|  |
| --- |
| Projet AL : Application client-serveur de type 3-tiers  https://archives.plil.fr/msauvage/projetAL\_LaplaceSauvage.git |



# **Introduction**

Dans le cadre du module Architecture Logicielle nous sommes amenés à réaliser une base de données et une application JEE qui exploite cette base en réseau. L’application doit permettre la gestion de spectacles au sein d’un théâtre et l’achat de billets de manière très simplifiée.

Dans ce rapport nous proposons en première partie une description détaillée des composants EJB, ensuite on retrouve en seconde partie l’explication de l’architecture logicielle retenue accompagnée d’un schéma. Enfin en dernière partie nous présentons nos différentes données de tests avec notamment le scénario utilisé, comment utiliser notre application cliente et une description de notre application cliente.

# 

# **Table des matières**

[**Introduction**](#_xu13du7gcfb4) **1**

[**Table des matières**](#_ze3ebrrtkvpb) **2**

[**1. Les composants EJB entités**](#_432so4me8rt1) **3**

[1.1. Schéma UML-EJB des entités](#_8ytvy5v5ata) 3

[1.2. Traduction Java des attributs et des identifiants](#_vqueie2zhfda) 4

[1.2.1. Classe Place](#_4drnbkc7r8o0) 4

[1.2.2. Classe CategoriePlace](#_es6dm6bi89j) 4

[1.2.3. Classe Spectacle](#_x3c2216db0e1) 5

[1.2.4. Classe Billet](#_50a2ln23gjwx) 6

[1.2.5. Classe Client](#_57lkgg4qxv17) 7

[**2. Architecture logicielle**](#_xpsad863dw6v) **8**

[2.1. Interaction entre les composants](#_urop5py85g2s) 8

[2.2. Schéma de l’architecture logicielle retenue](#_hoi3v7jnfwfj) 9

[**3. Données de test**](#_7twdow76mnbo) **10**

[3.1. Description du scénario](#_tjte35n4pm7y) 10

[3.1.1. Partie gestion du théâtre](#_ey2bvn3na6fa) 10

[3.1.2. Partie gestion des billets](#_4z9f9tpnph2o) 10

[3.2. Description de l’Application cliente développée](#_n10j4wy9w4wm) 11

[**4. Application cliente Web**](#_yyp1tkdeozdt) **12**

[4.1. Utilisation de l’Application cliente Web](#_l54acnqqr5iq) 12

[4.2. Description de l’application Web](#_sfv81fdcm572) 14

[4.2.1. index](#_zcg802t3tsmz) 14

[4.2.2. reserver](#_6ms3yfckhtop) 14

[4.2.3. resultatReservation](#_6hwu0k6pzztv) 15

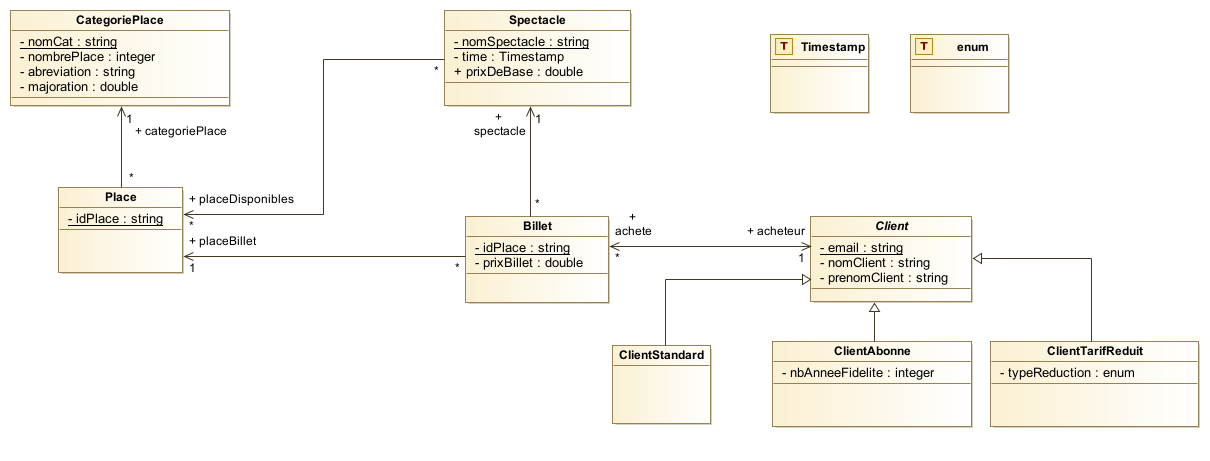
[**Conclusion**](#_8cs5ib5ecxss) **16**

# 

# **1. Les composants EJB entités**

## 1.1. Schéma UML-EJB des entités

L’analyse approfondie du cahier des charges nous a permis de dégager un schéma UML-EJB des entiés permettant de voir la correspondance entre les données et les objets.



*Figure 1-1 Modélisation UML-EJB des entités*

A partir de ce schéma, nous pouvons réaliser une traduction Java. D’abord, nous commençons par traduire les classes UML. Une classe UML devient une classe Java annotée par @entity. Cette classe doit posséder une fonction constructeur sans paramètre (pour introspection) et doit être sérializable. Nous devons traduire l’héritage UML via extends.

## 1.2. Traduction Java des attributs et des identifiants

### 1.2.1. Classe Place

|  |  |  |  |
| --- | --- | --- | --- |
| Attribut | Type | Annotation | Explication |
| idPlace | int | @Id | Cet attribut désigne la clé primaire |
| categoriePlace | CategoriePlace | @ManyToOne | Ajout de cet annotation pour caractériser l’association vers la classe CategoriePlace de cardinalité n:1 |

### 1.2.2. Classe CategoriePlace

|  |  |  |  |
| --- | --- | --- | --- |
| Attribut | Type | Annotation | Explication |
| nomCat | String | @Id | Cet attribut désigne la clé primaire |
| nombrePlace | int | rien |  |
| abreviation | String | rien |  |
| majoration | double | rien |  |

### 

### 1.2.3. Classe Spectacle

|  |  |  |  |
| --- | --- | --- | --- |
| Attribut | Type | Annotation | Explication |
| nomSpectacle | String | @Id | Cet attribut désigne la clé primaire |
| time | Timestamp | rien |  |
| prixDeBase | double | rien |  |
| placeDisponibles | Set<Place> | @ManyToMany (fetch=FetchType.EAGER) | Ajout de cet annotation pour caractériser l’association vers la classe Place de cardinalité n:p  Nous ajoutons attribuons la valeur de fetch à FetchType.EAGER pour que la relation soit chargée en même temps que l'entité qui la porte. |

### 

### 1.2.4. Classe Billet

|  |  |  |  |
| --- | --- | --- | --- |
| Attribut | Type | Annotation | Explication |
| idBillet | int | @Id  @GeneratedValue | Cet attribut désigne la clé primaire  Nous ajoutons l’annotation GeneratedValue afin de créer une séquence qui va générer automatiquement les valeurs entières de clés. |
| prixBillet | double | rien |  |
| placeBillet | Place | @ManyToOne | Ajout de cet annotation pour caractériser l’association vers la classe Place de cardinalité n:1 |
| spectacle | Spectacle | @ManyToOne | Ajout de cet annotation pour caractériser l’association vers la classe Spectacle de cardinalité n:1 |
| acheteur | Client | @ManyToOne | Ajout de cet annotation pour caractériser l’association vers la classe Client de cardinalité n:1 |

### 

### 1.2.5. Classe Client

|  |  |  |  |
| --- | --- | --- | --- |
| Attribut | Type | Annotation | Explication |
| email | String | @Id | Cet attribut désigne la clé primaire |
| nomClient | String | rien |  |
| prenomClient | String | rien |  |
| achete | Collection<Billet> | @OneToMany  (mappedBy =”acheteur”)  fetch=FetchType.EAGER) | Ajout de cet annotation pour caractériser l’association bidirectionnelle avec la classe Place de cardinalité 1:n  Nous ajoutons attribuons la valeur de fetch à FetchType.EAGER pour que la relation soit chargée en même temps que l'entité qui la porte.  L'attribut mappedBy est défini sur l'entité esclave de la relation. |

Trois classes héritent de la classe Client : ClientAbonne, ClientTarifReduit, ClientStandard. Nous avons adopté une stratégie d’héritage qui permet d’avoir une table par classe avec l’annotation @Inheritance (strategy=InheritanceType.TABLE\_PER\_CLASS).

## 

# 

# **2. Architecture logicielle**

## 2.1. Interaction entre les composants

Les EJB sessions contiennent les EJB entités afin de pouvoir les manipuler et constitue le conteneur EJB cela correspond au Business Tiers. Les EJB sessions communiquent avec les pages JSP qui constituent le Web Tiers par l’intermédiaire du contrôleur (nommé controlleur dans notre archive suite un soucis d’éditeur Java) avec à un accès local qui est possible grâce au serveur JEE Wildfly.

Ainsi sur la machine client on peut observer les vues qui sont gérées par le contrôleur soumises au protocole HTTP.

Les données sont stockées dans une base de données Postgres SQL disponible sur un serveur SGBD Postrgres.

Nous pouvons retrouver un schéma de l’architecture logicielle qui illustre cette interaction dans la partie suivante.

## 2.2. Schéma de l’architecture logicielle retenue

# 

*Figure 2-1 Schéma de l’architecture logicielle retenue*

# 

# **3. Données de test**

Nous avons developpé une application cliente Main.java qui teste toutes les fonctionnalités des composants sessions développés, nous avons écrit en dur un scénario qui permet de tester les composants EJB sans demander de à l’utilisateur de faire une saisie au clavier. Ce scénario correspond à celui qui est proposé par le sujet, nous avons mis en place des messages afin que le compte-rendu d’exécution soit le plus lisible possible.

## 3.1. Description du scénario

### 3.1.1. Partie gestion du théâtre

Configuration du théâtre :

* Catégorie ‘Balcon Droit’ (abbrev : ‘BD’, majoration : 20.0, nb : 30)
* Catégorie ‘Balcon Gauche’ (abbrev : ‘BG’, majoration : 20.0, nb : 30)
* Catégorie ‘Balcon Centre’ (abbrev : ‘BC’, majoration : 25.0, nb : 10)
* Catégorie ‘Orchestre’ (abbrev : ‘O’, majoration : 30.0, nb : 50)

Configuration des spectacle :

* Spectacle ‘Springsteen Tour 2019’ (le 2019-03-27 a 19:00:00), prixBase : 45.0  
  30 places en Balcon Droit, 30 places en Balcon Gauche, 40 places Orchestre
* Spectacle ‘Les fourberies de Scapin’ (le 2019-03-28 a 20:30:00), prixBase : 45.0  
  10 places en Balcon Centre, 50 places Orchestre

Création des clients :

* Client abonné : [dupont@gmail.com](mailto:dupont@gmail.com) (Jean Dupont, 3 ans de fidélité)
* Client tarif réduit : [dubois@gmail.com](mailto:dubois@gmail.com) (Paul Dubois, étudiant)
* Client standard : [durant@gmail.com](mailto:durant@gmail.com) (Marc Durant)

### 3.1.2. Partie gestion des billets

Affichage des catégories de places ouvertes en Orchestre pour Les fourberies de Scapin

Affichage du nombre de places disponibles pour en Orchestre pour Les fourberies de Scapin

Achat de places :

* client : dupont@gmail.com , achat 2 places en orchestre pour Les fourberies de Scapin
* client : dubois@gmail.com , achat 3 places en orchestre pour Les fourberies de Scapin

Affichage des billets obtenus de ces deux clients pour Les fourberies de Scapin.

Grâce à ce scénario on pourra observer si le calcul du prix des billets est correct en fonction du type de client.

## 3.2. Description de l’Application cliente développée

L’application cliente utilise les deux services GestionTheatre et GestionBillet. L’application commence par faire effectuer un lookup afin de pouvoir utiliser les méthodes de GestionTheatreBean par l’intermédiaire de l’objet GestionTheatre gt. A partir de ce dernier nous pouvons appeler la méthode creationCategoriePlace pour créer les catégories en utilisant les paramètres correspondant à ceux du scénario. De la même manière nous appelons la méthode creationSpectacle pour créer les spectacles du scénario. On configure ensuite les places ouvertes de chacun des spectacles avec la méthode ouverturePlace. Enfin on crée les clients à l’aide des méthodes associées aux types de clients à créer. On distingue alors trois méthodes : creationClientStandard pour créer le client standard, creationClientTarifRed pour créer le client tarif réduit et creationClientAbonne pour créer le client abonné.

En deuxième partie nous effectuons un deuxième lookup pour utiliser les méthodes de GestionBilletBean par l’intermédiaire de l’objet GestionBillet gb. Nous commençons par appeler la méthode listerCategorie avec ‘Les fourberies de Scapin’ en paramètre afin de récupérer les catégories de places ouvertes pour ce spectacle pouvoir afficher par la suite comme l’indique le scénario. Ensuite, nous appelons nbPlaceDispo avec les paramètres suivants : ‘Orchestre’ et ‘Les Fourberies de Scapin’ pour récupérer le nombre de places disponibles en Orchestre pour le spectacle Les fourberies de Scapin afin de pouvoir l’afficher. Enfin nous effectuons les achats de places grâce à la méthode achatDePlace avec les paramètres correspondants aux scénarios pour les clients [dubois@gmail.com](mailto:dubois@gmail.com) et [dupont@gmail.com](mailto:dupont@gmail.com) . Pour chacun des clients nous récupérons les billets achetés grâce à la méthode infoBillet pour le spectacle Les fourberies de Scapin puis nous les affichons. Nous avons les même prix que l'illustration du scénario présenté dans le sujet. Nous considérons alors les méthodes utilisées sont correct et nous avons pu alors passer au développement du module web.

# 

# 4. Application cliente Web

## 4.1. Utilisation de l’Application cliente Web

Au préalable le serveur Wildfly doit être actif.

Pour utiliser notre application cliente il faut d’abord modifier le fichier build.xml afin de cibler le bon répertoire dans lequel se trouve le serveur JEE WildFly. Ensuite l’utilisateur peut utiliser la commande Ant d’exécution DeployEntityAndSessionWithEAR il faut utiliser la commande run-client afin d’avoir une illustration similaire à celle disponible à celle du sujet.

Pour utiliser notre application cliente web il faut d’abord utiliser la commande And d’exécution DeployAll, il faut ensuite utiliser la commande run-client pour avoir les données du théâtre dans la base de données.

Pour accéder à la vue reserver comme la FIGURE 2 de l’affichage possible de la première vue du sujet il faut écrire l’url : <http://localhost:8080/billeterie/> afin d’arriver sur l’index de notre application permettant de choisir un client et un spectacle afin de pouvoir acheter des billets pour les places d’un spectacle entré en paramètre.

### 

*Figure 4-1 Vue ‘reserver’*

Ensuite, il faut choisir un nombre de billet à acheter pour une catégorie à l’aide d’une liste déroulante puis cliquer sur ‘Acheter’.



*Figure 4-2 Vue ‘reserver’*

Enfin, le résultat est donnée par la seconde vue ‘ResultatReservation’.



*Figure 4-3 Vue ‘resultatReservation’*

## 

## 4.2. Description de l’application Web

Dans le contrôleur Java nous gérons trois vues index, reserver et resultatReservation. Nous vérifions donc la fin de l’url afin de savoir quels sont les vues que nous allons utiliser pour faire l’affichage côté client.

### 4.2.1. index

Lorsque la fin de l’url vaut ‘index’ ou si nous écrivons juste <http://localhost:8080/billeterie/> la vue vaut ‘index.jsp’. Qui consiste en un formulaire qui demande à l’utilisateur de saisir un email client et un nom de spectacle. Ce formulaire renvoi vers la page ‘reserver’.

### 4.2.2. reserver

Lorsque la fin de l’url vaut ‘reserver’ suite à une saisie faites sur la page d’index de l’application cliente web la vue vaut ‘reserver.jsp’. Le contrôleur récupère donc les informations saisies grâce à un appel de la méthode request.getparameter() et les transmet sur la page ‘reserver.jsp’ grâce à un appel de la méthode request.setAttribute() qui définit les objets à utiliser sur la page JSP. A l’aide de la méthode *infoSpectacle(nomSpec)* nous récupérons l’objet Spectacle que nous pouvons utiliser par la suite afin d’avoir le nombre de places disponibles pour pouvoir élaborer les listes déroulantes du formulaire. Sur la page ‘reserver.jsp’ nous présentons les conditions tarifaires et nous proposons un formulaire d’achat de place où l’utilisateur peut choisir pour une catégorie ouverte, le nombre de place qu’il souhaite acheter.

### 

### 4.2.3. resultatReservation

Lorsque la fin de l’url vaut ‘resultatReservation’ suite à une saisie faites sur la page d’index de l’application cliente web et la vue vaut ‘resultatreservation.jsp’. Le contrôleur récupère donc les informations saisies grâce à un appel de la méthode request.getparameter() et les transmet sur la page ‘resultatReservation.jsp’ grâce à un appel de la méthode request.setAttribute() qui définit les objets à utiliser sur la page JSP. On vérifie que les places que le client souhaite acheter sont disponibles, si elles sont bien disponible nous envoyons sur la page JSP un chaîne de caractères qui indique que la commande est acceptée. Dans le cas contraire, nous envoyons une chaîne de caractères qui indique que la commande est refusée. Pour pouvoir obtenir les informations du billet du client nous utilisons la méthode infoBillets() pour les transmettre à la page JSP.

# 

# **Conclusion**

En guise de conclusion nous pouvons dire que notre application est entièrement fonctionnelle par rapport à ce qui était demandé dans le cahier des charges. En effet notre application permet d’obtenir un enchaînement des deux vues similaires aux illustrations du projet. De plus, l’exécution de l’application cliente Main.java permet de retrouver les mêmes données que la proposition de scénario présentée dans le sujet.