



RAPPORT DU PROJET DE JAKARTA EE

IMPLÉMENTATION D'UNE PLATEFORME DE VENTE DE VÉLOS

Mamadou WADE

TABLE DES MATIERES

Titres	Pages
I. INTRODUCTION	3
II. JAKARTA EE	4
III. SERVICE WEB RESTFUL	16

I. INTRODUCTION

Contexte et justification

Dans le monde dynamique du commerce des vélos, où les attentes des clients évoluent rapidement et où la concurrence est féroce, il est impératif de rester à la pointe de la technologie pour prospérer. C'est dans ce contexte passionnant que nous avons entrepris le développement d'une application innovante dédiée à la gestion intégrée des systèmes de vente de vélos.

Le paysage actuel du marché du vélo est caractérisé par une demande croissante de solutions de vente plus efficaces, transparentes et personnalisées. Les détaillants de vélos, qu'ils soient de petites boutiques locales ou des entreprises en pleine expansion, sont confrontés à des défis tels que la gestion des stocks, la traçabilité des commandes, la fidélisation de la clientèle et l'adaptation rapide aux nouvelles tendances du marché.

C'est dans ce contexte je me suis lancé dans le développement d'une application complète et novatrice qui vise à transformer la manière dont les entreprises gèrent leurs opérations de vente de vélos. Mon objectif est de fournir une solution qui va au-delà de la simple gestion des transactions pour offrir une expérience globale améliorée, tant pour les commerçants que pour leurs clients.

L'application que nous concevons s'articule autour de plusieurs axes clés :

1. **Gestion Intelligente des Produits** : Permettant aux commerçants de mettre en avant et de gérer facilement leur catalogue de vélos, tout en fournissant aux clients des informations détaillées sur chaque modèle.
2. **Suivi des Stocks en Temps Réel** : Offrant une visibilité instantanée sur les niveaux de stock, permettant ainsi une gestion plus précise et une réduction des risques de pénurie ou d'excédent.
3. **Processus de Commande Transparent** : Simplifiant le suivi des commandes depuis leur réception jusqu'à la livraison, assurant une expérience client fluide et fiable.
4. **Personnalisation des Promotions et des Remises** : Permettant aux commerçants de mettre en œuvre des stratégies marketing ciblées, favorisant l'acquisition et la fidélisation de la clientèle.
5. **Intégration avec les Plateformes de Commerce Électronique** : Facilitant la gestion synchronisée des stocks et des commandes en ligne, simplifiant ainsi l'expérience omnicanale.

Ce projet ambitieux vise à créer une solution, avec la plateforme Jakarta EE, qui s'adapte aux besoins spécifiques de chaque entreprise, qu'elle soit une petite boutique de quartier ou une enseigne nationale. C'est application qui propulsera les entreprises de vente de vélos vers de nouveaux sommets en matière d'efficacité, de productivité et d'expérience client. Je suis convaincu que cette initiative marquera une avancée significative dans l'industrie du vélo, offrant des opportunités nouvelles et passionnantes pour tous les acteurs du secteur.

II) PARTIE 1 : APPLICATION WEB (JAKARTA EE)

Dans le cadre de notre projet de développement d'une application de vente de vélos avec Jakarta EE, ce rapport offre une vue approfondie des différentes étapes entreprises pour concevoir une solution robuste et fonctionnelle. Du choix du serveur d'application à la configuration de la base de données, chaque aspect de ce développement est minutieusement documenté.

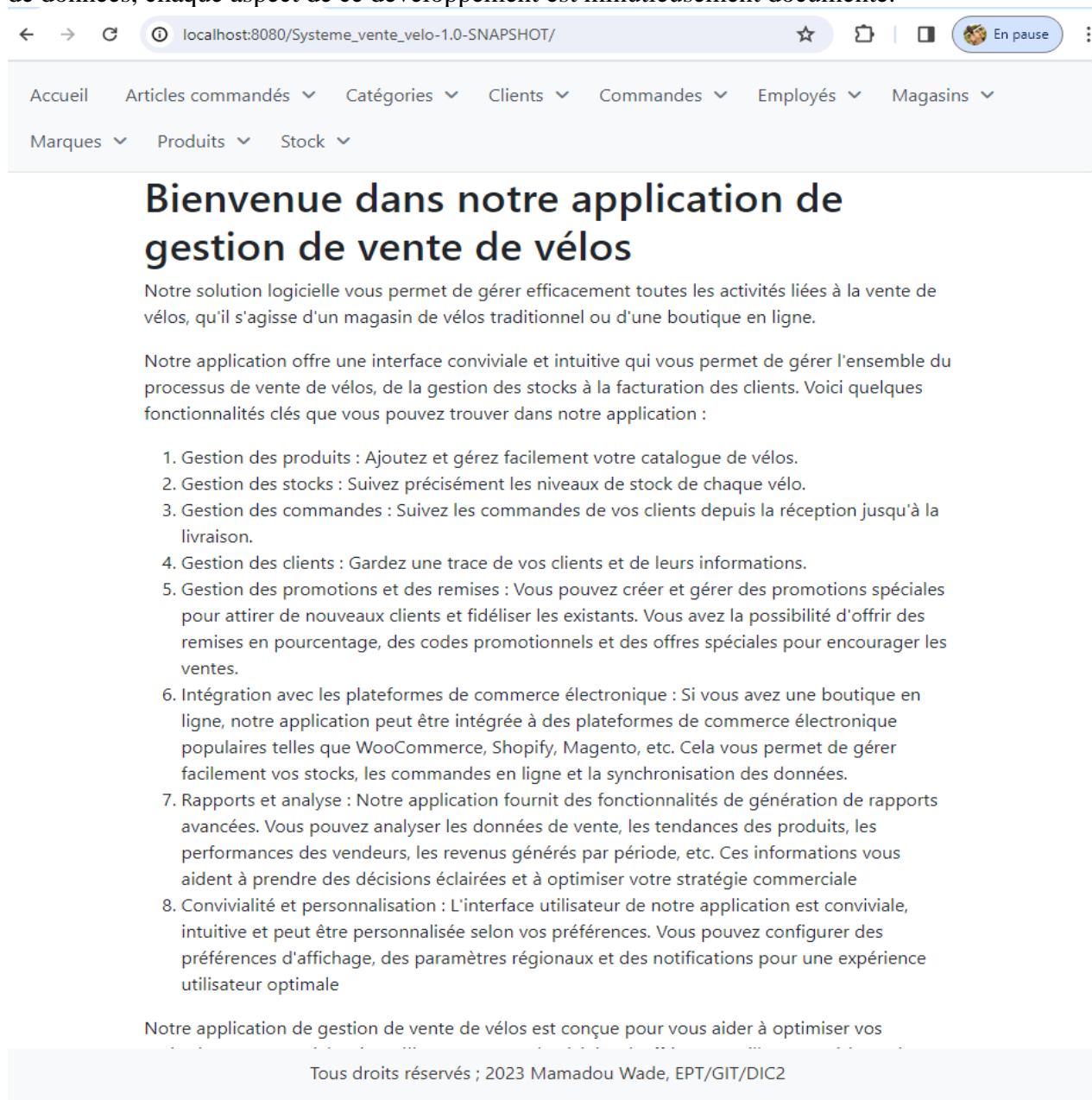


Figure 1: Interface web de l'application

1. Configuration des serveurs

1.1 Installation serveur d'application

Le serveur d'application WildFly a été sélectionné et installé pour répondre aux exigences de Jakarta EE, fournissant une plateforme solide pour la mise en œuvre des Enterprise Beans (EJB). Cette étape garantit une base stable et performante pour notre application.

1.2 Installation SGBD

L'installation de PostgreSQL, en remplacement de MySQL, a été entreprise dans le but de diversifier les options de SGBD. Cette décision a permis une exploration plus approfondie de la configuration Jakarta Persistence (JPA) avec d'autres systèmes de gestion de base de données.

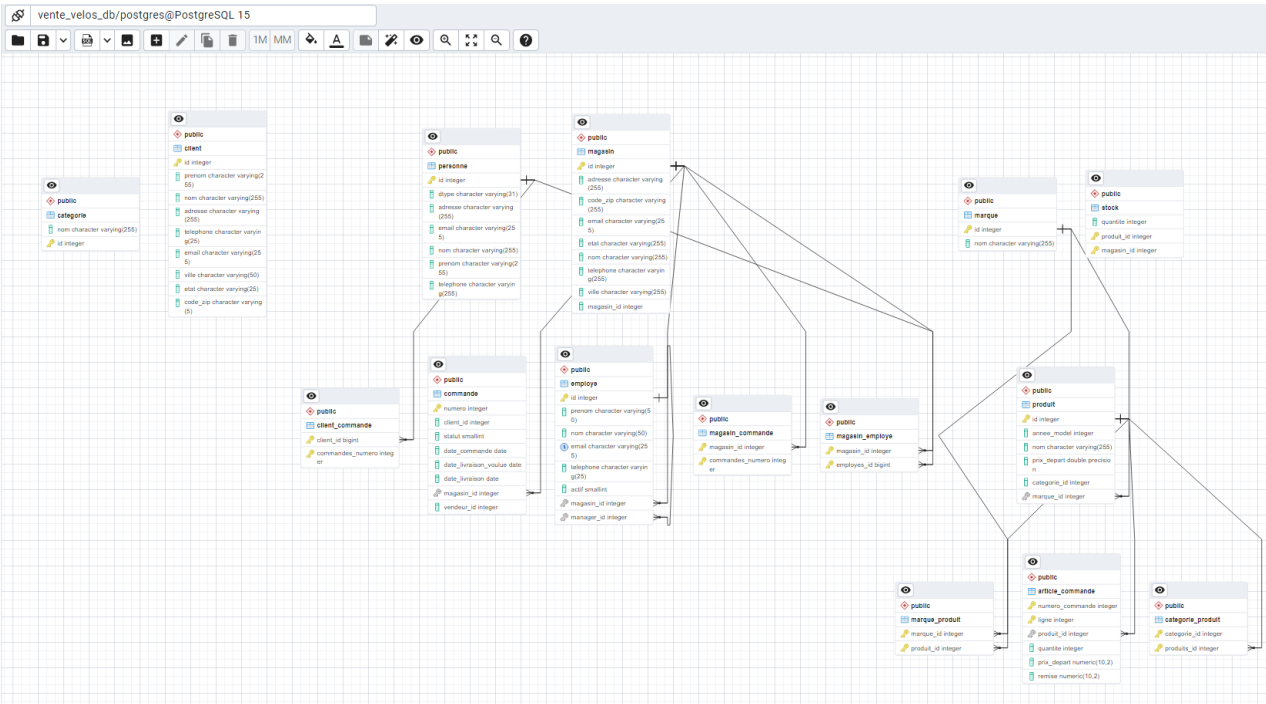


Figure 2: Les tables sur postgres

1.3 Création source de données

La création réussie d'une source de données et de la ressource associée constitue une étape cruciale, permettant aux applications déployées d'accéder efficacement à la base de données à partir du serveur Jakarta EE, assurant ainsi une connectivité fluide.

Edit JDBC Connection Pool Properties

Modify properties of an existing JDBC connection pool.

Pool Name: Postgres_pool

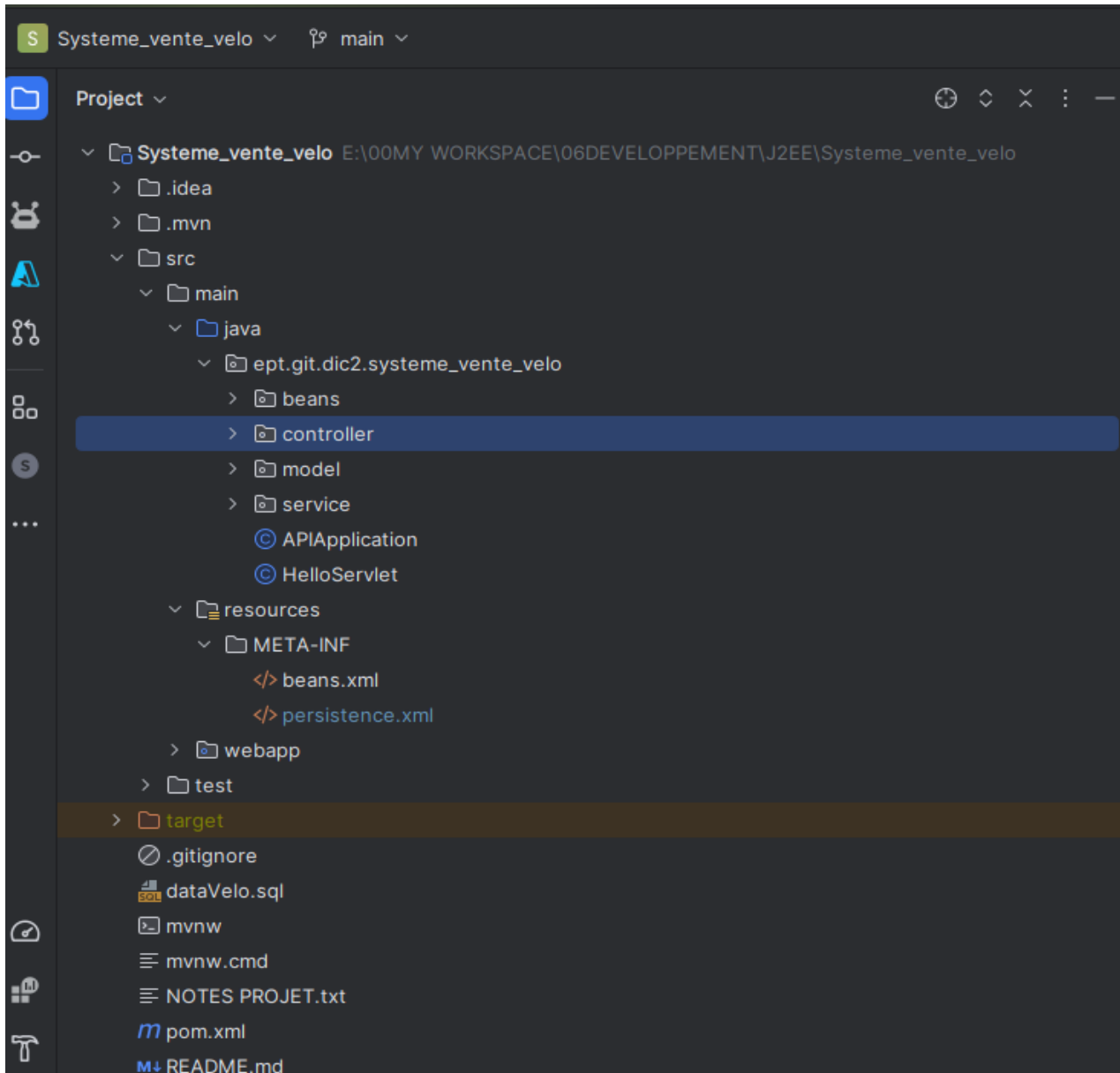
Select	Name	Value
<input type="checkbox"/>	password	MomoWade
<input type="checkbox"/>	databaseName	vente_velos_db
<input type="checkbox"/>	serverName	localhost
<input type="checkbox"/>	user	postgres
<input type="checkbox"/>	portNumber	5432

Figure 3: Configuration sur glassfish

2. Mapping JPA

2.1 Création nouveau projet

Un nouveau projet web a été créé sur **IntelliJ**, intégrant la source de données précédemment établie. Cette étape essentielle assure une intégration optimale entre le projet et la base de données, jetant les bases d'une gestion de données cohérente.



2.2 Création des entités

Les entités ont été créées en suivant le diagramme de classes de la figure donnée, avec une attention particulière portée à l'utilisation de la stratégie JOINED pour l'héritage. Cette approche

garantit une modélisation appropriée des données dans le contexte de notre application.

```

<persistence-unit name="vente_velo_persis">
  <jta-data-source>jdbc/PostgresDataSource</jta-data-source>
  <class>ept.git.dic2.systeme_vente_velo.model.Categorie</class>
  <class>ept.git.dic2.systeme_vente_velo.model.Client</class>
  <class>ept.git.dic2.systeme_vente_velo.model.Commande</class>
  <class>ept.git.dic2.systeme_vente_velo.model.Employe</class>
  <class>ept.git.dic2.systeme_vente_velo.model.Magasin</class>
  <class>ept.git.dic2.systeme_vente_velo.model.Marque</class>
  <class>ept.git.dic2.systeme_vente_velo.model.Produit</class>
  <class>ept.git.dic2.systeme_vente_velo.model.ArticleCommande</class>
  <class>ept.git.dic2.systeme_vente_velo.model.CategorieProduit</class>
  <class>ept.git.dic2.systeme_vente_velo.model.ClientCommande</class>
  <class>ept.git.dic2.systeme_vente_velo.model.Eleve</class>
  <class>ept.git.dic2.systeme_vente_velo.model.MagasinCommande</class>
  <class>ept.git.dic2.systeme_vente_velo.model.MagasinEmploye</class>
  <class>ept.git.dic2.systeme_vente_velo.model.MarqueProduit</class>
  <class>ept.git.dic2.systeme_vente_velo.model.Personne</class>
  <class>ept.git.dic2.systeme_vente_velo.model.Stock</class>

  <properties>
    <property name="toplink.jdbc.url" value="jdbc:postgresql://localhost:5432/vente_velos_db"/>
    <property name="toplink.jdbc.driver" value="org.postgresql.Driver"/>
    <property name="toplink.jdbc.user" value="postgres"/>
    <property name="toplink.jdbc.password" value="MomoWade"/>
    <property name="hibernate.connection.url" value="jdbc:postgresql://localhost:5432/vente_velo"/>
    <property name="hibernate.connection.driver_class" value="org.postgresql.Driver"/>
  </properties>

```

3. Développement des EJB et des façades

3.1 Création des Facades

Les façades nécessaires ont été mises en place pour assurer un enregistrement efficace des entités. Cette couche d'abstraction facilite l'interaction entre les entités et la base de données, optimisant ainsi les opérations de gestion des données.

```

public abstract class AbstractBean<T> {

    5 usages
    private Class<T> entityClass;

    ± MOMO_ADMIN
    public AbstractBean(Class<T> entityClass) { this.entityClass = entityClass; }

    10 implementations ± MOMO_ADMIN
    protected abstract EntityManager getEntityManager();

    ± MOMO_ADMIN
    public void create(T entity) { getEntityManager().persist(entity); }

    ± MOMO_ADMIN
    public void edit(T entity) { getEntityManager().merge(entity); }

    ± MOMO_ADMIN
    public void remove(T entity) { getEntityManager().remove(getEntityManager().merge(entity)); }

    ± MOMO_ADMIN
    public T find(Object id) { return getEntityManager().find(entityClass, id); }

    ± MOMO_ADMIN
    public List<T> findAll() {
        jakarta.persistence.criteria.CriteriaQuery cq = getEntityManager().getCriteriaBuilder().
        cq.select(cq.from(entityClass));
        return getEntityManager().createQuery(cq).getResultList();
    }

    no usages ± MOMO_ADMIN
    public List<T> findRange(int[] range) {
        jakarta.persistence.criteria.CriteriaQuery cq = getEntityManager().getCriteriaBuilder().
        cq.select(cq.from(entityClass));
        jakarta.persistence.Query q = getEntityManager().createQuery(cq);
        q.setMaxResults(range[1] - range[0] + 1);
        q.setFirstResult(range[0]);
        return q.getResultList();
    }
}

```


Figure 4: Facade abstraction

```

package ept.git.dic2.système_vente_velo.beans;

import ...
10 usages  MOMO_ADMIN
@Stateless
public class ProduitBean extends AbstractBean<Produit> {
    1 usage
    @PersistenceContext(name = "dic2PU")
    private EntityManager entityManager;
    no usages  MOMO_ADMIN
    public ProduitBean() { super(Produit.class); }
    MOMO_ADMIN
    protected EntityManager getEntityManager() { return entityManager; };
}

```

Figure 5: Facade Produit

3.2 Initialisation de la BD

Une solution pratique pour l'initialisation de la base de données avec les données de test a été implémentée, garantissant une base de données prête à l'emploi avec des données significatives pour les tests et le développement ultérieurs.

Voici un exemple de table : **l'entité client**

public.client/vente_velos_db/postgres@PostgreSQL 15

Query Query History Scratch Pad

```

1 SELECT * FROM public.client
2 ORDER BY id ASC

```

Data Output Messages Notifications

	id [PK] integer	prenom character varying (255)	nom character varying (255)	adresse character varying (255)	telephone character v
1	1	Debra	Burks	9273 Thorne Ave.	
2	2	Kasha	Todd	910 Vine Street	[null]
3	4	Daryl	Spence	988 Pearl Lane	[null]
4	5	Charolette	Rice	107 River Dr.	(916) 381-
5	6	Lyndsey	Bean	769 West Road	[null]
6	7	Latasha	Hays	7014 Manor Station Rd.	(716) 986-
7	8	Jacqueline	Duncan	15 Brown St.	[null]
8	9	Genoveva	Baldwin	8550 Spruce Drive	[null]
9	10	Pamelia	Newman	476 Chestnut Ave.	[null]
10	11	Deshawn	Mendoza	8790 Cobblestone Street	[null]
11	12	Robby	Sykes	486 Rock Maple Street	(516) 583-
12	13	Lashawn	Ortiz	27 Washington Rd.	[null]

3.3 Session Bean pour l'envoi d'email

La création d'une EJB Session Bean pour gérer l'envoi d'e-mails renforce les fonctionnalités de l'application. En intégrant cette fonctionnalité, l'application peut maintenant notifier les utilisateurs de manière proactive via le canal de communication choisi.

```
package ept.git.dic2.systeme_vente_velo.beans;

> import ...

2 usages  ▲ MOMO_ADMIN
@Stateless
public class EmailBean {

    1 usage
    @Resource(name = "java:jboss/mail/Default")
    private Session mailSession;

    3 usages  ▲ MOMO_ADMIN
    public void sendEmail(String recipient, String subject, String message) throws Messaging
        Message email = new MimeMessage(mailSession);
        email.setRecipients(Message.RecipientType.TO, InternetAddress.parse(recipient));
        email.setSubject(subject);
        email.setText(message);
        Transport.send(email);
    }
}
```

3.4 Envoi d'alertes automatiques

Deux EJB ont été développés pour automatiser l'envoi d'e-mails à chaque démarrage et arrêt de l'application, ainsi que pour les alertes périodiques sur l'état des stocks. Ces fonctionnalités contribuent à la surveillance proactive et à la maintenance de l'application.

```
package ept.git.dic2.systeme_vente_velo.beans; //package ept.git.dic2.systeme_vente_velo.t
import ...

no usages  ± Momo_Wade
@Singleton
//@Startup
public class StartupShutdownBean {
    2 usages
    @EJB
    private EmailBean emailBean;

    ± Momo_Wade
    @PostConstruct
    public void sendStartupEmail() {
        try {
            emailBean.sendEmail( recipient: "recipient@example.com", subject: "Application Star
        } catch (MessagingException e) {
            e.printStackTrace();
        }
    }

    ± Momo_Wade
    @PreDestroy
    public void sendShutdownEmail() {
        try {
            emailBean.sendEmail( recipient: "recipient@example.com", subject: "Application Shut
        } catch (MessagingException e) {
            e.printStackTrace();
        }
    }
}
```

4. Développement des interfaces web JSF

4.1 CRUD sur les entités

Le développement des pages web pour le CRUD des entités, en utilisant des composants comme PrimeFaces DataView et DataTable, garantit une interface utilisateur conviviale et fonctionnelle pour la gestion des données.

Voici des exemples :

a) Entité Produit

The screenshot shows a web browser at the URL `localhost:8080/Systeme_vente_velo-1.0-SNAPSHOT/produitAjout.xhtml`. The navigation menu includes: Accueil, Articles commandés, Catégories, Clients, Commandes, Employés, Magasins, Marques, Produits, and Stock. The main content area is titled "Ajout d'un produit" and contains a form with the following fields:

- Nom
- Année du modèle
- Prix de départ
- Catégorie ID
- Marque ID

At the bottom of the form are two buttons: "Enregistrer" (red) and "Annuler" (green).

localhost:8080/Systeme_vente_velo-1.0-SNAPSHOT/produitList.xhtml

Accueil Articles commandés Catégories Clients Commandes Employés Magasins Marques Produits Stock

Liste des Produits

ID	Année du Modèle	Nom	Prix de Départ	Catégorie ID	Marque ID	Modification	Suppression
1	2016	Trek 820 - 2016	379.99	6	9	Modifier	Supprimer
3	2016	Surly Wednesday Frameset - 2016	999.99	6	8	Modifier	Supprimer
4	2016	Trek Fuel EX 8 29 - 2016	2899.99	6	9	Modifier	Supprimer
6	2016	Surly Ice Cream Truck Frameset - 2016	469.99	6	8	Modifier	Supprimer
7	2016	Trek Slash 8 27.5 - 2016	3999.99	6	9	Modifier	Supprimer
8	2016	Trek Remedy 29 Carbon Frameset - 2016	1799.99	6	9	Modifier	Supprimer
9	2016	Trek Conduit+ - 2016	2999.99	5	9	Modifier	Supprimer
		Surly					

b) Entité client

localhost:8080/Systeme_vente_velo-1.0-SNAPSHOT/clientAjout.xhtml

Accueil Articles commandés Catégories Clients Commandes Employés Magasins Marques Produits Stock

Ajout d'un client

Prénom

Nom

Adresse

Téléphone

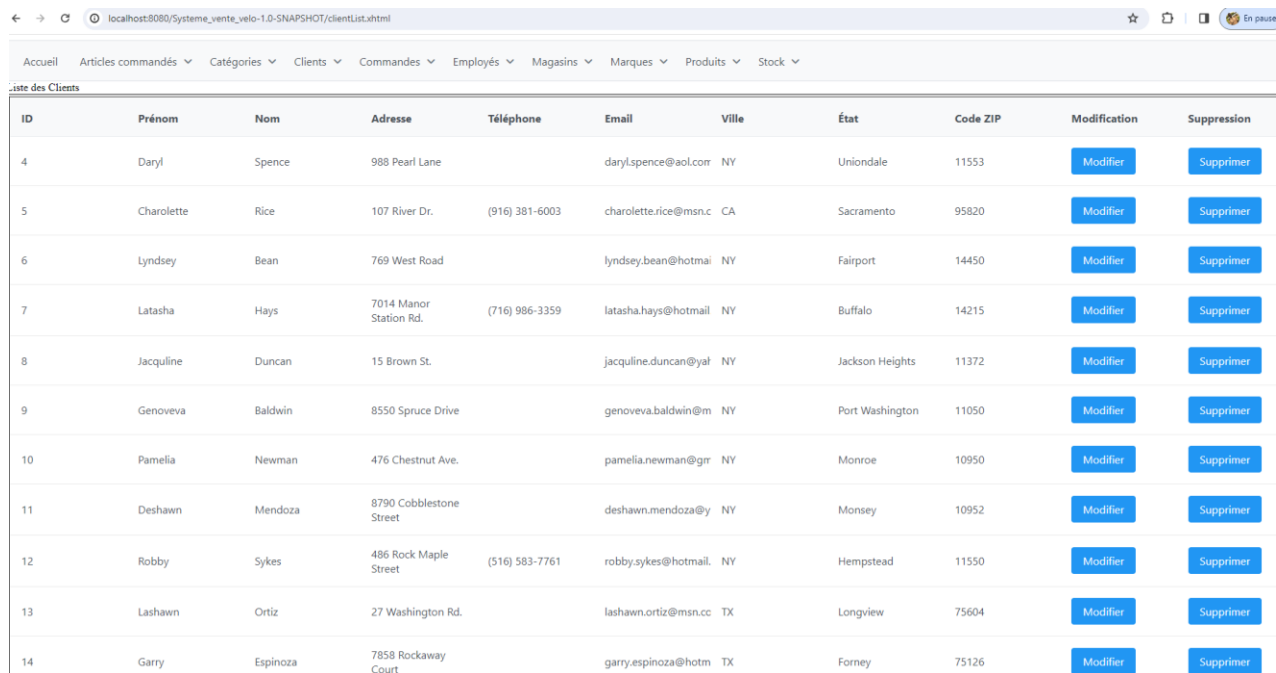
Email

Ville

État

Code ZIP

Enregistrer Annuler



ID	Prénom	Nom	Adresse	Téléphone	Email	Ville	État	Code ZIP	Modification	Suppression
4	Daryl	Spence	988 Pearl Lane		daryl.spence@aol.com	NY	Uniondale	11553	Modifier	Supprimer
5	Charollette	Rice	107 River Dr.	(916) 381-6003	charollette.rice@msn.c	CA	Sacramento	95820	Modifier	Supprimer
6	Lyndsey	Bean	769 West Road		lyndsey.bean@hotmail	NY	Fairport	14450	Modifier	Supprimer
7	Latasha	Hays	7014 Manor Station Rd.	(716) 986-3359	latasha.hays@hotmail	NY	Buffalo	14215	Modifier	Supprimer
8	Jacqueline	Duncan	15 Brown St.		jacqueline.duncan@ya	NY	Jackson Heights	11372	Modifier	Supprimer
9	Genoveva	Baldwin	8550 Spruce Drive		genoveva.baldwin@m	NY	Port Washington	11050	Modifier	Supprimer
10	Pamella	Newman	476 Chestnut Ave.		pamella.newman@grr	NY	Monroe	10950	Modifier	Supprimer
11	Deshawn	Mendoza	8790 Cobblestone Street		deshawn.mendoza@y	NY	Monsey	10952	Modifier	Supprimer
12	Robby	Sykes	486 Rock Maple Street	(516) 583-7761	robby.sykes@hotmail	NY	Hempstead	11550	Modifier	Supprimer
13	Lashawn	Ortiz	27 Washington Rd.		lashawn.ortiz@msn.c	TX	Longview	75604	Modifier	Supprimer
14	Garry	Espinoza	7858 Rockaway Court		garry.espinoza@hotm	TX	Forney	75126	Modifier	Supprimer

4.2 Créer un menu

L'ajout d'un menu, géré par un composant de Primefaces, facilite la navigation entre les différentes pages de l'application, améliorant ainsi l'expérience utilisateur globale.



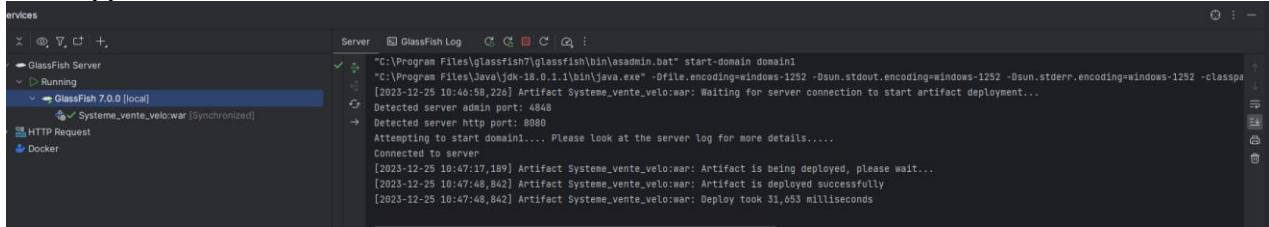
4.3 Filtrage, triage et pagination

La mise en œuvre d'une page de liste des produits avec le composant DataTable en mode lazy, avec pagination, filtrage et triage, offre une expérience utilisateur fluide et optimisée pour la gestion des stocks.

5. Déploiement dans un environnement de production

L'installation des outils nécessaires au déploiement dans le serveur s'est déroulée avec succès, préparant ainsi l'environnement pour le déploiement de l'application.

Le déploiement réussi de l'application à la racine du serveur Linux signifie qu'elle est prête à être utilisée dans un environnement de production, marquant la conclusion réussie de la phase de développement.



Conclusion

Cette partie du rapport offre une vision complète du développement de notre application de vente de vélos avec Jakarta EE. Chaque section souligne l'importance des décisions prises et des fonctionnalités implémentées, garantissant ainsi la création d'une application robuste, performante et prête à répondre aux besoins du marché.

III) PARTIE 2 : Service Web RESTful et du Client Angular

La deuxième phase de notre projet vise à concevoir un service web RESTful pour répondre aux besoins des clients mobiles, tels qu'Android, et des clients web, notamment Angular. Ce rapport détaille chaque étape du processus, de la définition des ressources à la création du client Angular, en mettant l'accent sur la conformité aux normes RESTful et la documentation exhaustive des services web.

1. Définir les ressources

Dans cette étape cruciale, nous avons identifié toutes les ressources nécessaires au bon fonctionnement des applications mobiles et web. Des entités telles que les produits, les catégories, les commandes et les clients ont été minutieusement répertoriées, assurant une compréhension claire des données manipulées par le service web.

Chaque ressource a été associée à des actions spécifiques. Les URI et les méthodes HTTP ont été soigneusement définies conformément aux normes RESTful. Cette approche garantit une communication efficace et cohérente entre le service web et ses clients, assurant une expérience utilisateur sans heurts.

2. Conception des services web

2.1 Développement des services web

Le développement des services web a été effectué en suivant les spécifications définies dans la section précédente. Des fonctionnalités telles que la création, la lecture, la mise à jour et la suppression ont été implémentées pour chaque ressource. Une attention particulière a été accordée à la robustesse et à l'efficacité de ces services.

Voici un exemple d'implémentation d'une ressource (pour l'entité commande) :

```

1 package ept.git.dic2.système_vente_velo.service;
2
3 import ...
4
5 @MOMO_ADMIN
6 @Path("/")
7 public class CommandeResource {
8     7 usages
9     @EJB
10     private CommandeBean commandeFacade;
11
12     @MOMO_ADMIN
13     @GET
14     @Produces({MediaType.APPLICATION_JSON, MediaType.APPLICATION_XML})
15     public List<Commande> getListCommandes() { return commandeFacade.findAll(); }
16
17     @MOMO_ADMIN
18     @GET
19     @Path("/{id}")
20     @Produces({MediaType.APPLICATION_JSON, MediaType.APPLICATION_XML})
21     public Response findCommandeById(@PathParam("id") int id) {
22         Commande commande = commandeFacade.find(id);
23         if (commande == null) {
24             Response reponse = new Response("La commande dont le id est "+commande.getNumero()+"est introuvable");
25             return Response.status(Response.Status.NOT_FOUND).entity(reponse).build();
26         }
27         return Response.status(Response.Status.OK).entity(commande).build();
28     }
29
30     @MOMO_ADMIN
31     @POST
32     @Consumes({MediaType.APPLICATION_JSON, MediaType.APPLICATION_XML})
33     public Commande addCommandes(Commande e) {

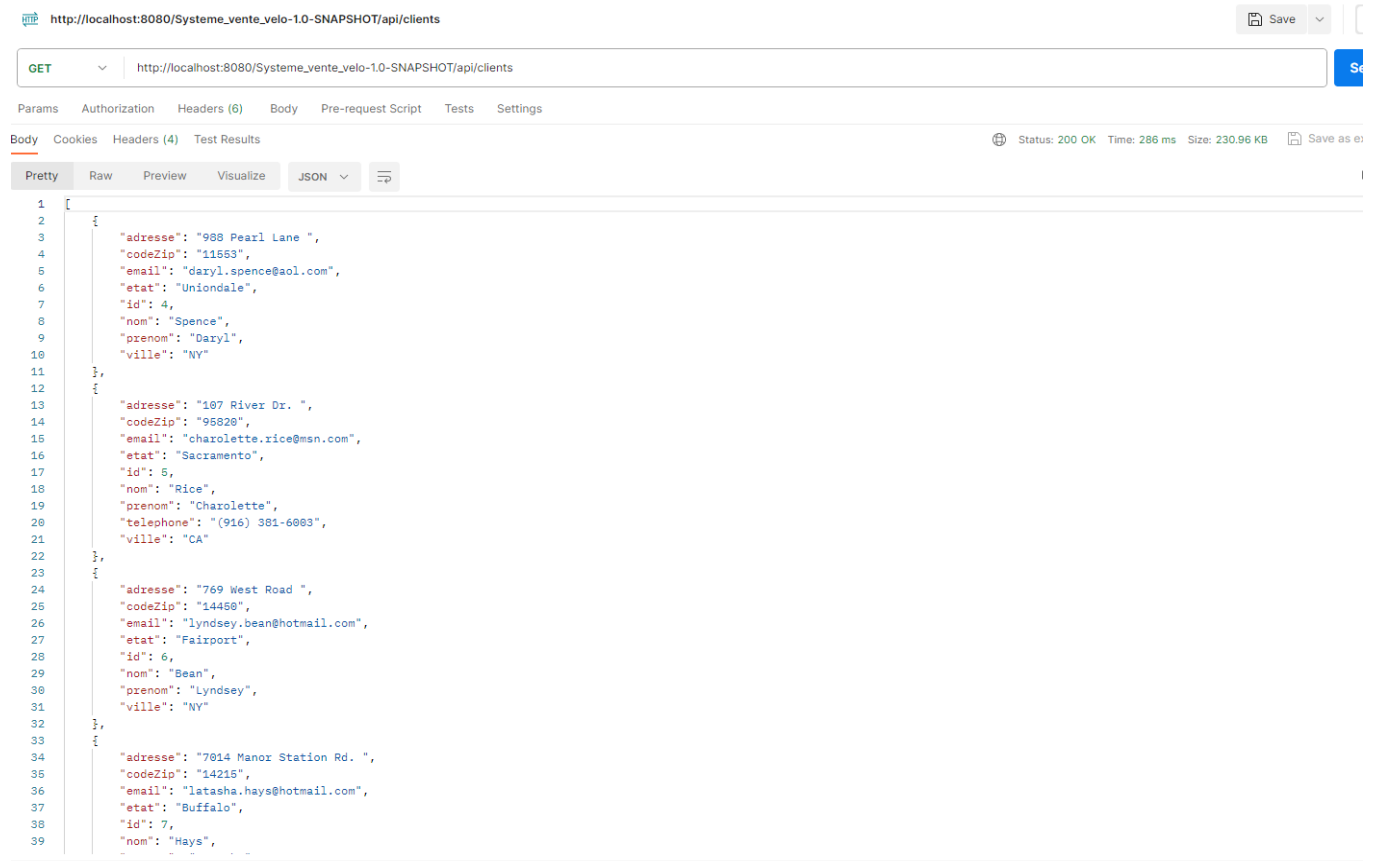
```

Nous avons suivi le modèle pour implémenter les autres.

3. Test des services web

Nous avons utilisé **Postman** pour effectuer des tests approfondis des services web. La documentation exhaustive a permis de valider chaque endpoint de manière systématique. La nécessité d'automatiser les tests unitaires a été mise en avant, avec la création d'un projet dédié intégrant ces tests. Cette approche assure une validation systématique à chaque modification du code.

a) méthode GET



b) méthode POST

The screenshot shows a REST client interface with the following details:

- Method:** POST
- URL:** http://localhost:8080/Systeme_vente_velo-1.0-SNAPSHOT/api/categories
- Body (raw JSON):**

```
{
  "nom": "Vélo tout-terrain"
}
```
- Response:** 200 OK, 11 ms, 231 B. The response body is displayed in a pretty-printed JSON format:

```
{
  "id": 199,
  "nom": "Vélo tout-terrain"
}
```

c) méthode PUT

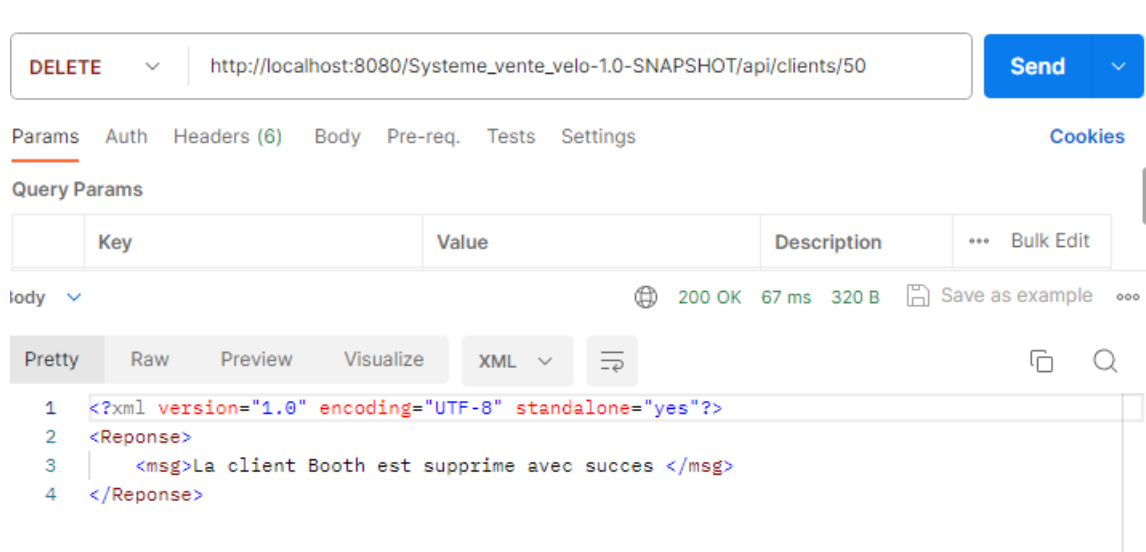
The screenshot shows a REST client interface with the following details:

- Method:** PUT
- URL:** http://localhost:8080/Systeme_vente_velo-1.0-SNAPSHOT/api/clients/2
- Body (raw JSON):**

```
{
  "adresse": "910 Vine Street ",
  "codeZip": "95008",
  "email": "kasha.todd@yahoo.com",
  "etat": "California",
  "nom": "Todd",
  "prenom": "Kasha",
  "ville": "CA"
}
```
- Response:** 200 OK, 15 ms, 344 B. The response body is displayed in a pretty-printed JSON format:

```
{
  "adresse": "910 Vine Street ",
  "codeZip": "95008",
  "email": "kasha.todd@yahoo.com",
  "etat": "California",
  "id": 2,
  "nom": "Todd",
  "prenom": "Kasha",
  "ville": "CA"
}
```

d) méthode DELETE



4. Client Angular+bootstrap

4.1 Création du client Angular

Pour les interfaces web, un client Angular a été créé en réutilisant des composants Angular existants. Des composants paramétrables ont été développés au besoin, utilisant le routage Angular pour une navigation fluide entre les pages. Les services Angular ont été employés pour établir une communication efficace avec les services web RESTful.

Conclusion

Cette deuxième phase du projet a abouti à un service web RESTful conforme aux normes, associé à un client Angular offrant une expérience utilisateur intuitive. Chaque étape du processus, de la définition des ressources à la documentation et aux tests, a été réalisée avec précision, garantissant la stabilité et la qualité du système. Ce rapport fournit une référence complète pour les futures évolutions du système.