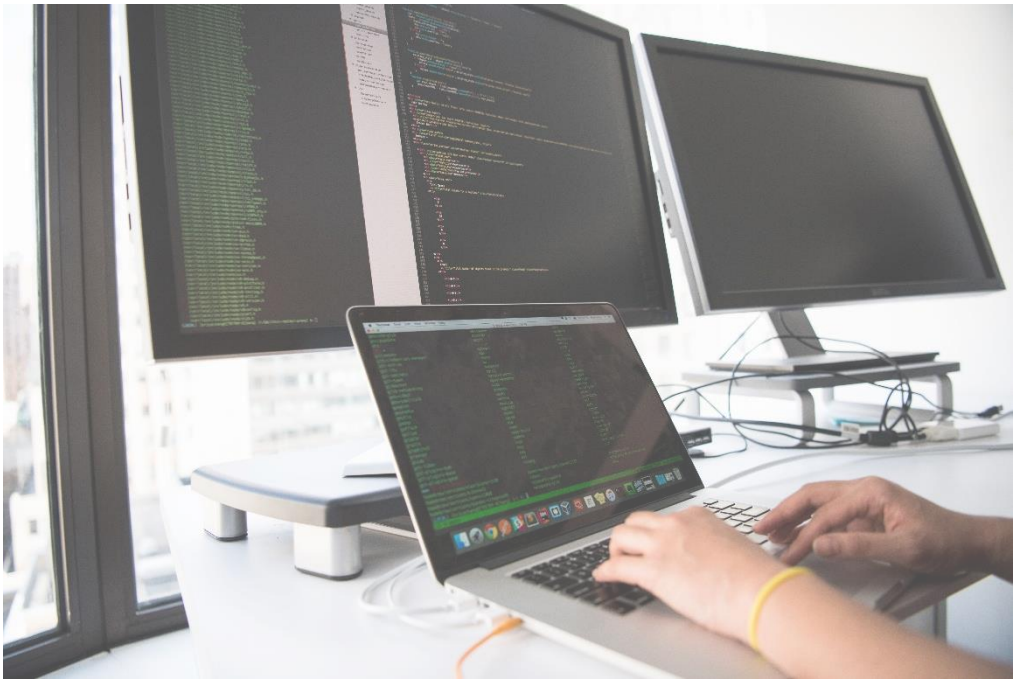
Formation

Architecture N-Tiers



Sébastien PHILIPPOT

Contents

1

Présentation du projet QUIZZ

Dans ce document, nous allons nous baser sur un projet de type QUIZZ pour illustrer les concepts vus au cours de la formation Spring. Ce projet sera notre fil conducteur pour expliquer les différentes étapes de la mise en place d'un projet avec Spring Boot.

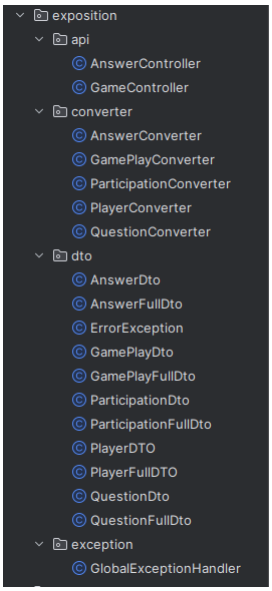
Le projet QUIZZ est conçu pour fournir une API REST qui permet à des applications tierces de communiquer avec les services proposés. Ce projet a pour but de créer une plateforme interactive où les utilisateurs peuvent participer à des quizz sur différents sujets. Les principales fonctionnalités de l'application incluent :

- Création et gestion de parties de quizz (GamePlay).
- Suivi de leurs scores (Player).
- Possibilité pour les joueurs de participer à différentes parties.
- Gestion des questions et des réponses pour chaque quizz.
- Fourniture d'une interface API REST pour intégrer facilement l'application avec d'autres systèmes ou applications front-end.

2

Couche exposition

2.1 Le projet QUIZZ



- **api** : Contient les contrôleurs qui gèrent les requêtes HTTP et retournent des réponses.
  - AnswerController gère la soumission des réponses aux questions.
  - GameController contrôle la logique de jeu, comme la création d'une nou-

velle partie.

- **converter** : Contient les classes nécessaires pour convertir les objets de transfert de données en entités de domaine et vice versa.
  - AnswerConverter convertit les données de réponse en entités métiers. Ces entités sont également converties au format attendu par le client.
  - GamePlayConverter transforme les données de jeu en entités. Ces entités sont également converties au format attendu par le client.
- **dto** : Contient des objets de transfert de données simplifiés utilisés entre les couches de l'application.
  - AnswerDto contient les informations nécessaires pour afficher les détails d'une réponse sans son identifiant.
  - PlayerDto présente les informations essentielles d'un joueur pour les listes ou les sommaires.
- **exception** : Gère les exceptions et les erreurs qui surviennent pendant le fonctionnement de l'application.
  - GlobalExceptionHandler capture et traite les exceptions globales et les mappe aux codes d'erreur HTTP appropriés.

2.1.1 dto

Un DTO (Data Transfert Object) est un objet dédié au transfert de l'application vers un client (application mobile, navigateur web etc...). Il a une structure qui correspond aux données nécessaires pour une opération particulière et ne contient pas de logique métier. Ainsi pour l'opération *afficher une liste de produit* cela correspond a un DTO avec seulement les attributs nécessaires pour cette intention. Si une seconde opération consiste a afficher les détails d'un produit cela conduit à un second DTO avec les attributs nécessaire pour cette opération.

Nous n'envoyons jamais l'entité à une application cliente, mais seulement une copie limitée aux besoins

```
package com.example.TPQUIZZ.exposition.dto;
```

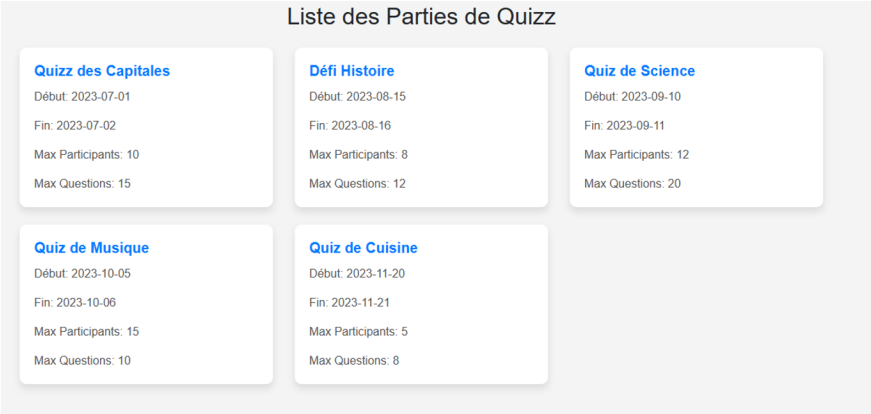
```
import java.util.Date;

public class GamePlayDto {

    private Long id;
    private String nom;
    private Date dateDebut;
    private Date dateFin;
    private int nbreMaxParticipants;
    private int nbreMaxQuestion;

    public GamePlayDto() {
        // Constructeur par d faut  n cessaire pour la d s rialisation
            JSON
    }

    public GamePlayDto(Long id, String nom, Date dateDebut, Date dateFin,
         int nbreMaxParticipants, int nbreMaxQuestion) {
        this.id = id;
        this.nom = nom;
        this.dateDebut = dateDebut;
        this.dateFin = dateFin;
        this.nbreMaxParticipants = nbreMaxParticipants;
        this.nbreMaxQuestion = nbreMaxQuestion;
    }

    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
    }

    public String getNom() {
        return nom;
    }

    public void setNom(String nom) {
        this.nom = nom;
    }

    public Date getDateDebut() {
        return dateDebut;
    }

    public void setDateDebut(Date dateDebut) {
        this.dateDebut = dateDebut;
    }

    public Date getDateFin() {
        return dateFin;
    }

    }
```

```
    public void setDateFin(Date dateFin) {
        this.dateFin = dateFin;
    }

    public int getNbreMaxParticipants() {
        return nbreMaxParticipants;
    }

    public void setNbreMaxParticipants(int nbreMaxParticipants) {
        this.nbreMaxParticipants = nbreMaxParticipants;
    }

    public int getNbreMaxQuestion() {
        return nbreMaxQuestion;
    }

    public void setNbreMaxQuestion(int nbreMaxQuestion) {
        this.nbreMaxQuestion = nbreMaxQuestion;
    }
}
```

GamePlayDto permet de répondre au besoin d'afficher une partie de façon réduite par exemple lors de la liste des parties.

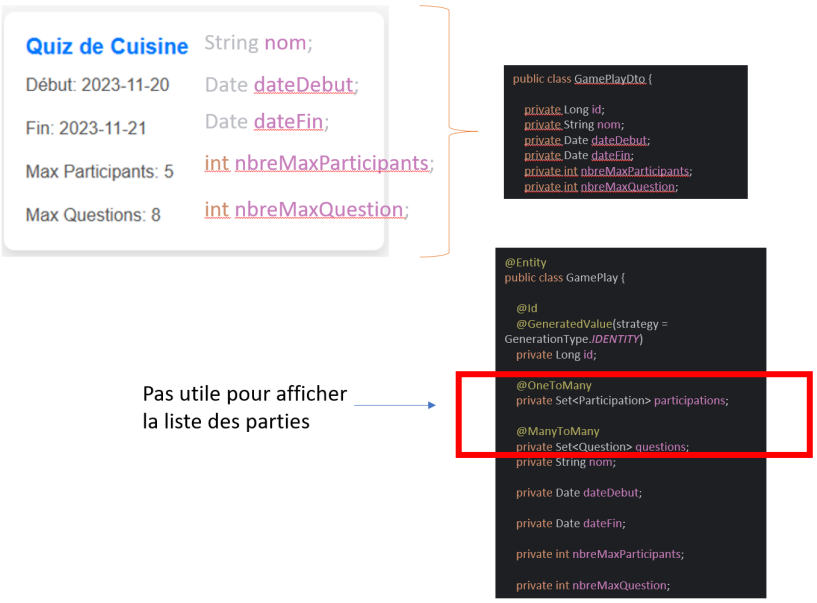
Cette classe contient deux constructeurs, un qui permet d'initialiser, l'ensemble des attributs et un par défaut vide.

Si nous n'avions pas cette classe, on devrait envoyer au front-end une liste de GamePlay, c'est à dire l'entité. Le front end affiche les parties de la manière suivante



On remarque que les propriétés affichées à l'utilisateur sont en nombre réduits par

rapport à l’entité GamePlay



C'est donc à cela que sert la classe *GamePlayDto*. Seules les propriétés nécessaires sont envoyées au front-end. On envoie pas l'entité mais une version réduite (*GamePlayDto*

L'id n'est pas affiché mais reste nécessaire pour pouvoir dire au back-end que l'on a sélectionné une partie (pour la modifier ou la supprimer par exemple)

. Le front-end peut avoir besoin de la totalité des propriétés de l'entité *GamePlay*. C'est le cas pour la création d'une partie.

Ici on voit bien qu'au moment de la création d'une partie, le front a besoin de remplir la liste des participants ainsi que des questions.

Dans ce cas nous avons utilisé un dto malgré le fait que l'entité pourrait correspondre car elle contient toutes les propriétés. En passant par un DTO, nous avons une séparation entre l'objet attendu par le front end et l'objet métier qui sera stocké en base de données. Ici, l'objet attendu par le front-end contient les même propriétés que l'objet métier, en les séparant ces deux objets peuvent évoluer différemment l'un de l'autre.

```
package com.example.TPQUIZZ.exposition.dto;


import java.util.Date;
import java.util.Set;

public class GamePlayFullDto extends GamePlayDto {

    private Set<ParticipationDto> participations;
    private Set<QuestionDto> questions;

    public GamePlayFullDto() {
        // Constructeur par d faut  n cessaire pour la d s rialisation
             JSON
    }

    public GamePlayFullDto(
            Long id, String nom, Date dateDebut, Date dateFin,
            int nbreMaxParticipants, int nbreMaxQuestion,
            Set<ParticipationDto> participations, Set<QuestionDto>
                questions
    ) {
```

```
        super(id, nom, dateDebut, dateFin, nbreMaxParticipants,
            nbreMaxQuestion);
        this.participations = participations;
        this.questions = questions;
    }

    // Getters et setters pour participations et questions

    public Set<ParticipationDto> getParticipations() {
        return participations;
    }

    public void setParticipations(Set<ParticipationDto> participations) {
        this.participations = participations;
    }

    public Set<QuestionDto> getQuestions() {
        return questions;
    }

    public void setQuestions(Set<QuestionDto> questions) {
        this.questions = questions;
    }
}
```

Cette classe reprends les attribut de GamePlayDto (*extends GamePlayDto*) et apporte des propriétés supplémentaires *participation* et *Questions*

2.1.2 exceptionHandler

```
package com.example.TPQUIZZ.exposition.exception;

import com.example.TPQUIZZ.application.exception.GamePlayFormatException;
import com.example.TPQUIZZ.application.exception.
    GamePlayNotFoundException;
import com.example.TPQUIZZ.exposition.dto.ErrorException;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.ControllerAdvice;
import org.springframework.web.bind.annotation.ExceptionHandler;

@ControllerAdvice
public class GlobalExceptionHandler {

    @Value("${agence}")
    String codeAgence;
    @ExceptionHandler(GamePlayFormatException.class)
    public ResponseEntity<ErrorException> handleGamePlayFormatException(
        GamePlayFormatException ex){
```

```
        ErrorException error=new ErrorException();
        error.setCodeAgence(codeAgence);
        error.setMessage(ex.getMessage());
        error.setCodeErreur("BAD_FORMAT");

        //envoyer l'objet error vers un syst me de stat

        //envoyer un mail a l'admin

        return new ResponseEntity<>(error,HttpStatus.BAD_GATEWAY);
    }

    @ExceptionHandler(GamePlayNotFoundException.class)
    public ResponseEntity<ErrorException> handleGamePlayNotFoundException
        (GamePlayNotFoundException ex){
        ErrorException error=new ErrorException();
        error.setCodeAgence(codeAgence);
        error.setMessage(ex.getMessage());
        error.setCodeErreur("NOT_FOUND");
        return new ResponseEntity<>(error,HttpStatus.NOT_FOUND);
    }

    @ExceptionHandler(Exception.class)
    public ResponseEntity<ErrorException> handleException(Exception ex){
        ErrorException error=new ErrorException();
        error.setCodeAgence(codeAgence);
        error.setMessage(ex.getMessage());
        error.setCodeErreur("GENERIC");
        return new ResponseEntity<>(error,HttpStatus.
            INTERNAL_SERVER_ERROR);
    }
}
```

Cette classe permet de transformer l'exception *GamePlayFormatException* en code erreur 502. Nous avons utilisé ErrorException pour transférer la réponse, mais c'est un choix nous aurions pu envoyer une chaîne de caractère et envoyer le contenu de l'exception. Ce dto permet simplement d'adapter la réponse à un client.

- **@ControllerAdvice** : Cette annotation indique que la classe est un contrôleur qui permet de gérer des exceptions.

- **@Value("${agence}") String codeAgence** : Injecte une valeur depuis le fichier de configuration (application.properties) de l'application dans la variable *codeAgence*.

- **@ExceptionHandler(GamePlayFormatException.class)** : Indique que la méthode suivante gère les exceptions de type *GamePlayFormatException*.

- **handleGamePlayFormatException** : Méthode pour gérer *GamePlayFormatException,* créant un objet *ErrorException* et renvoyant une réponse HTTP avec le statut *BAD_GATEWAY*.

- **Création de l'objet ErrorException** : Un nouvel objet *ErrorException* est créé avec des détails sur l'erreur.

- **Retour de ResponseEntity** : Retourne un objet *ResponseEntity* contenant l'objet *error* et un statut HTTP.

- **Autres méthodes @ExceptionHandler** : Méthodes similaires pour gérer d'autres types d'exceptions avec des statuts HTTP différents.

ErrorException est une classe représentant une erreur survenue au cours du programme. Elle contient des informations métiers.

```
package org.example.produit.dto;


/**
 * Cette classe permet d'avoir des informations suppl mentaires pour des
     exeptions
 * Elle n'est pas obligatoire
 */
public class ErrorException {

    private String message;
    private String codeErreur;
    private String codeAgence;

    public ErrorException(String message, String codeErreur, String
        codeAgence) {
        this.message = message;
        this.codeErreur = codeErreur;
        this.codeAgence = codeAgence;
    }

    public ErrorException() {
    }

    public String getMessage() {
        return message;
    }

    public void setMessage(String message) {
        this.message = message;
    }

    public String getCodeErreur() {
        return codeErreur;
    }
```

```
    public void setCodeErreur(String codeErreur) {
        this.codeErreur = codeErreur;
    }

    public String getCodeAgence() {
        return codeAgence;
    }

    public void setCodeAgence(String codeAgence) {
        this.codeAgence = codeAgence;
    }
}
```

Il s'agit d'un simple DTO avec :

- message représente le contenu de l'exception
- codeErreur qui représente un code pour situer l'erreur ici par exemple il prend pour valeur *NOT_FOUND*
- codeAgence représente l'agence sur laquelle l'erreur est survenue.

2.1.3 converter

```
package com.example.TPQUIZZ.exposition.converter;

import com.example.TPQUIZZ.domaine.GamePlay;
import com.example.TPQUIZZ.exposition.dto.GamePlayDto;
import com.example.TPQUIZZ.exposition.dto.GamePlayFullDto;
import org.modelmapper.ModelMapper;
import org.springframework.stereotype.Component;

@Component
public class GamePlayConverter {

    private final ModelMapper modelMapper;

    public GamePlayConverter(ModelMapper modelMapper) {
        this.modelMapper=modelMapper;

    }

    public GamePlayDto convertToDto(GamePlay gamePlay) {
        return modelMapper.map(gamePlay, GamePlayDto.class);
    }

    public GamePlayFullDto convertToFullDto(GamePlay gamePlay) {
```

```
            return modelMapper.map(gamePlay, GamePlayFullDto.class);
        }

        public GamePlay convertToEntity(GamePlayDto gamePlayDTO) {
            return modelMapper.map(gamePlayDTO, GamePlay.class);
        }

    }
```

Cette classe permet de faire la conversion d'une entité en DTO ainsi qu'un DTO en entité.

Cette classe est annotée @component c'est donc un objet que Spring va reconnaître automatiquement et qui sera mis à disposition du développeur. Celui ci n'aura pas à gérer son cycle de vie

- **@Component** : L'annotation *@Component* indique que c'est un composant Spring, ce qui permet son injection automatique dans d'autres parties de l'application.
- **Variable modelMapper** : Déclare une variable privée finale *modelMapper* de type *ModelMapper*.
- **Constructeur de GamePlayConverter** : Un constructeur qui initialise *modelMapper*.
- **Méthode convertToDto** : Méthode pour convertir un objet *GamePlay* en un objet *GamePlayDto* en utilisant *modelMapper*.
- **Méthode convertToEntity** : Méthode pour convertir un objet *GamePlayDto* en un objet *GamePlay*, également à l'aide de *modelMapper*.

2.1.4 api

```
package com.example.TPQUIZZ.exposition.api;

import com.example.TPQUIZZ.application.IGamePlayService;
import com.example.TPQUIZZ.application.exception.GamePlayFormatException;
import com.example.TPQUIZZ.application.exception.
    GamePlayNotFoundException;
import com.example.TPQUIZZ.domaine.GamePlay;
import com.example.TPQUIZZ.domaine.Participation;
import com.example.TPQUIZZ.domaine.Player;
import com.example.TPQUIZZ.exposition.converter.GamePlayConverter;
```

```
import com.example.TPQUIZZ.exposition.converter.PlayerConverter;
import com.example.TPQUIZZ.exposition.dto.GamePlayDto;
import com.example.TPQUIZZ.exposition.dto.GamePlayFullDto;
import com.example.TPQUIZZ.exposition.dto.PlayerDTO;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

import java.util.ArrayList;
import java.util.List;
import java.util.stream.Collectors;

@RestController
@RequestMapping("/game")
public class GameController {
 //------Injection de d pendances-----------------------------------
     //@Autowired
    //IGamePlayService service;

    //@Autowired
    //GamePlayConverter converter;

     //OU
    IGamePlayService service;

    GamePlayConverter converter;

    PlayerConverter playerConverter;


    public GameController(IGamePlayService service, GamePlayConverter
        converter,PlayerConverter playerConverter) {
        this.service = service;
        this.converter = converter;
        this.playerConverter = playerConverter;
    }

    @PostMapping("/create")
    public GamePlayFullDto createGamePlay(@RequestBody GamePlayFullDto
        gpDto) throws GamePlayFormatException {
        var gamePlay = converter.convertToEntity(gpDto);
        var savedGamePlay = service.createGamePlay(gamePlay);
        return converter.convertToFullDto(savedGamePlay);
    }

    @GetMapping("/all")
    public List<GamePlayDto> findAll() {
        var gamePlays = service.findAll();
        return gamePlays.stream()
                .map(converter::convertToDto)
                .collect(Collectors.toList());

        // ou avec JAVA 7
```

```
        /* List<GamePlayDto> listGamePlayDto=new ArrayList<>();
          for(GamePlay gp:service.findAll()){
              listGamePlayDto.add(converter.convertToDto(gp));
         }*/
    }

    @GetMapping("/{id}")
    public GamePlayFullDto findById(@PathVariable("id") Long id) throws
        GamePlayNotFoundException {
        var gamePlay = service.findById(id);
        return converter.convertToFullDto(gamePlay);
    }

    @PutMapping("/{id}")
    public GamePlayFullDto update(@PathVariable("id") Long id,
        @RequestBody GamePlayFullDto gpDto) throws
        GamePlayFormatException, GamePlayNotFoundException {
        var gamePlay = converter.convertToEntity(gpDto);
        var updatedGamePlay = service.update(id, gamePlay);
        return converter.convertToFullDto(updatedGamePlay);
    }

    @PatchMapping("/{id}/addPlayer")
    public GamePlayDto addPlayer(@PathVariable("id") Long id,
        @RequestBody PlayerDTO playerDto) throws
        GamePlayNotFoundException, GamePlayFormatException {
        var player = playerConverter.convertToEntity(playerDto);
        var updatedGamePlay = service.addPlayer(id, player);
        return converter.convertToDto(updatedGamePlay);
    }

    @PatchMapping("/{id}/removePlayer")
    public GamePlayDto removePlayer(@PathVariable("id") Long id,
        @RequestBody PlayerDTO playerDto) throws
        GamePlayNotFoundException {
        var player = playerConverter.convertToEntity(playerDto);
        var updatedGamePlay = service.removePlayer(id,player );
        return converter.convertToDto(updatedGamePlay);
    }

    @DeleteMapping("/{id}")
    public ResponseEntity<?> delete(@PathVariable("id") Long id) throws
        GamePlayNotFoundException {
        service.delete(id);
        return ResponseEntity.ok().build();
    }
}
```

Ce code est une classe de contrôleur de l'API REST pour un produit. Il utilise les annotations @RestController et @RequestMapping pour définir le chemin de l'API. La classe utilise également les annotations @Autowired pour injecter les dépen-

dances IProduitService et ProduitMapper. La classe utilise également les annotations @PostMapping et @GetMapping pour définir les méthodes de l'API pour la création et la récupération de produits. La méthode create() utilise @RequestBody pour récupérer les données d'un nouveau produit à partir d'une requête HTTP POST, et renvoie une réponse HTTP créée avec les données du produit. La méthode getAllProducts() renvoie la liste de tous les produits sous forme de ProduitDto. La méthode getProduitById() récupère un produit en fonction de son ID à l'aide de l'annotation @PathVariable et renvoie les données du produit sous forme de ProduitDto. Les activités de la classe sont enregistrées à l'aide du logger LoggerFactory.

- `@RestController` indique que cette classe va traiter les requêtes Web.
- `@RequestMapping("/game")` signifie que les méthodes de cette classe répondront aux requêtes sur l'URL commençant par `/game`.

2.1.5 Injection de Dépendances

La classe utilise l'injection de dépendances pour obtenir les objets nécessaires tels que `IGamePlayService`, `GamePlayConverter`, et `PlayerConverter`. L'injection de dépendance se fait à l'aide du constructeur de la classe. Nous aurions très bien pu utiliser l'annotation *@Autowired*.

```
@RestController
@RequestMapping("/game")
public class GameController {
 //------Injection de dpendances-----------------------------------
     @Autowired
    IGamePlayService service;

    @Autowired
    GamePlayConverter converter;

     @Autowired
     PlayerConverter playerConverter;

    @PostMapping("/create")
    public GamePlayFullDto createGamePlay(@RequestBody GamePlayFullDto
        gpDto) throws GamePlayFormatException {
        var gamePlay = converter.convertToEntity(gpDto);
        var savedGamePlay = service.createGamePlay(gamePlay);
        return converter.convertToFullDto(savedGamePlay);
    }
       ...
}
```

2.1.6 Méthodes du Contrôleur

Chaque méthode dans `GameController` correspond à une action spécifique de l'utilisateur:

- `@PostMapping("/create")` crée une nouvelle partie de quizz.
- `@GetMapping("/all")` renvoie toutes les parties disponibles.
- `@GetMapping("/id")` trouve une partie spécifique.
- `@PutMapping("/id")` met à jour une partie.
- `@PatchMapping("/{id}/addPlayer")`
- `@PatchMapping("/{id}/removePlayer")` gèrent l'ajout et le retrait des joueurs.
- `@DeleteMapping("/id")` supprime une partie.

2.1.7 Conversion DTO et Interaction avec le Service

Le contrôleur utilise des convertisseurs pour transformer les données entre les formats DTO et les entités, et interagit avec `IGamePlayService` pour exécuter la logique métier.

3

Couche application

3.1 les règles métiers

Nous avons vu que la classe *GamePlayController* à besoin d'un service concernant une partie. En effet elle a besoin d'avoir la liste des parties, de pouvoir créer une partie et de retrouver une partie par son id. Pour demander un service la seule chose a connaître c'est quels sont les éléments entrants, c'est à dire ce que doit fournir le contrôleur au service ainsi que ce qu'il va recevoir de ce service. Par exemple, lorsque j'amène une voiture en panne chez un garagiste je connais les entrants (une voiture en panne, le paiement, la cause ...) et ce que je vais recevoir en retour c'est une voiture réparée. Il n'y a pas besoin de savoir comment la réparation s'effectue. La couche service doit donc être abstraite pour la couche exposition. Il doit y avoir une interface entre les deux faisant office de contrat. Nous avons appelé cette interface *IGamePlayServiceService*

```
package com.example.TPQUIZZ.application;

import com.example.TPQUIZZ.application.exception.GamePlayFormatException;
import com.example.TPQUIZZ.application.exception.
    GamePlayNotFoundException;
import com.example.TPQUIZZ.domaine.GamePlay;
import com.example.TPQUIZZ.domaine.Player;

import java.util.List;

public interface IGamePlayService {

    GamePlay createGamePlay(GamePlay gp) throws GamePlayFormatException;
```

```
        List<GamePlay> findAll();

        GamePlay findById(Long id) throws GamePlayNotFoundException;

        GamePlay update(Long id, GamePlay gp) throws GamePlayFormatException,
             GamePlayNotFoundException;

        void delete(Long id) throws GamePlayNotFoundException;

        GamePlay addPlayer(Long id, Player player) throws
            GamePlayFormatException;

        GamePlay removePlayer(Long id, Player player);
    }
```

et son implémentation

```
  package com.example.TPQUIZZ.application;

import com.example.TPQUIZZ.application.exception.GamePlayFormatException;
import com.example.TPQUIZZ.application.exception.
    GamePlayNotFoundException;
import com.example.TPQUIZZ.domaine.GamePlay;
import com.example.TPQUIZZ.domaine.Participation;
import com.example.TPQUIZZ.domaine.Player;
import com.example.TPQUIZZ.domaine.Question;
import com.example.TPQUIZZ.infrastructure.GamePlayRepository;
import org.springframework.stereotype.Service;

import java.util.HashSet;
import java.util.List;
import java.util.Set;
import java.util.stream.Collectors;

@Service
public class IGamePlayServiceImpl implements IGamePlayService {

    GamePlayRepository gamePlayRepository;
    ParticipationService participationService;

    QuestionService questionService;

    public IGamePlayServiceImpl(GamePlayRepository repo,
        ParticipationService participationService,QuestionService
        questionService ) {
        this.gamePlayRepository = repo;
        this.participationService=participationService;
        this.questionService=questionService;
    }

    @Override
    public GamePlay createGamePlay(GamePlay gamePlay) throws
        GamePlayFormatException {

        // Validation de base
        if(gamePlay.getNbreMaxParticipants() <= 0) {
            throw new GamePlayFormatException("Le nombre de participants
                est obligatoire.");
        } else if(gamePlay.getDateFin().before(gamePlay.getDateDebut()))
            {
            throw new GamePlayFormatException("La date de fin est avant
                la date de d but.");
        }

        // Enregistrer les participations s par ment car nous n'avons
            pas mise de cascade sur les entit s
        Set<Participation> savedParticipations = new HashSet<>();
        for (Participation participation : gamePlay.getParticipations())
            {
            Participation savedParticipation = participationService.
```

```
                createParticipation(participation);
            savedParticipations.add(savedParticipation);
        }

        // Enregistrer les questions s par ment car nous n'avons pas
            mise de cascade sur les entit s
        Set<Question> savedQuestions = new HashSet<>();
        for (Question question : gamePlay.getQuestions()) {
            Question savedQuestion = questionService.createQuestion(
                question);
            savedQuestions.add(savedQuestion);
        }

        // Associer les participations et les questions enregistr es au
            GamePlay
        gamePlay.setParticipations(savedParticipations);
        gamePlay.setQuestions(savedQuestions);

        // Enregistrer le GamePlay
        return gamePlayRepository.save(gamePlay);
    }


    @Override
    public List<GamePlay> findAll() {
        return gamePlayRepository.findAll();
    }

    @Override
    public GamePlay findById(Long id) throws GamePlayNotFoundException {
        return gamePlayRepository.findById(id).orElseThrow(
                () ->  new GamePlayNotFoundException("Le gamePlay avec l'
                    id "+id+" n'existe pas"));
    }

    @Override
    public GamePlay update(Long id, GamePlay gp) throws
        GamePlayFormatException, GamePlayNotFoundException {
        GamePlay gpFromBdd=gamePlayRepository.findById(id).get();
        if(gp==null){
            throw new GamePlayNotFoundException("Le gamePlay avec l'id "+
                id+" n'existe pas");
        }
        gpFromBdd.setDateDebut(gp.getDateDebut());
        gpFromBdd.setNom(gp.getNom());
        if(gp.getParticipations()!=null ){
            gpFromBdd.setParticipations(gp.getParticipations());
        }

        if(gp.getNbreMaxParticipants()<=0){
            throw new GamePlayFormatException("le nombre de participant
                est obligatoire");
        }
```

```
        gpFromBdd.setNbreMaxParticipants(gp.getNbreMaxParticipants());
        if(gp.getQuestions()!=null ){
            gpFromBdd.setQuestions(gp.getQuestions());
        }
        gpFromBdd.setNbreMaxQuestion(gp.getNbreMaxQuestion());
        return gamePlayRepository.save(gpFromBdd);
    }

    @Override
    public void delete(Long id) throws GamePlayNotFoundException {
        GamePlay gpFromBdd=gamePlayRepository.findById(id).get();
        if(gpFromBdd==null){
            throw new GamePlayNotFoundException("Le gamePlay avec l'id "+
                id+" n'existe pas");
        }

        gamePlayRepository.delete(gpFromBdd);
    }

    @Override
    public GamePlay addPlayer(Long id, Player player) throws
        GamePlayFormatException {

        var gamePlay = findById(id);

        if(gamePlay.getParticipations().size()>gamePlay.
            getNbreMaxParticipants()){
            throw new GamePlayFormatException("nombre max depass ");
        }
        Participation participation=new Participation();
        participation.setPlayer(player);
        participation.setScoreValue(0);

        gamePlay.getParticipations().add(participation);

        return gamePlayRepository.save(gamePlay);

    }

    @Override
    public GamePlay removePlayer(Long id, Player player) {
        GamePlay gamePlay = findById(id);
        gamePlay.setParticipations(
                gamePlay.getParticipations().stream()
                        .filter(participation -> !participation.getPlayer
                            ().getId().equals(id))
                        .collect(Collectors.toSet())
        );
        return gamePlayRepository.save(gamePlay);
    }


}
```

3.1.1 Créer une nouvelle partie

Avant de créer une partie, la méthode vérifie que les données fournies par le controller sont valides :

- Le nombre de participants doit être supérieur à zéro.
- La date de fin du jeu doit être après la date de début.
- Si ces conditions ne sont pas remplies, une erreur de type *GamePlayFormatException* est signalée.

3.1.1.1 Enregistrement des Participations et Questions

La méthode enregistre ensuite séparément les participations et les questions car nous n'avons pas mis de cascade:

- Chaque participation et chaque question sont enregistrées une par une.
- Ces informations sont stockées pour être associées à la partie de quizz.

3.1.1.2 Association avec la Partie de Quizz

Après l'enregistrement, les participations et les questions sont associées à la partie de quizz :

- La méthode ajoute les participations et les questions à la partie de quizz.

3.1.1.3 Enregistrement Final

Enfin, la partie de quizz complète est enregistrée dans la base de données.

3.1.1.4 Recherche de la Partie de Quizz

La méthode commence par trouver la partie de quizz à laquelle le joueur doit être ajouté en utilisant son identifiant (ID).

3.1.2 Ajouter un joueur à une partie

3.1.2.1 Vérification du Nombre Maximum de Participants

Avant d'ajouter un joueur, la méthode vérifie si l'ajout d'un nouveau joueur ne dépasse pas le nombre maximum de participants autorisés pour cette partie :

- Si le nombre de participants actuels est déjà au maximum, une erreur est déclenchée pour empêcher l'ajout.

3.1.2.2 Création d'une Nouvelle Participation

Si l'ajout est possible, la méthode crée une nouvelle participation pour le joueur :

- Un objet `Participation` est créé et associé au joueur.
- Le score initial du joueur est défini à zéro.

3.1.2.3 Ajout de la Participation à la Partie

La nouvelle participation est ensuite ajoutée à la liste des participations de la partie de quizz.

3.1.2.4 Enregistrement des Modifications

Enfin, la partie de quizz mise à jour, avec le nouveau joueur, est enregistrée dans la base de données.

Les autres méthodes ne présentent pas de difficultés particulières.

4

Couche infrastructure

4.0.1 Les entités

```
package com.example.TPQUIZZ.domaine;

import jakarta.persistence.*;

@Entity
@Table(name="reponses")
public class Answer {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String reponseText;

    private Boolean isCorrect;

    @ManyToOne
    @JoinColumn(name="question_id")
    private Question question;

    public Answer(Long id, String reponseText, Boolean isCorrect,
        Question question) {
        this.id = id;
        this.reponseText = reponseText;
        this.isCorrect = isCorrect;
        this.question = question;
    }

    public Answer() {
```

```
    }

    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
    }

    public String getReponseText() {
        return reponseText;
    }

    public void setReponseText(String reponseText) {
        this.reponseText = reponseText;
    }

    public Boolean getCorrect() {
        return isCorrect;
    }

    public void setCorrect(Boolean correct) {
        isCorrect = correct;
    }

    public Question getQuestion() {
        return question;
    }

    public void setQuestion(Question question) {
        this.question = question;
    }
}
```

```
package com.example.TPQUIZZ.domaine;

import jakarta.persistence.*;

import java.util.Date;
import java.util.Set;

@Entity
public class GamePlay {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @OneToMany
    private Set<Participation> participations;

    @ManyToMany
```

```
private Set<Question> questions;
private String nom;

private Date dateDebut;

private Date dateFin;

private int nbreMaxParticipants;

private int nbreMaxQuestion;

public GamePlay(Long id, Set<Participation> participations, Set<
    Question> questions, String nom, Date dateDebut, Date dateFin,
    int nbreMaxParticipants, int nbreMaxQuestion) {
    this.id = id;
    this.participations = participations;
    this.questions = questions;
    this.nom = nom;
    this.dateDebut = dateDebut;
    this.dateFin = dateFin;
    this.nbreMaxParticipants = nbreMaxParticipants;
    this.nbreMaxQuestion = nbreMaxQuestion;
}

public GamePlay() {
}

public Long getId() {
    return id;
}

public void setId(Long id) {
    this.id = id;
}

public Set<Participation> getParticipations() {
    return participations;
}

public void setParticipations(Set<Participation> participations) {
    this.participations = participations;
}

public Set<Question> getQuestions() {
    return questions;
}

public void setQuestions(Set<Question> questions) {
    this.questions = questions;
}

public String getNom() {
    return nom;
```

```
    }

    public void setNom(String nom) {
        this.nom = nom;
    }

    public Date getDateDebut() {
        return dateDebut;
    }

    public void setDateDebut(Date dateDebut) {
        this.dateDebut = dateDebut;
    }

    public Date getDateFin() {
        return dateFin;
    }

    public void setDateFin(Date dateFin) {
        this.dateFin = dateFin;
    }

    public int getNbreMaxParticipants() {
        return nbreMaxParticipants;
    }

    public void setNbreMaxParticipants(int nbreMaxParticipants) {
        this.nbreMaxParticipants = nbreMaxParticipants;
    }

    public int getNbreMaxQuestion() {
        return nbreMaxQuestion;
    }

    public void setNbreMaxQuestion(int nbreMaxQuestion) {
        this.nbreMaxQuestion = nbreMaxQuestion;
    }
}
```

```
package com.example.TPQUIZZ.domaine;

import jakarta.persistence.*;

@Entity
public class Participation {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @OneToOne
    private Player player;

    int scoreValue;
```

```
    public Participation() {
    }

    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
    }

    public Player getPlayer() {
        return player;
    }

    public void setPlayer(Player player) {
        this.player = player;
    }

    public int getScoreValue() {
        return scoreValue;
    }

    public void setScoreValue(int scoreValue) {
        this.scoreValue = scoreValue;
    }
}
```

```
package com.example.TPQUIZZ.domaine;

import jakarta.persistence.*;

import java.util.List;

@Entity
public class Player {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String username;

    @OneToMany
    private List<GamePlay> gameHistory;
    private int score;

    public Player(Long id, String username, int score) {
        this.id = id;
        this.username = username;
        this.score = score;
    }
```

```
    public Player() {
    }

    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
    }

    public String getUsername() {
        return username;
    }

    public void setUsername(String username) {
        this.username = username;
    }

    public int getScore() {
        return score;
    }

    public void setScore(int score) {
        this.score = score;
    }
}
```

```
package com.example.TPQUIZZ.domaine;

import jakarta.persistence.*;

import java.util.List;

@Entity
@Table(name="questions")
public class Question {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String questionText;

    @OneToOne
    @JoinColumn(name = "correct_answer_id")
    private Answer correctAnswer;

    @OneToMany(mappedBy = "question")
    private List<Answer> answers;

    public Question(Long id, String questiontext, Answer correctAnswer,
```

```
            List<Answer> answers) {
        this.id = id;
        this.questionText = questiontext;
        this.correctAnswer = correctAnswer;
        this.answers = answers;
    }

    public Question() {
    }

    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
    }

    public String getQuestiontext() {
        return questionText;
    }

    public void setQuestiontext(String questiontext) {
        this.questionText = questiontext;
    }

    public Answer getCorrectAnswer() {
        return correctAnswer;
    }

    public void setCorrectAnswer(Answer correctAnswer) {
        this.correctAnswer = correctAnswer;
    }

    public List<Answer> getAnswers() {
        return answers;
    }

    public void setAnswers(List<Answer> answers) {
        this.answers = answers;
    }
}
```

Du point de vue métier, les relations entre les classes reflètent la structure logique de l'application de quizz :

- **GamePlay** représente une session ou une partie de quizz. Chaque partie peut comporter plusieurs questions et est ouverte à plusieurs participants.
- **Participation** lie un joueur spécifique (**Player**) à une partie de quizz

(**GamePlay**). Cela représente la participation d'un joueur à une session de quizz et enregistre son score pour cette session.

- **Player** est le profil d'un utilisateur dans l'application. Un joueur peut participer à plusieurs parties de quizz, d'où l'historique des parties (**GamePlay**) associées à chaque joueur.

- **Question** est un élément individuel d'un quizz. Chaque question appartient à une ou plusieurs parties de quizz et a un ensemble de réponses possibles (**Answer**).

- **Answer** représente une réponse possible à une question. Chaque réponse est associée à une question spécifique.

Techniquement, ces relations sont implémentées en utilisant des annotations JPA pour gérer les interactions avec la base de données :

- Entre **GamePlay** et **Participation** : Une relation OneToMany est utilisée. Cela signifie qu'une partie de quizz peut avoir plusieurs participations, mais chaque participation est liée à une seule partie de quizz.

- Entre **GamePlay** et **Question** : La relation ManyToMany indique que chaque partie de quizz peut contenir plusieurs questions et chaque question peut apparaître dans plusieurs parties de quizz.

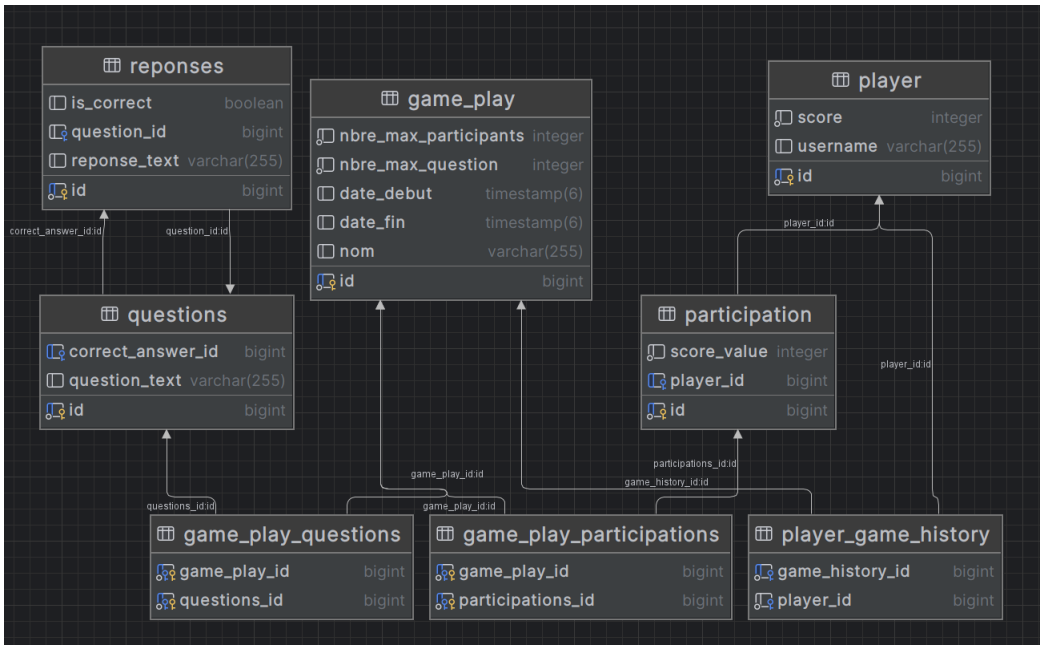
- La classe **Participation** a une relation OneToOne avec **Player**, indiquant qu'une participation est associée à un seul joueur.

- La classe **Question** a une relation OneToMany avec **Answer**, car une question peut avoir plusieurs réponses, mais chaque réponse est liée à une seule question.

Ce qui donne la base de données suivante :

4.0.2 La couche data

```
package com.example.TPQUIZZ.infrastructure;

import com.example.TPQUIZZ.domaine.GamePlay;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.repository.CrudRepository;
import org.springframework.stereotype.Repository;
```




```
@Repository
public interface GamePlayRepository extends JpaRepository<GamePlay, Long>
     {
}
```

Il n'y a aucune méthode définie dans cette interface mais du fait qu'elle hérite de CrudRepository, elle bénéficie des méthodes de la classe mère dont les principales sont :

- T save(T entity) : Renvoie un objet d'un certain type noté ici *T*, qui est défini par le premier argument dans l'instruction *extends CrudRepository<GamePlay,Long>*. Ici, *T* représente la classe GamePlay. Elle prend en paramètre un objet de type *T*, donc un objet GamePlay, qui correspond à la partie de quizz que l'on souhaite sauvegarder.

- Iterable<T> findAll() : Cette méthode ne prend aucun paramètre en entrée et renvoie une liste de *T*, qui représente ici des objets GamePlay. Elle renvoie donc une liste de parties de quizz sous forme d'Iterable.

- Optional<T> findById(ID id) : Cette méthode renvoie un Optional<T>, soit un Optional<GamePlay>, qui est un objet pouvant contenir une partie de quizz ou être vide. Elle prend en paramètre un objet de type *ID*, donné par le second argument de l'instruction *extends CrudRepository<GamePlay,Long>*, ici *ID* représente un Long. La méthode pourrait s'écrire :

```
Optional<GamePlay> findById(Long id)
```

- void delete(T entity) : Cette méthode ne renvoie rien et permet de supprimer une partie de quizz, donnée en paramètre.

- void deleteById(ID id) : Cette méthode est similaire à la précédente mais prend en paramètre l'identifiant de la partie de quizz que l'on souhaite supprimer.

Nous aurions pu étendre la classe *JpaRepository* pour inclure des fonctionnalités supplémentaires, telles que la vérification de l'existence d'entités par identifiant, le décompte du nombre d'entités et la suppression de toutes les entités. JpaRepository est une interface qui étend CrudRepository et offre une implémentation par défaut pour les opérations de persistance de JPA. Cette classe est spécifique aux bases de données relationnelles, alors que *CrudRepository* est adaptée à tout type de base de données mais reste plus générique.