Institut Supérieur des Sciences Appliquées et de Technologie de Sousse

Conception et réalisation d'une application JEE

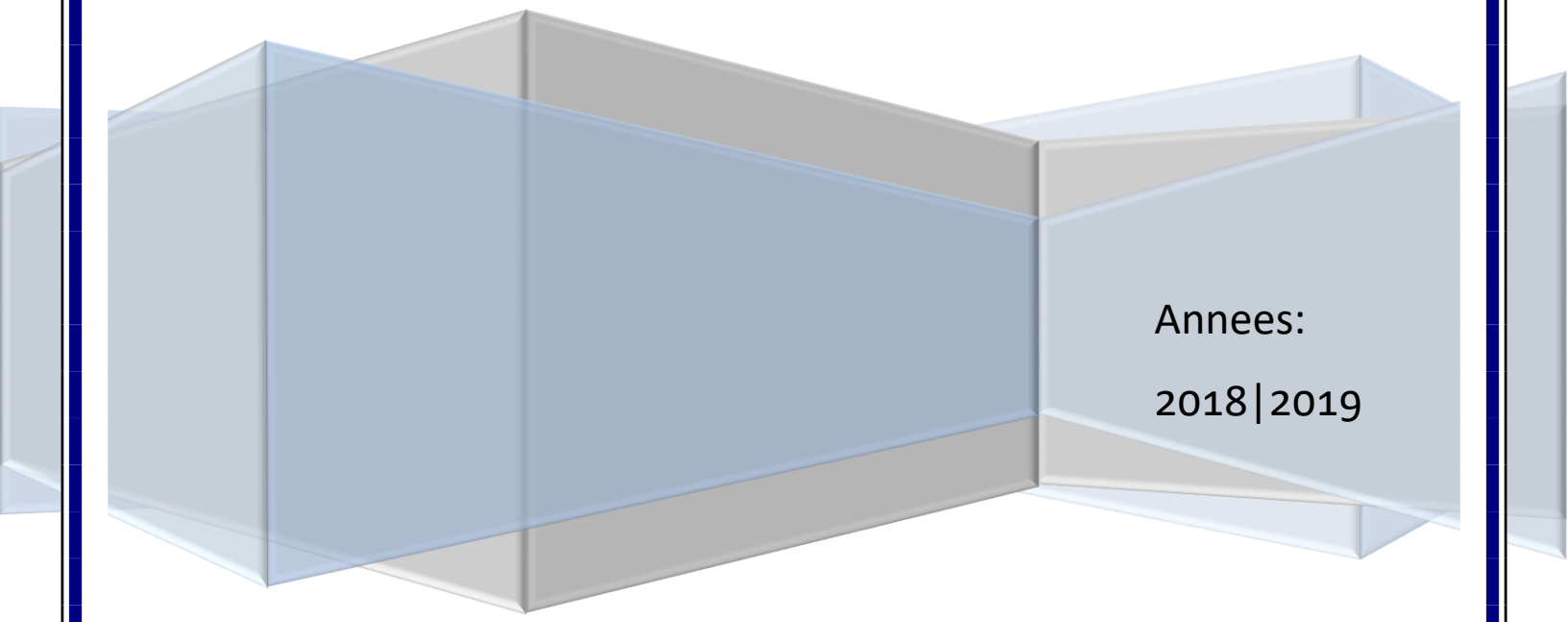
Livraison des logiciels informatiques

Réaliser par:

Ferhi Malek **Slimen Anis**

Mahdouani Wissal **Aguerbaoui Nadia**

Nasraoui Olfa



Annees:

2018|2019

Choix de la plateforme JEE :

Pour le développement de l'application, nous avons utilisé la plateforme JEE (Java Edition Entreprise).

Parmi les avantages de la plateforme JEE, on peut citer :

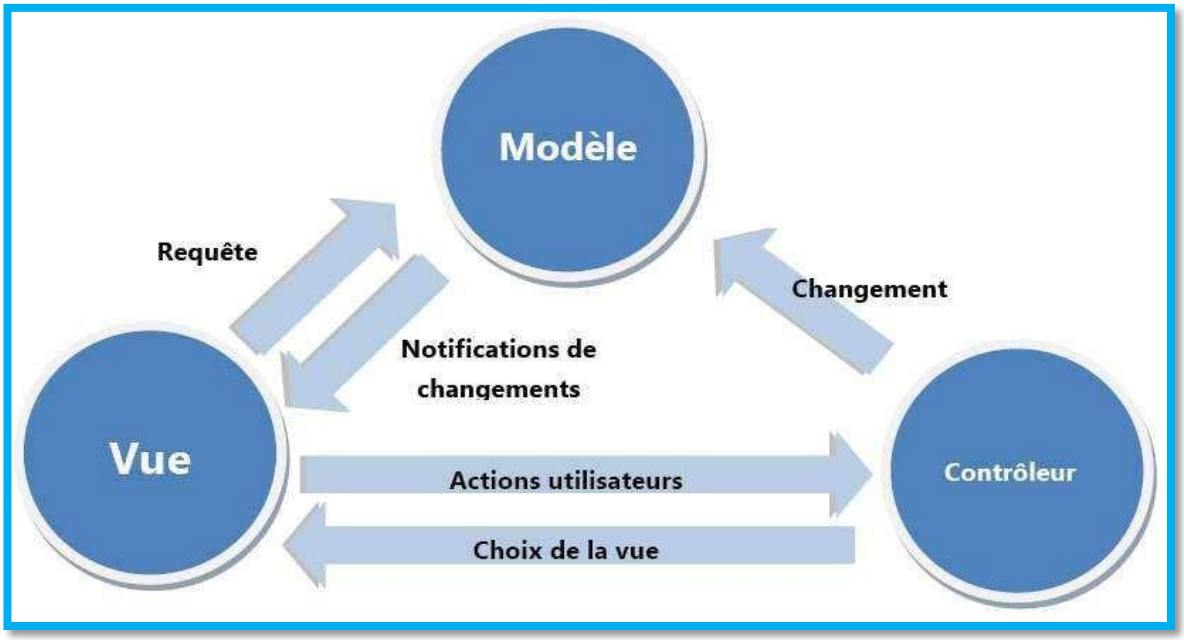
- Technologie sans frais : Java (sur le quel est basé JEE) est une technologie Open Source, les outils de développement JEE sont disponibles gratuitement.
- Maintenabilité : Les applications JEE sont plus faciles à entretenir, dans la plupart des cas, ils sont conçus en plusieurs couches. Il est facile d'ajouter de nouvelles fonctionnalités tierces pour les applications JEE en raison de sa fonction d'évolutivité.
- Indépendance : Les applications développées avec JEE peuvent être déployées sur la plupart des matériels disponibles. Elles offrent une flexibilité de matériel à l'utilisateur final. Ainsi, l'utilisateur peut déployer et exécuter des applications JEE sur le système d'exploitation et le matériel de son choix.

Choix de développement :

Pour le développement, nous avons appliqué le modèle Model-View-Controller (MVC). Ce paradigme divise l'IHM (Interface Homme Machine) en un modèle (M pour modèle de données) une vue (V pour la présentation, l'interface utilisateur) et un contrôleur (C pour la logique de contrôle, et la gestion des événements / synchronisation), chacun ayant un rôle précis dans l'interface.

L'organisation globale d'une interface graphique est souvent délicate. L'architecture MVC ne résout pas tous les problèmes. Elle fournit souvent une première approche qui peut ensuite être adaptée. Elle offre aussi un cadre pour structurer une application.

Ce patron d'architecture impose la séparation entre les données, la présentation et les traitements, ce qui donne trois parties fondamentales dans l'application finale : le modèle, la vue et le contrôleur.



- Le modèle : Le modèle représente le comportement de l'application : traitements des données, interactions avec la base de données, etc. Il décrit ou contient les données manipulées par l'application. Il assure la gestion de ces données et garantit leur intégrité. Dans le cas typique d'une base de données, c'est le modèle qui la contient. Le modèle offre des méthodes pour mettre à jour ces données (insertion, suppression, changement de valeur). Il offre aussi des méthodes pour récupérer ces données. Les résultats renvoyés par le modèle sont dénués de toute présentation.
- La vue : La vue correspond à l'interface avec laquelle l'utilisateur interagit. Sa première tâche est de présenter les résultats renvoyés par le modèle. Sa seconde tâche est de recevoir toutes les actions de l'utilisateur (clic de souris, sélection d'une entrée, boutons, etc.). Ces différents événements sont envoyés au contrôleur. La vue n'effectue aucun traitement, elle se contente d'afficher les résultats des traitements effectués par le modèle et d'interagir avec l'utilisateur.
- Le contrôleur : Le contrôleur prend en charge la gestion des événements de synchronisation pour mettre à jour la vue ou le modèle et les synchroniser. Il reçoit tous les événements de l'utilisateur et enclenche les actions à effectuer. Si une action nécessite un changement des données, le contrôleur demande la modification des données au modèle, et ce dernier notifie la vue que les données ont changée pour qu'elle les mette à jour.

Les technologies utilisées :

- Couche présentation : JSP

Les pages JSP sont une des technologies de la plate-forme Java EE les plus puissantes, simples à utiliser et à mettre en place. Elles se présentent sous la forme d'un simple fichier au format texte, contenant des balises respectant une syntaxe à part entière. Le langage JSP combine à la fois les technologies HTML, XML, servlet et JavaBeans en une seule solution permettant aux développeurs de créer des vues dynamiques.

- Couche métier : Les EJB

Un EJB (Enterprise JavaBean) est un composant logiciel de la plate-forme JEE de Sun, qui fournit un environnement Java pur pour l'élaboration et l'exécution d'applications distribuées.

Les EJB sont écrites comme des modules logiciels qui contiennent le logique métier de l'application. Il existe trois types des EJB :

Les sessions EJB : Une session est un objet non persistant. Sa durée de vie est la durée d'une interaction particulière entre le client et l'EJB. Le client crée normalement un

EJB, appelle des méthodes sur lui, et puis le supprime. Si le client ne parvient pas à le supprimer, le conteneur d'EJB l'enlève après une certaine période d'inactivité.

Les entités EJB : Les entités sont des objets persistants qui sont généralement synchronisés avec une base des données relationnelles dans une application orientée-objet.

Les message-driven EJB : Ces sont des composants métiers conçus pour les traitements asynchrones.

Les EJB sont exécutées dans un moteur d'exécution appelé un conteneur EJB, qui offre une multitude d'interfaces et de services communs à l'EJB, y compris la sécurité et le support transactionnel.

- Couche d'accès aux données : JPA

Le JPA (Java Persistence API) est l'API standard pour la gestion de la persistance et de mapping objet- relationnel des données. Le mapping objet-relationnel consiste à associer une ou plusieurs classes avec une table, et chaque attribut de la classe avec un champ de la table.

L’API JPA est basée sur les entités EJB. Chaque entité est le mapping d’une table relationnelle à une classe java, cette dernière permet d’encapsuler les données de la table.

Les interactions entre la base de données et les entités EJB sont assurées par un objet de type EntityManager qui permet de lire et rechercher des données mais aussi de les mettre à jour (ajout, modification, suppression). Le cycle de vie de l'EntityManager est géré par le conteneur EJB.

Puisque nous avons utilisé un langage de POO (Java) avec une base des données relationnelle, nous avons développé la couche d’accès aux données avec l’API JPA.

Modélisation :

- Diagramme de cas d’utilisation :

Le diagramme de cas d'utilisation décrit les utilisations requises d'un système, ou ce qu'un système est supposé faire. Les principaux concepts de ce diagramme sont les acteurs, cas d'utilisation et sujets. Un sujet représente un système avec lequel les acteurs et autres sujets interagissent.

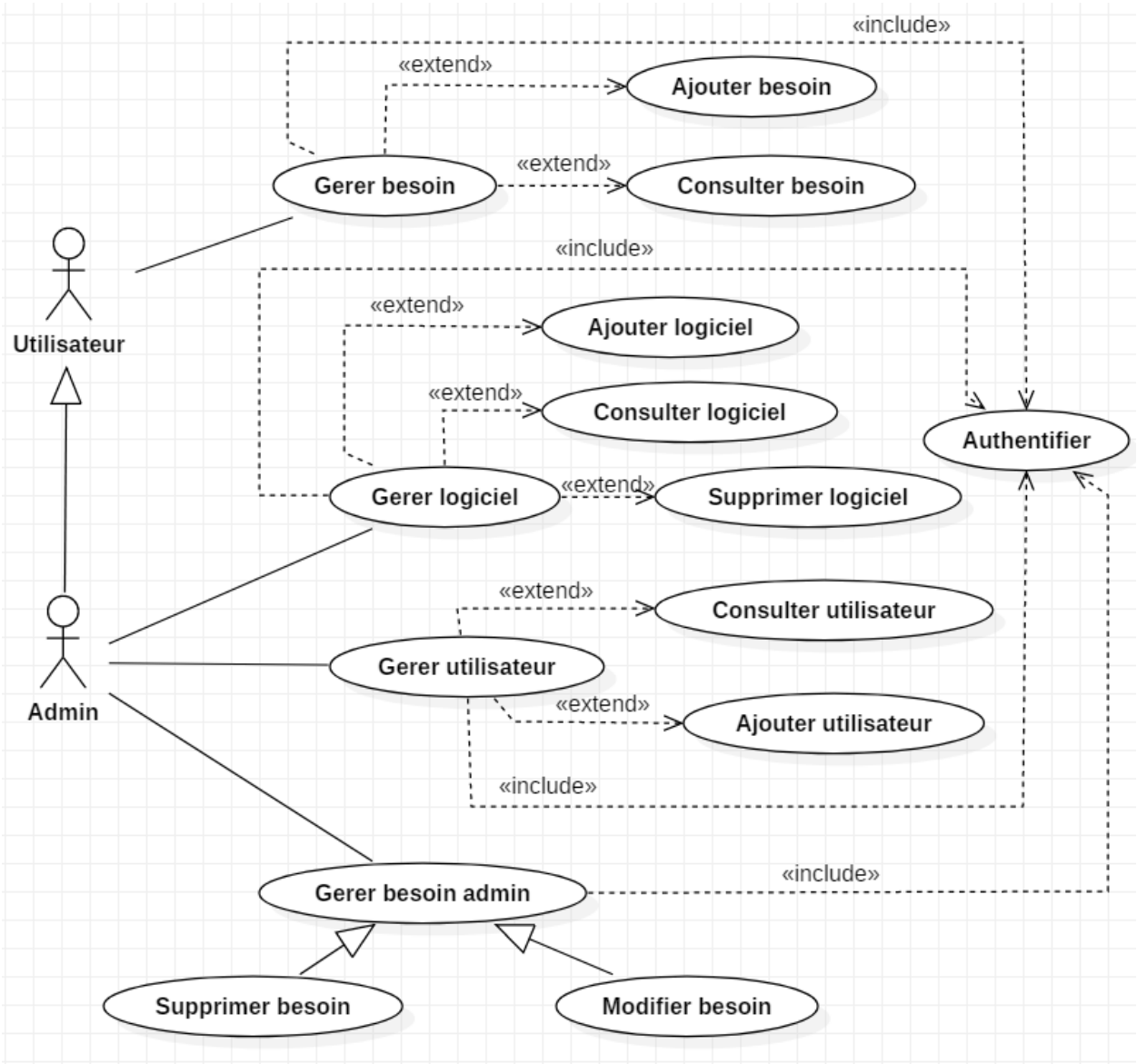


Diagramme cas d’utilisation générale

Cas d'utilisation: Afficher toute la liste des utilisateurs

But : Ce cas d'utilisation permet à l'admin de consulter la liste de tous les utilisateurs.

Les acteurs : Admin.

Description textuelle : Ce cas d'utilisateur donne le droit d'accées pour l'admin de consulter la liste des utilisateurs qui sont enregistrés dans la base de données.

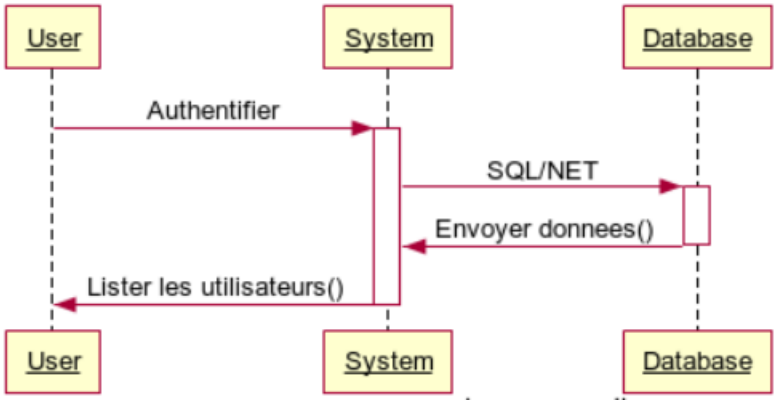
Scenario nominale :

1. L'utilisateur ouvre l'application
2. Le système affiche la page d'index et demande à l'utilisateur de s'authentifier
3. L'utilisateur saisie son e-mail et son mot de passe
4. Le système connecte à la base de données afin de vérifier les informations déjà saisie
5. L'utilisateur clique sur le lien « lister »
6. Le système recupére la liste des utilisateurs et l'affiche au utilisateur

Scénario alternatif :

3. L'oublie de mot de passe ou e-mail ➔redirection vers la page d'index

4. Les informations saisie ne sont pas correcte ➔ redirection vers la page d'index



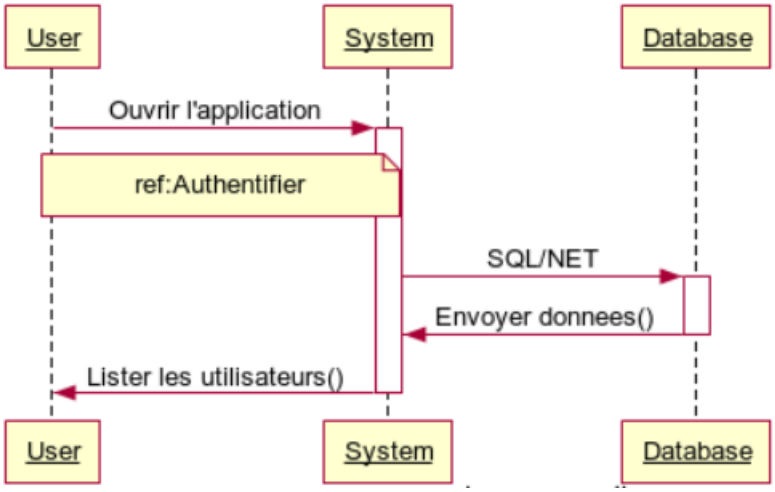


Diagramme de séquence : Afficher liste utilisateur

- Diagramme de sequence :

Un diagramme de séquence est un document graphique qui montre pour des scénarios de cas d'utilisation précis, les événements générés et les interactions entre objets en se basant sur des messages ordonnés. Chaque message transitant sur un lien est symbolisé par une flèche porteuse d'une expression. La lecture se fait de haut en bas, et l'ordre chronologique doit respecter ce sens.

La réalisation de diagramme de séquence permet de lister les méthodes dont on aura besoin lors de la phase de développement. Pour ce faire, la description doit être suffisamment générale et exhaustive pour identifier tous les algorithmes

Cas d'utilisation : Ajouter besoin

But : Ce cas d'utilisation permet à l'utilisateur d'ajouter un nouveau besoin

Les acteurs : Admin ou utilisateur final

Description textuelle : Ce cas d'utilisateur donne le droit d'accées pour l'utilisateur d'ajouter un nouveau besoin qui n'existe pas dans la base de données. Et puis l'admin Consulte la liste des besoins et peuvent accepter ou supprimer ce nouveau besoin.

Scenario nominale :

1. L'utilisateur clique sur le lien « créer besoin »
2. Le système affiche la page jsp « addbes.jsp » qui contient le formulaire a remplir
3. L'utilisateur choisie le categorie du nouveau besoin
4. L'utilisateur saisie la description propre a ce besoin et valide l'ajout

5. La page jsp envoit le données à la servlet «addbes.java »
6. La servlet recupére les données en utilisant(JNDI) la methode « Get_atr() »
7. La servlet cree la liste de type Besoin
8. La servlet appel l'EJB BesoinFascadeLocal
9. La servlet cree ce besoin a travers l'EJB et retourne un message de validation de l'ajout

Scenario alternatif :

9.L'echec de l'ajout du besoin ➔ un message d'erreur s'affiche et rtourne vers la deuxieme etape

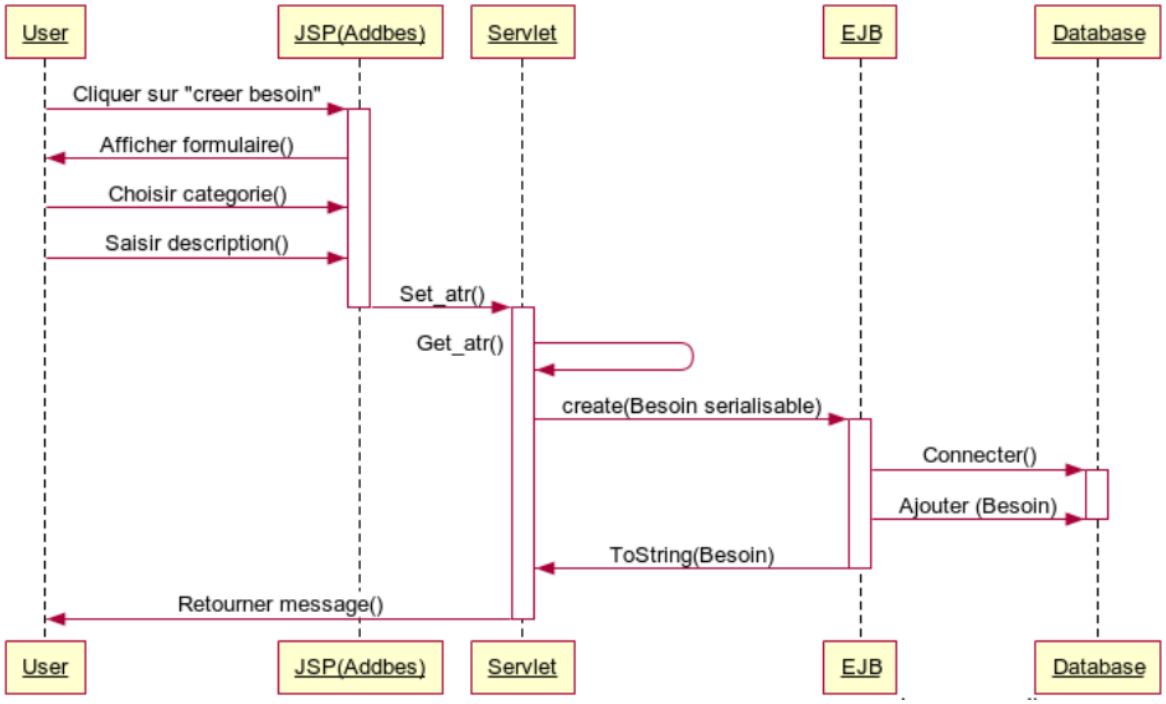


Diagramme de sequence Ajouter besoin

Cas d'utilisation : Confirmer l'ajout du besoin

But : Ce cas d'utilisation permet au admin de confirmer l'ajout d' un nouveau besoin

Les acteurs : Admin

Description textuelle : l'admin Consulte la liste des besoins et accepte le nouveau besoin.

Scenario nominale :

1. L'admin consulte la liste des besoins
2. Le système affiche la page jsp « Allbes.jsp » qui contient un tableau de tous les besoins
3. L'utilisateur choisie le besoin a accepté
4. La page jsp envoit le données à la servlet «addbes.java »
5. La servlet recupére les données en utilisant(JNDI) la methode « Get_atr() »
6. La servlet cree la liste de type Besoin
7. La servlet appel l'EJB BesoinFascadeLocal
8. La servlet cree ce besoin a travers l'EJB et retourne un message de validation de l'ajout

Scenario alternatif :

8.L'echec de l'ajout du besoin ➔ un message d'erreur s'affiche et retourne vers la deuxieme etape

Cas d'utilisation : Supprimer le besoin

But : Ce cas d'utilisation permet au admin de supprimé un besoin

Les acteurs : Admin

Description textuelle : l'admin Consulte la liste des besoins et supprime le nouveau besoin.

Scenario nominale :

1. L'admin consulte la liste des besoins
2. Le système affiche la page jsp « Allbes.jsp » qui contient un tableau de tous les besoins
3. L'utilisateur choisie le besoin a supprimé
4. La page jsp envoit le données à la servlet «deletebes.java »
5. La servlet recupére les données en utilisant(JNDI) la methode « Get_atr() »
6. La servlet cree la liste de type Besoin
7. La servlet appel l'EJB BesoinFascadeLocal
8. La servlet supprime ce besoin a travers l'EJB et retourne un message de validation de la suupression

Scenario alternatif :

8.L'echec de la suppression du besoin ➔ un message d'erreur s'affiche et retourne vers la deuxieme etape

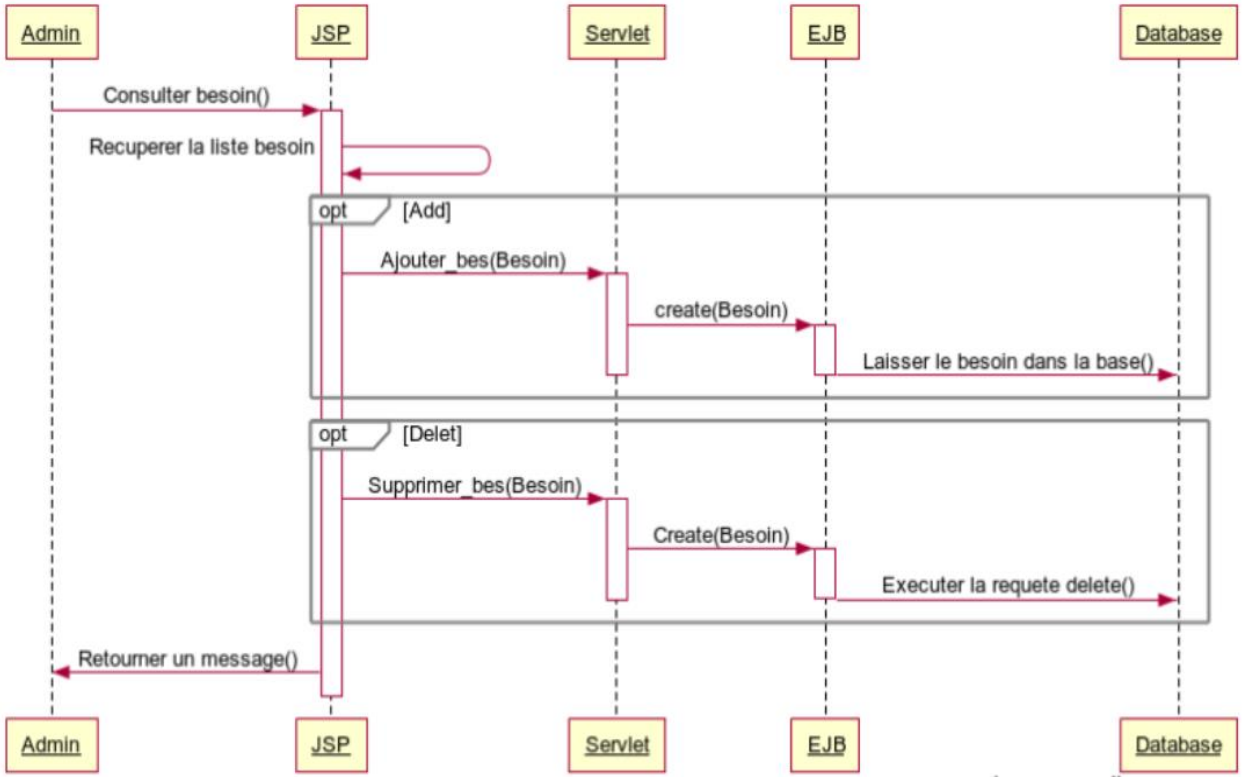


Diagramme de sequence :Confirmer ou supprimer le besoin

Cas d'utilisation de modification du besoin

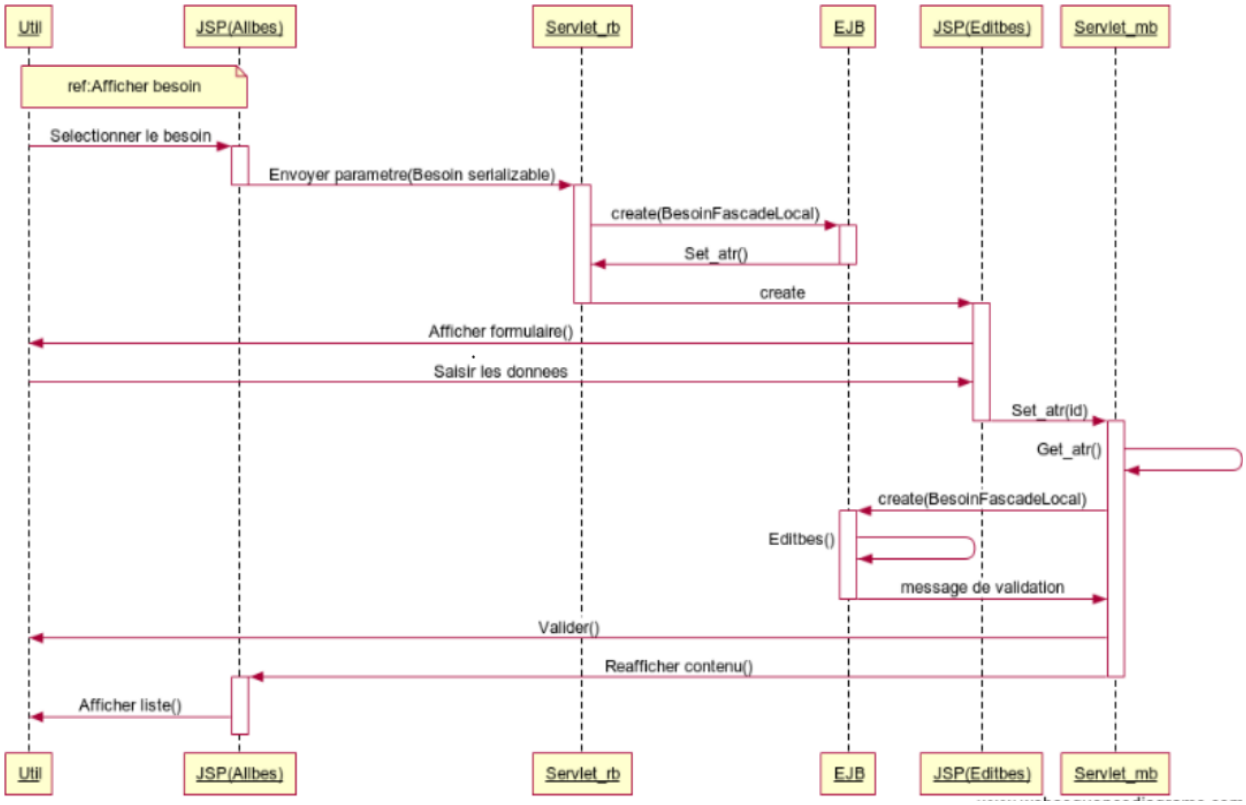


Diagramme de séquences modifier besoin

- Description detaille des classes :

Classe	Attribut	Methode	Description
User	Id_user	Get_atr()	Recuperer l'attribut
	Nom_util	Set_atr()	Envoyer l'attribut
	Mot_pass	Ajouter_user()	Ajouter un nouveau utilisateur
	E_mail		
Besoin	Id_bes	Get_atr()	Recuperer l'attribut
	Categorie	Set_atr()	Envoyer l'attribut
	Description	Ajouter_bes()	Ajouter un nouveau besoin
		Modifier_bes()	Modifier la description d'un besoin
		Supprimer_bes()	Le non confirmation d'un nouveau besoin

Logiciel	Id_log	Get_atr()	Recuperer l'attribut
	Nom	Set_atr()	Envoyer l'attribut
	Version	Ajouter_Log()	Ajouter un nouveau logiciel

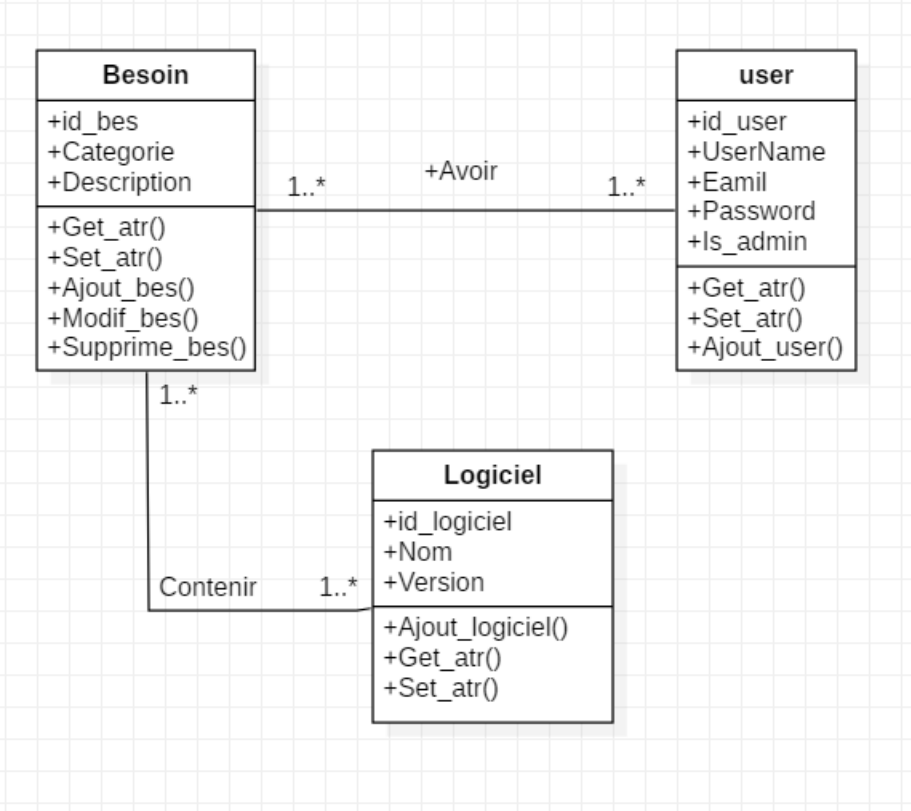


Diagramme de classe générale