

Diagramme de paquetage : mdr

Diagramme de classe : Done

Explications :

Personnage : Contient les personnages du jeu (9 personnages au total).

QuestionAnswer : Contient les questions/réponses du jeu (50 questions pour la démo).

Theme : Contient les différents thèmes du jeu (Programmation, Réseau, Langues, ...). Ne contient qu'un seul thème pour la démo.

Player : Non utilisée. Devait servir à stocker les informations des joueurs pour une utilisation de l'application sur plusieurs téléphones.

LevelStage : Non utilisée. Elle devait servir à déterminer le stage et donner les informations à son sujet.

BuildingType : Non utilisée. Elle devait servir à déterminer les différents bâtiments mais ceux-ci ne sont finalement pas traitables via l'application.

Diagramme de Séquence : A faire

Description écrite architecture :

- . **draw** : le dossier contenant les pages nécessaires au fonctionnement des cartes. Il s'appelle ainsi pour évoquer la « pioche ».
- . **entities** : contient les pages utilisées dans l'interface d'administrateur, qui permettent de créer, mettre à jour, supprimer ou visualiser les différentes classes et leurs données.
- . **fight** : contient les pages de sélection des adversaires ainsi que les pages d'animation du combat.
- . **game** : la page d'accueil regroupant les personnages et le rôle actuel (Attaquant ou Défenseur)
- . **home** : la page permettant de lancer le jeu, contenant le bouton « Jouer » redirigeant sur « game »
- . **layouts** : dossier contenant les layouts. Nous avons créé un layout nommé « SideMenu » qui correspond au menu sur le côté, présent sur toutes les pages et permet tant de changer de rôle, de changer de page, et d'incrémenter les tours.
- . **team** : contient les pages d'affichage et modification des personnages. Affiche la liste des personnages en fonction du rôle actuel.
- . **content** : contient les images et les fichiers css de l'application.

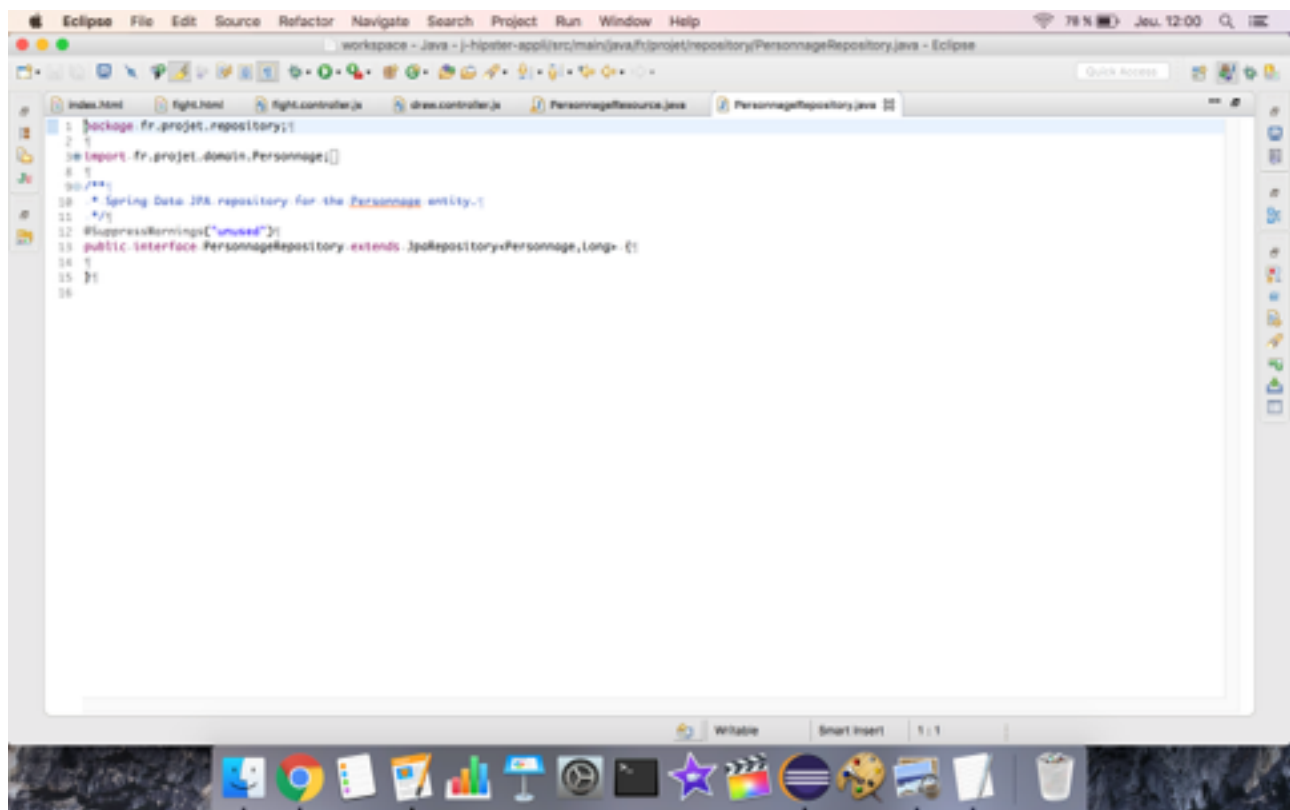
bases (classes, structures, instances, ...):
Instance de la classe SpringApplication à la ligne 62.

```
index.html  fight.html  fight.controller.js  draw.controller.js  PersonnageResource.java  JHipsterApp.java
31 @Object
32 private Environment env;
33
34 /**
35  * Initializes JHipsterApp.
36  *
37  * Spring profiles can be configured with a program arguments --spring.profiles.active=your-active-profile
38  *
39  * You can find more information on how profiles work with JHipster on http://jhipster.github.io/profiles/
40  */
41 @PostConstruct
42 public void initApplication() {
43     log.info("Running with Spring profile(s) : {}", Arrays.toString(env.getActiveProfiles()));
44     Collection<String> activeProfiles = Arrays.asList(env.getActiveProfiles());
45     if (activeProfiles.contains(Constants.SPRING_PROFILE_DEVELOPMENT) && activeProfiles.contains(Constants.SPRING_PROFILE_PRODUCTION)) {
46         log.error("You have misconfigured your application! It should not run with both the 'dev' and 'prod' profiles at the same time.");
47     }
48     if (activeProfiles.contains(Constants.SPRING_PROFILE_DEVELOPMENT) && activeProfiles.contains(Constants.SPRING_PROFILE_CLOUD)) {
49         log.error("You have misconfigured your application! It should not run with both the 'dev' and 'cloud' profiles at the same time.");
50     }
51 }
52
53 /**
54  * Main method, used to run the application.
55  */
56 public static void main(String[] args) throws UnknownHostException {
57     // on args the command line arguments
58     // throws UnknownHostException if the local host name could not be resolved into an address
59
60     SpringApplication app = new SpringApplication(JHipsterApp.class);
61     DefaultProperties.addDefaultProfile(app);
62     Environment env = app.run(args).getEnvironment();
63     log.info("Application '{}' is running! Access URLs:\n\t" +
64         "Local: \t%s://localhost:{}", env.getProperty("spring.application.name"),
65         env.getProperty("server.port"),
66         InetAddress.getLocalHost().getHostAddress(),
67         env.getProperty("server.port"));
68 }
69 }
```

La classe PersonnageResource contenant les méthodes permettant de récupérer, insérer, mettre à jour et supprimer des personnages

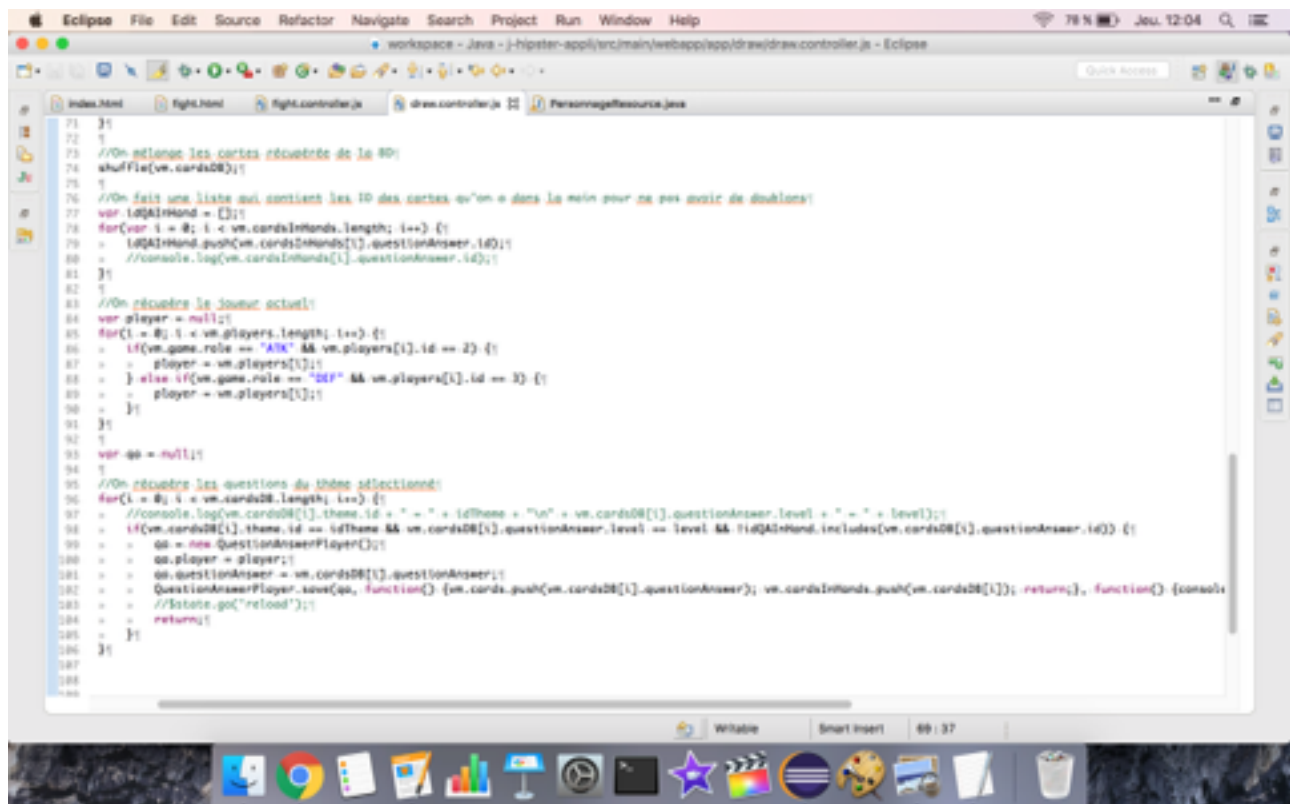
```
index.html  fight.html  fight.controller.js  draw.controller.js  PersonnageResource.java  JHipsterApp.java
1 package fr.projet.web.rest;
2
3 import com.codahale.metrics.annotation.Timed;
4
5 /**
6  * REST controller for managing Personnage.
7  */
8 @RestController
9 @RequestMapping("/api")
10 public class PersonnageResource {
11
12     private final Logger log = LoggerFactory.getLogger(PersonnageResource.class);
13
14     private PersonnageRepository personnageRepository;
15
16     /**
17      * POST /personnages : Create a new personnage.
18      *
19      * @param personnage the personnage to create
20      * @return the ResponseEntity with status 201 (Created) and with body the new personnage, or with status 400 (Bad Request) if the personnage has already an ID
21      * @throws URISyntaxException if the Location URI syntax is incorrect
22      */
23     @PostMapping("/personnages")
24     @Timed
25     public ResponseEntity<Personnage> createPersonnage(@RequestBody Personnage personnage) throws URISyntaxException {
26         log.debug("REST request to save Personnage : {}", personnage);
27         if (personnage.getId() != null) {
28             return ResponseEntity.badRequest().headers(HeaderUtil.createFailureAlert("personnage", "idexists", "A new personnage cannot already have an ID")).body(null);
29         }
30         Personnage result = personnageRepository.save(personnage);
31         HttpHeaders headers = HeaderUtil.createEntityCreationAlert("personnage", result.getId().toString());
32         return ResponseEntity.ok().headers(headers).body(result);
33     }
34
35     /**
36      * PUT /personnages : Updates an existing personnage.
37      *
38      * @param personnage the personnage to update
39      * @return the ResponseEntity with status 200 (OK) and with body the updated personnage,
40      * or with status 400 (Bad Request) if the personnage is not valid,
41      * or with status 500 (Internal Server Error) if the personnage couldn't be updated
42      * @throws URISyntaxException if the Location URI syntax is incorrect
43      */
44     @PutMapping("/personnages")
45     public ResponseEntity<Personnage> updatePersonnage(@RequestBody Personnage personnage) throws URISyntaxException {
46         log.debug("REST request to update Personnage : {}", personnage);
47         if (personnage.getId() == null) {
48             return ResponseEntity.badRequest().headers(HeaderUtil.createFailureAlert("personnage", "idrequired", "A personnage must have an ID")).body(null);
49         }
50         Personnage result = personnageRepository.save(personnage);
51         HttpHeaders headers = HeaderUtil.createEntityUpdateAlert("personnage", result.getId().toString());
52         return ResponseEntity.ok().headers(headers).body(result);
53     }
54 }
```

abstraction (héritage, interfaces, polymorphisme) :
Un exemple d'interface qui hérite de JpaRepository



```
1 package fr.projet.repository;
2
3 import fr.projet.domain.Personnage;
4
5 /**
6  * Spring Data JPA repository for the Personnage entity.
7  */
8 @SuppressWarnings("unused")
9 public interface PersonnageRepository extends JpaRepository<Personnage, Long> {}
10
11
```

collections simples (listes, tableaux, ...):
Un exemple d'utilisation de tableaux sous javascript.

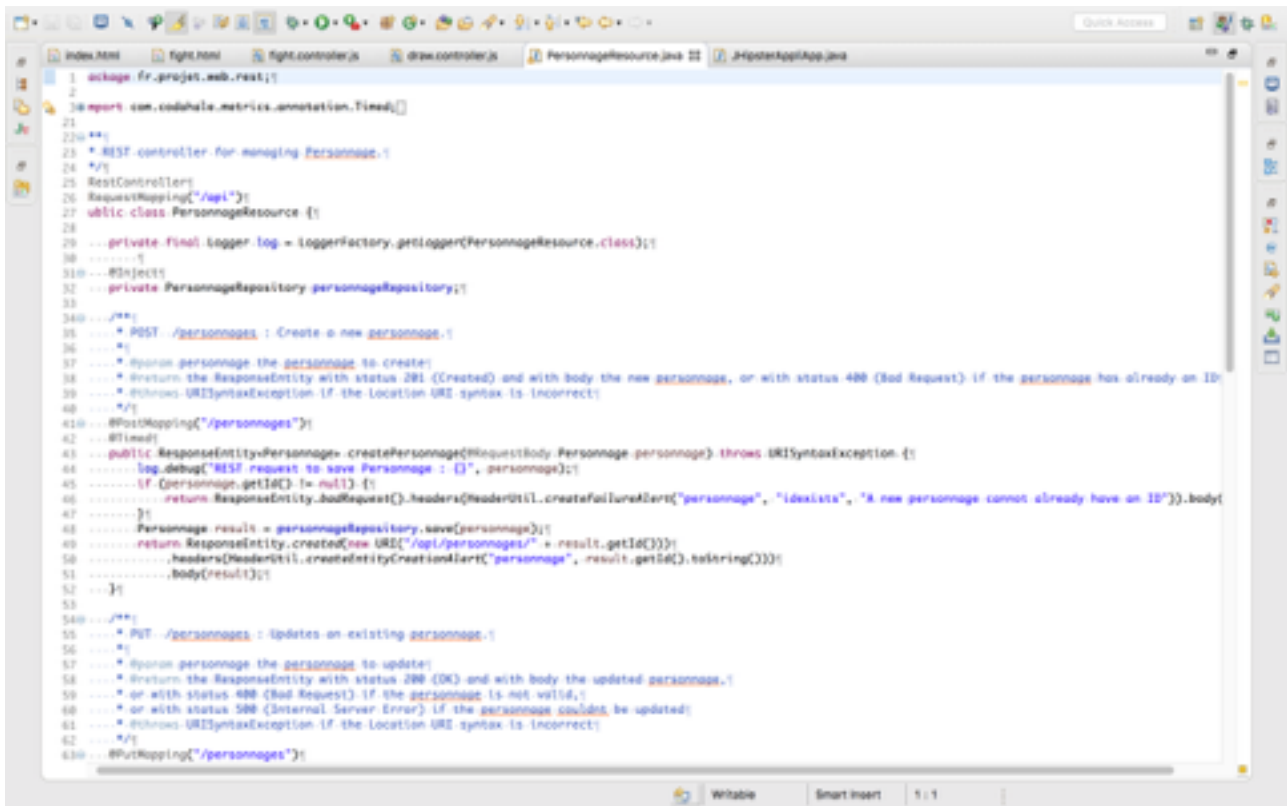


```
71 }
72
73 //On mélange les cartes récupérées de la BD
74 shuffle(vm.cardsDB);
75
76 //On fait une liste qui contient les ID des cartes qu'on a dans la main pour ne pas avoir de doublons
77 var idQAInHand = [];
78 for(var i = 0; i < vm.cardsInHand.length; i++) {
79   idQAInHand.push(vm.cardsInHand[i].questionAnswer.id);
80   //console.log(vm.cardsInHand[i].questionAnswer.id);
81 }
82
83 //On récupère le joueur actuel
84 var player = null;
85 for(i = 0; i < vm.players.length; i++) {
86   if(vm.game.role == "AUX" && vm.players[i].id == 2) {
87     player = vm.players[i];
88   } else if(vm.game.role == "DEP" && vm.players[i].id == 3) {
89     player = vm.players[i];
90   }
91 }
92
93 var qa = null;
94
95 //On récupère les questions du thème sélectionné
96 for(i = 0; i < vm.cardsDB.length; i++) {
97   //console.log(vm.cardsDB[i].theme.id + " - " + idTheme + " - " + vm.cardsDB[i].questionAnswer.level + " - " + level);
98   if(vm.cardsDB[i].theme.id == idTheme && vm.cardsDB[i].questionAnswer.level == level && !idQAInHand.includes(vm.cardsDB[i].questionAnswer.id)) {
99     qa = new QuestionAnswerPlayer();
100     qa.player = player;
101     qa.questionAnswer = vm.cardsDB[i].questionAnswer;
102     QuestionAnswerPlayer.save(qa, function() { vm.cards.push(vm.cardsDB[i].questionAnswer); vm.cardsInHand.push(vm.cardsDB[i]); return; }, function() { console
103     //console.log("reloaded");
104     return;
105   });
106 }
107
108
```

collections avancées (dictionnaire) :
pas compris

encapsulation :

Plusieurs encapsulations dans cette capture d'écran : attributs privés et méthodes publiques



```
1 package fr.projet.web.rest;
2
3 import com.codahale.metrics.annotation.Timed;
4
5 /**
6  * REST controller for managing Personnage.
7  */
8 @RestController
9 @RequestMapping("/api")
10 public class PersonnageResource {
11     ...private final Logger log = LoggerFactory.getLogger(PersonnageResource.class);
12     ...
13     ...@Inject
14     ...private PersonnageRepository personageRepository;
15     ...
16     /**
17      * POST /personages : Create a new personage.
18      *
19      * @param personage the personage to create
20      * @return the ResponseEntity with status 201 (Created) and with body the new personage, or with status 400 (Bad Request) if the personage has already an ID
21      * @throws URISyntaxException if the Location URI syntax is incorrect
22      */
23     @PostMapping("/personages")
24     @Timed
25     public ResponseEntity<Personnage> createPersonage(@RequestBody Personnage personage) throws URISyntaxException {
26         log.debug("REST request to save Personnage : {}", personage);
27         ...if (personage.getId() != null) {
28             ...return ResponseEntity.badRequest().headers(HeaderUtil.createFailureAlert("personage", "idexists", "A new personage cannot already have an ID")).body(
29                 ...
30             );
31         }
32         Personnage result = personageRepository.save(personage);
33         return ResponseEntity.created(new URI("/api/personages/" + result.getId()))
34             .headers(HeaderUtil.createEntityCreationAlert("personage", result.getId().toString()))
35             .body(result);
36     }
37     /**
38      * PUT /personages : Updates an existing personage.
39      *
40      * @param personage the personage to update
41      * @return the ResponseEntity with status 200 (OK) and with body the updated personage,
42      * or with status 400 (Bad Request) if the personage is not valid,
43      * or with status 500 (Internal Server Error) if the personage couldn't be updated
44      * @throws URISyntaxException if the Location URI syntax is incorrect
45      */
46     @PutMapping("/personages")
47     ...
48 }
```

tests fonctionnels :

L'utilisateur attaquant n'a pas de cartes dans sa main.

Il clique sur le paquet de cartes.

Une carte apparaît dans sa main.

Il clique sur la carte.

Elle s'affiche en plus gros.

Il clique sur « Utilisée ».

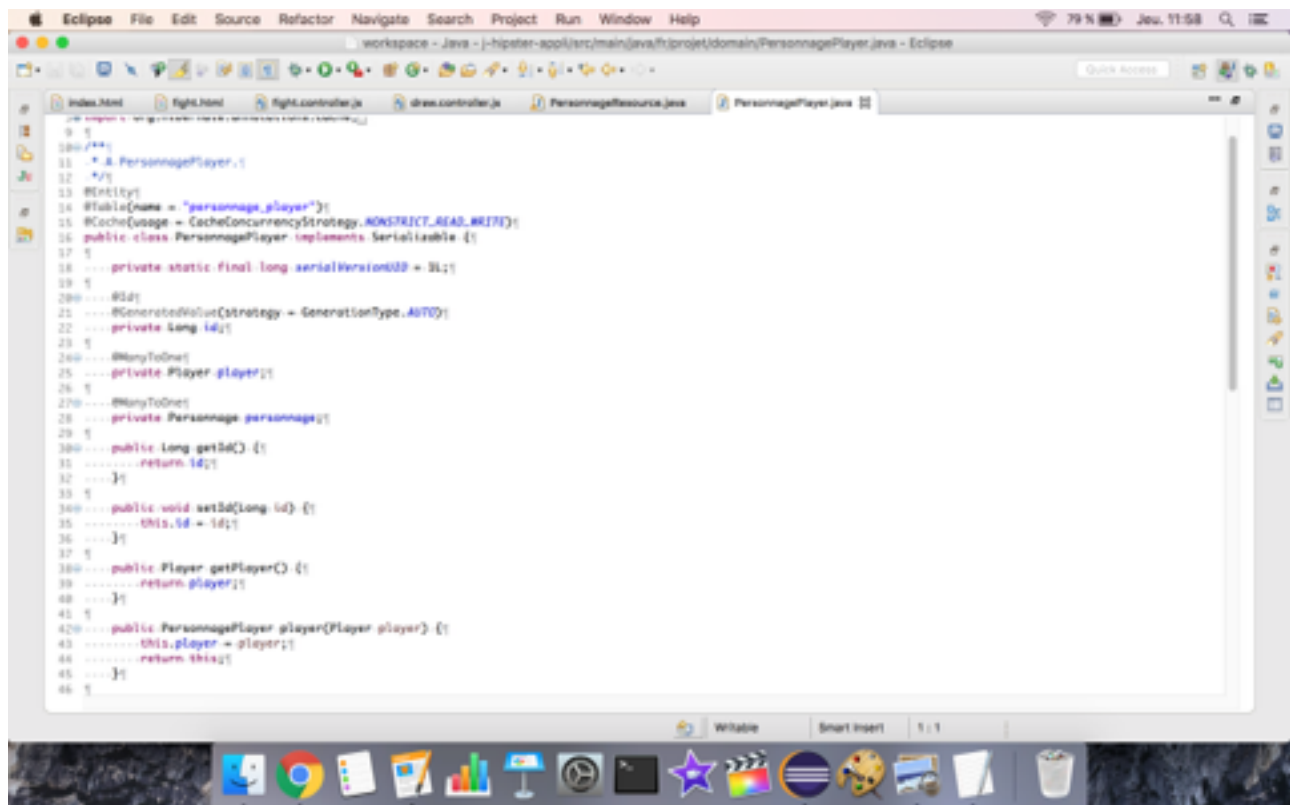
Il n'a plus la carte dans sa main.

(Prendre des screenshots)

tests unitaires :

JPA :

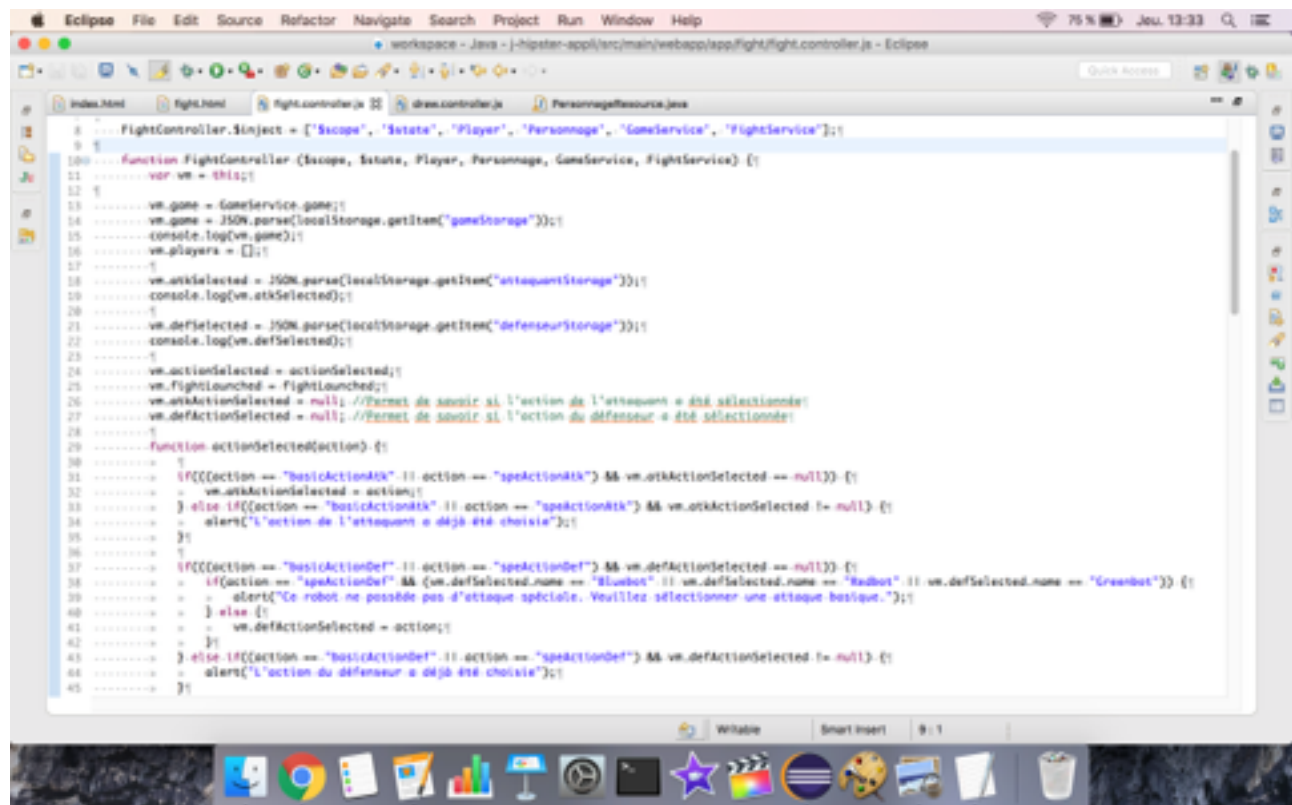
Utilisation de JPA pour la table Personnage



Spring :



Angular :



```
1 FightController.$inject = ['Scope', '$state', 'Player', 'Personnage', 'GameService', 'FightService'];
2
3 function FightController (Scope, $state, Player, Personnage, GameService, FightService) {
4     var vm = this;
5
6     vm.game = GameService.game;
7     vm.game = JSON.parse(localStorage.getItem("gameStorage"));
8     console.log(vm.game);
9     vm.players = [];
10
11     vm.attSelected = JSON.parse(localStorage.getItem("attaquantStorage"));
12     console.log(vm.attSelected);
13
14     vm.defSelected = JSON.parse(localStorage.getItem("defenseurStorage"));
15     console.log(vm.defSelected);
16
17     vm.actionSelected = actionSelected;
18     vm.fightLaunched = fightLaunched;
19     vm.attActionSelected = null; //Permet de savoir si l'action de l'attaquant a été sélectionnée
20     vm.defActionSelected = null; //Permet de savoir si l'action du défenseur a été sélectionnée
21
22     function actionSelected(action) {
23
24         if((action == "basicActionAtk" || action == "speActionAtk") && vm.attActionSelected == null) {
25             vm.attActionSelected = action;
26         } else if((action == "basicActionDef" || action == "speActionDef") && vm.defActionSelected != null) {
27             alert("l'action de l'attaquant a déjà été choisie");
28         }
29
30         if((action == "basicActionDef" || action == "speActionDef") && vm.defActionSelected == null) {
31             if(action == "speActionDef" && (vm.defSelected.name == "Bluebot" || vm.defSelected.name == "Redbot" || vm.defSelected.name == "Greenbot")) {
32                 alert("Ce robot ne possède pas d'attaque spéciale. Veuillez sélectionner une attaque basique.");
33             } else {
34                 vm.defActionSelected = action;
35             }
36         } else if((action == "basicActionDef" || action == "speActionDef") && vm.defActionSelected != null) {
37             alert("l'action du défenseur a déjà été choisie");
38         }
39     }
40 }
```