiche un message d’erreur.

# Conception

## Diagramme de classes de chaque application

### Application 1

Voici le diagramme de classe de l’application client 1 :

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

Le contenu généré par l’IA peut être incorrect.

Dans ce diagramme il y a un fichier index, une connexion et un administrateur. Ces fichiers correspondent aux différentes pages de notre application.

Il y a, pour chacun de ces fichiers, un controller.js, et un worker.js commun à tous les controller.

Il y a un fichier douanier (le pont entre les fichiers javascript et java).

### Application 2

Voici le diagramme de classe de l’application client 2 :

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

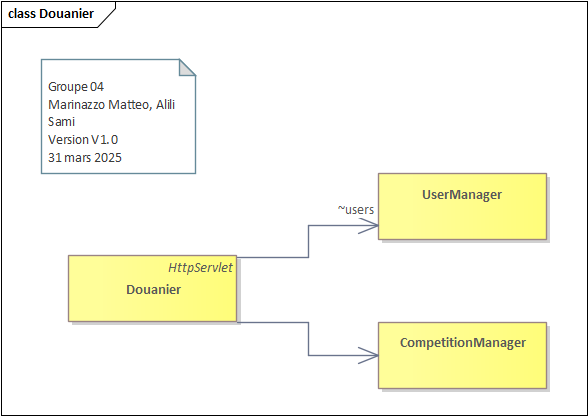
Le contenu généré par l’IA peut être incorrect.

Dans ce diagramme il y a un fichier index, une connexion et un utilisateur. Ces fichiers correspondent aux différentes pages de notre application.

Il y a, pour chacun de ces fichiers, un controller.js, et un worker.js commun à tous les controller.

Il y a un fichier douanier (le pont entre les fichiers javascript et java).

### API Gateway



Ceci représente le diagramme de classe ApiGateway. Elle contiendra la classe Douanier qui appellera les manager adéquat et qui appelleront ensuite les bon Rest.

### API Rest 1

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

Le contenu généré par l’IA peut être incorrect.

Ce diagramme représente le diagramme de classe Rest1. Il contient le point d’entré CtrlRest1 qui utilisera les 2 services afin d’effectuer certaines actions sur les model à l’aide des repository. Ces services retourneront les DTO et non les model afin de gérer les informations retournées.

### API Rest 2

Une image contenant texte, capture d’écran, diagramme, Post-it

Le contenu généré par l’IA peut être incorrect.

Le diagramme de classes ci-dessus est un diagramme de classes de l’API Rest 2 contenant 11 fichiers.

Un dto utilisateur, un service utilisateurService, voteService, participationService.

Il y a aussi un repository pour le model Utilisateur, Vote, Participation et un controller qui contrôle tout ça.

## Navigation Diagram complète des applications

Dans le cadre de la conception de notre projet il faut créer un diagramme de navigation pour chacune des applications de notre projet.

### Application 1

Voici le digramme de navigation pour la première application :

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

Le contenu généré par l’IA peut être incorrect.

Au lancement de l’application le visiteur arrive sur la page de connexion où il peut se connecter en tant qu’administrateur. Si le mot de passe est valide, une session est créée et le visiteur devient administrateur et est redirigé vers une autre page, où il pourra faire diverses actions.

Les actions se traduisent par des redirections ou envoies de données. L’action « modification compétition » envoie la modification en JSON à l’ApiGW qui lui l’envoie vers l’application Rest pour mettre à jour la base de données.

La déconnexion de l’administrateur se traduit par une redirection vers la page de connexion.

Le principe est répété pour toutes les actions que l’administrateur peut faire.

### Application 2

Voici le digramme de navigation pour la première application :

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

Le contenu généré par l’IA peut être incorrect.

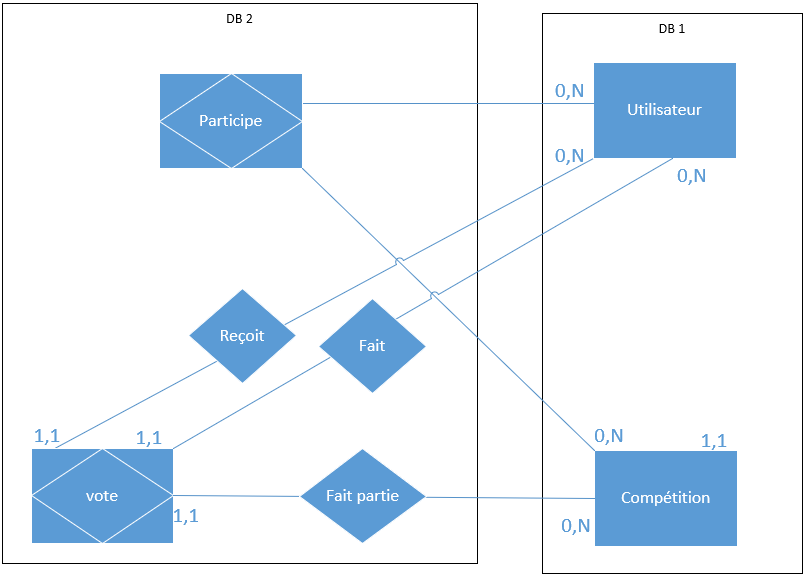
Au lancement de l’application le visiteur arrive sur la page de connexion où il peut se connecter en tant qu’utilisateur ou s’enregistrer. Si les informations sont valides, une session est créée et le visiteur devient un utilisateur et est redirigé vers sa page utilisateur où il pourra effectuer plusieurs actions.

Par exemple, l’action « Participer à une compétition » envoie la demande de participation à l’ApiGW qui l’envoie à son tour vers l’application Rest pour qu’elle ajoute le vote à la base de données.

# Bases de données

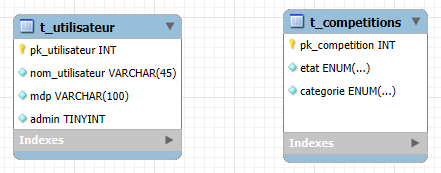
## Modèles ER

Voici le modèle ER de nos deux bases de données :

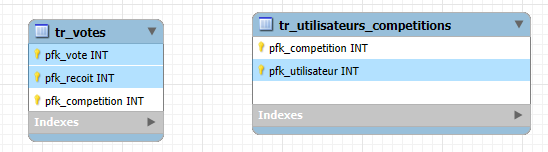


## Modèles Workbench MySQL

## DB 1



## DB 2



# Implémentation des applications

## Une descente de code globale (depuis client 2 vers Rest 1 en passant par la GW)

Dans le constructeur de « userView.js », il y a l’appel de la méthode « getCompetitions ».

constructor() {

        $(".competition-details-section").hide();

        getCompetitions(

            (x, y, z) => this.getCompetitionsSuccess(x, y, z),

            this.getCompetitionsError

        );

    }

Cette descente de code débute donc au moment où l’utilisateur s’est connecté et que la page a chargée.

Cette méthode appelle directement une méthode du serviceHttp.js.

function getCompetitions(successCallback, errorCallback) {

    $.ajax({

        type: "GET",

        dataType: "json",

        url: BASE\_URL + "getCompetitions",

        success: successCallback,

        error: errorCallback,

    });

}

Cette méthode envoie la requête au backend. En cas de succès la méthode « getCompetitionsSuccess » sera exécutée, sinon ce sera « getCompetitionsError ».

Ensuite arrivé dans l’Api GW, nous allons faire notre requête en direction du rest1 avec les paramètres indiqués depuis le client.

public ResponseEntity<Map> getCompetitions(int idCompetition) {

    String url = USER\_SERVICE\_URL + "getCompetitions";

    MultiValueMap<String, String> params = new LinkedMultiValueMap<>();

    params.add("participants", Boolean.TRUE.toString());

    if (idCompetition != -1) {

        params.add("idCompetition", Integer.toString(idCompetition));

    }

    String urlFinal = UriComponentsBuilder.fromUriString(url)

            .queryParams(params)

            .toUriString();

    return restTemplate.getForEntity(urlFinal, Map.class);

}

Ensuite, la requête arrive dans le Rest1. Pour commencer, il va vérifier si la requête demande une compétition spécifique ou toutes les compétitions et si besoin, il va faire des requêtes vers le Rest2 afin de récupérer tous les participants et leurs votes.

@GetMapping("/getCompetitions")

    public ResponseEntity<Map<String, Object>> getCompetitions(

            @RequestParam(required = false) Integer idCompetition,

            @RequestParam(required = false, defaultValue = "false") boolean participants) {

        if (idCompetition == null) {

            List<CompetitionDTO> comps = competitionService.getCompetitions();

            if (!comps.isEmpty() && participants) {

                comps.forEach((comp) -> addParticipantsCompetition(comp));

            }

            return ResponseEntity.ok(Map.<String, Object>of("data", comps));

        }

        CompetitionDTO comp = competitionService.getCompetitionAvecId(idCompetition.intValue());

        if (comp != null && participants) {

            addParticipantsCompetition(comp);

        }

        Map<String, Object> rep = new HashMap<>();

        rep.put("data", comp);

        return ResponseEntity.ok(rep);

    }

S’il n’y a eu aucune erreurs, la méthode de succès du client effectue ce code :

    getCompetitionsSuccess(data, text, jqXHR) {

        this.#competitions = data.data;

        $(".competitions-list").empty();

        this.#competitions.forEach((competition) => {

            let divComp = $("<div>");

            divComp.addClass("competition-item");

            divComp = divComp.text(competition.nom);

            divComp.click(() => this.afficherCompetitionInfos(competition));

            $(".competitions-list").append(divComp);

        });

    }

On commence par créer une variable privée contenant la liste des compétitions.

Ensuite on vide la liste des compétitions visibles sur la page WEB, pour éviter de créer des doublons par exemple.

Ensuite, pour chaque compétition dans la variable privée, on créer une nouvelle balise <div>, avec une classe, un texte et un écouteur qui écoute le clic, ce qui permettra d’y affecter un CSS, que son nom soit visible par l’utilisateur et qu’il puisse cliquer dessus pour voir les informations de la compétition.

Pour finir on ajoute cette nouvelle compétition à la vue.

Dans le cas où une erreur est retournée, cette méthode sera appelée :

    getCompetitionsError(jqXHR, textStatus, errorThrown) {

        alert("Erreur lors de la connexion : " + textStatus);

    }

Cette dernière affiche simplement une popup avec un message d’erreur.

# Outils, langages (versions, définitions, installations)

Voici la liste des outils et langages utilisés lors du projet de ce module :

* Outils
  + Visual Studio Code, version 1.99
  + Docker, version 4.39.0
  + MySQL Workbench, version 8.0 CE
  + WSL, version 2
* Backend
  + Java, version 17
  + Maven, version 3.8.4
  + MySQL, version 8
* Frontend
  + HTML, version 5
  + CSS, version 2
  + JavaScript, version 13
  + JQuery, version 3.7.0

# Tests de fonctionnement du projet

Voici une liste non-exhaustive des tests effectués sur notre projet :

|  |  |  |  |
| --- | --- | --- | --- |
| N° du test | Description | Résultat attendu | Résultat obtenu |
| 1 | Participer à une compétition qui a l’état participation | OK | OK |
| Capture d’écran | | |
|  | | |
| N° du test | Description | Résultat attendu | Résultat obtenu |
| 2 | Participer à une compétition qui n’a pas l’état participation | NOK | NOK |
| Capture d’écran | | |
|  | | |
| N° du test | Description | Résultat attendu | Résultat obtenu |
| 3 | Voter dans une compétition qui a l’état votation | OK | OK |
| Capture d’écran | | |
|  | | |
| N° du test | Description | Résultat attendu | Résultat obtenu |
| 4 | Voter dans une compétition qui n’a pas l’état votation | NOK | NOK |
| Capture d’écran | | |
|  | | |
| N° du test | Description | Résultat attendu | Résultat obtenu |
| 5 | Se déconnecter d’un user | OK | OK |
| Capture d’écran | | |
|  | | |
| N° du test | Description | Résultat attendu | Résultat obtenu |
| 6 | Ouvrir une compétition | OK | OK |
| Capture d’écran | | |
|  | | |
| N° du test | Description | Résultat attendu | Résultat obtenu |
| 7 | Changer l’état d’une compétition | OK | OK |
| Capture d’écran | | |
|  | | |
| N° du test | Description | Résultat attendu | Résultat obtenu |
| 8 | Supprimer compétition | OK | OK |
| Capture d’écran | | |
|  | | |
| N° du test | Description | Résultat attendu | Résultat obtenu |
| 9 | Désinscrire un participant | OK | OK |
| Capture d’écran | | |
|  | | |
| N° du test | Description | Résultat attendu | Résultat obtenu |
| 10 | Login avec un user sur la page admin | NOK | NOK |
| Capture d’écran | | |
|  | | |

# Auto-évaluations et conclusions

## Auto-évaluation et conclusion de client 1

J’ai apprécié ce module, il n’était pas spécialement difficile. Cependant, pour le projet, nous avons rencontré plus de problèmes.

## Auto-évaluation et conclusion de client 2

J’ai beaucoup aimé ce module malgré les difficultés que j’ai eu avec certains exercices. Néanmoins, ils étaient utiles et m’ont permis de comprendre pas mal de choses avant de me lancer dans l’implémentation de ce projet.

Je trouve que j’ai acquis la plupart des objectifs de ce module car j’ai réussi à implémenter ma partie du projet correctement. La seule chose que je n’ai pas réellement acquis est la gestion des sessions vu que ce n’était pas dans ma partie. Outre le projet, j’ai effectué beaucoup de documentation et la plupart de la gestion de projet.