Une image contenant texte, clipart, graphiques vectoriels

Description générée automatiquement

Une image contenant graphique

Description générée automatiquement

studi.com

**ÉVALUATION BLOC 3**

Développement d’une solution digitale avec Java

|  |  |  |  |
| --- | --- | --- | --- |
| Date de début | 01/07/2023 | Elève(s) | Josselin Fatah-Roux |
| Date de fin | 26/09/2023 | Evaluateur(s) | Non connu |
| Nombre de jour effectif | 88 |  |

Contexte du projet

Mercadona, célèbre entreprise dans le milieu du retail dispose de 1675 magasins en Espagne réalisant ainsi un chiffre d’affaires de 20 milliards d’euros. Son succès se dessine à travers des promotions très fréquentes, il y a toujours un minimum d’une centaine de produits en promotion et à tout moment de l’année.

Pour le moment, Mercadona ne fonctionne que par des tracts publicitaires. Le problème qui se pose est la dimension écologique : Mercadona prend de plus en plus conscience que les énergies doivent être économisées et souhaite ne plus proposer des tracts, mais une application web.

Mercadona n’est pas expert du web, de ce fait, ils ont pris contact avec PromoWeb dont vous faites partie, une ESN spécialisée dans ce domaine. Mercadona a nommé dans leurs effectifs une personne qui va se charger de présenter de projet à PromoWeb, il se nomme José.

Les missions que vous devrez effectuer sont les suivantes :

Voici le processus de réservation donné :

* Analyse du projet
* Production des documents nécessaires au développement d’une application
* Développement de l’application

Afin de mener cela à bien, José, connaissant tous les tenants et aboutissant, vous fait un résumé de tout ce que le client attend.

Le client, Mercadona, souhaite une application en trois parties :

* Front-end : permettant à un utilisateur de consulter les promotions
* Back-end : enregistrant les données vers la base de données
* Base de données : stockage de l’information

José vous explique que le but de l’application est d’afficher des promotions.

Pour ce faire, votre application web devra pouvoir authentifier un visiteur (uniquement possible pour les administrateurs) afin de pouvoir ajouter des promotions sur des produits.

Le processus de mise en ligne d’une promotion est le suivant :

* L’administrateur crée un produit sur son espace avec un premier prix :
* Un produit est défini par :
* Un libelle, une description, un prix, une image ainsi qu’une catégorie
* Une catégorie est définie par un libelle uniquement
* Une fois les produits créent, il est possible de le mettre en promotion, pour ce faire, il doit accéder au produit depuis l’espace Admin et clic sur un bouton « promotion » et saisis un pourcentage de promotion à appliquer au prix. Le nouveau prix est calculé automatiquement :
* Une promotion est caractérisée par :
* Une date de début ainsi qu’une date de fin
* Un pourcentage de remise
* Tous les produits, même ceux sans promotion, sont visibles sur l’onglet « catalogue », accessible depuis le menu de l’application web sans aucune authentification. Un filtre est disponible sur cette page afin d’afficher les produits par catégorie.
* La vue détaillée d’un produit n’est pas possible, la page « catalogue » affiche toutes les informations d’un produit : lorsqu’un produit est en promotion, son prix doit être affiché en gras et en rouge.

José vous demande d’effectuer les missions qui vous sont exposées dans les dossiers suivants.

Aucune contrainte technique n’est imposée, mise à part PostGreSQL ainsi que Java pour la partie back-end.

Si vous n’avez pas d’idée technique, voici un exemple possible :

* Front : template HTML Spring (thymeleaf) ou un front classique (ReactJS, Angular, etc)
* Back-end : Java (Spring Boot) => obligatoire
* Base de données : PostgreSQL => obligatoire
* Déploiement : fly.io , Heroku

# Dossier 1 : Spécifier une solution digitale

## 1.1 - Énumérez toutes les fonctionnalités attendues par le client sous forme d’User Story (agilité)

Front-end :

* En tant qu'administrateur, je veux pouvoir m'authentifier sur l'application web pour accéder à des fonctionnalités d'administration.
* En tant qu'administrateur, je veux pouvoir créer un nouveau produit en spécifiant son libellé, sa description, son prix, son image et sa catégorie.
* En tant qu'administrateur, je veux pouvoir mettre en promotion un produit en spécifiant une date de début, une date de fin et un pourcentage de remise, avec calcul automatique du nouveau prix en gras et en rouge dans la liste des produits.
* En tant qu'utilisateur, je veux pouvoir visualiser la liste de tous les produits disponibles sur l'application, y compris ceux sans promotion, avec la possibilité de les filtrer par catégorie.

Back-end :

* En tant qu'administrateur, je veux pouvoir m'authentifier en fournissant des identifiants valides pour accéder aux fonctionnalités d'administration.
* En tant qu'administrateur, je veux pouvoir créer un nouveau produit en fournissant toutes les informations nécessaires et en l'ajoutant à la base de données.
* En tant qu'administrateur, je veux pouvoir mettre en promotion un produit en spécifiant une date de début, une date de fin et un pourcentage de remise, et mettre à jour les données du produit en conséquence.
* En tant qu'administrateur, je veux que les données des produits enregistrés dans la base de données soient cohérentes et précises.

Base de données :

* En tant que système, je veux pouvoir stocker toutes les informations relatives aux produits, y compris leur libellé, leur description, leur prix, leur image, leur catégorie ainsi que les informations relatives à une éventuelle promotion.

## 1.2 - Quels sont les éléments que vous allez sécuriser ? Comment allez-vous procéder ?

Les éléments à sécuriser dans cette application web de Mercadona sont les suivants :

1 - L'authentification des administrateurs : il est important de sécuriser le processus d'authentification pour que seuls les administrateurs autorisés puissent accéder aux fonctionnalités d'administration.

2 - Les données sensibles des utilisateurs : les informations personnelles des administrateurs et des utilisateurs doivent être stockées de manière sécurisée. Il est essentiel d'utiliser des techniques de chiffrement appropriées pour protéger les données sensibles, telles que les mots de passe.

3 - L'accès aux fonctionnalités d'administration : les fonctionnalités d'administration doivent être accessibles uniquement aux administrateurs authentifiés. Il est recommandé de mettre en place des contrôles d'autorisation pour s'assurer que seuls les utilisateurs autorisés ont accès aux fonctionnalités d'administration.

4 - La protection contre les attaques de sécurité : l'application web doit être protégée contre les attaques courantes telles que les injections SQL, les attaques XSS (Cross-Site Scripting) et les attaques CSRF (Cross-Site Request Forgery). Cela peut être réalisé en appliquant des pratiques de codage sécurisées, en validant et en filtrant les données d'entrée, et en utilisant des mécanismes de protection tels que les jetons CSRF et les requêtes préparées.

La sécurisation de ces éléments peut être réalisée des manières suivantes :

1 - Utiliser des bibliothèques et des frameworks sécurisés : il faut utiliser uniquement des bibliothèques et des frameworks bien établis qui sont régulièrement mis à jour avec des correctifs de sécurité.

2 - Appliquer les principes du moindre privilège : il doit être accordé des privilèges d'accès et des autorisations uniquement lorsque cela est nécessaire. Les utilisateurs et les administrateurs doivent avoir les privilèges appropriés pour effectuer leurs tâches, mais pas plus.

3 - Chiffrer les données sensibles : des algorithmes de chiffrement solides existent pour protéger les données sensibles, en particulier les mots de passe des utilisateurs. Les mots de passe doivent être hachés et salés lors du stockage dans la base de données et non en texte brut.

4 - Valider et filtrer les données d'entrée : toujours effectuer une validation stricte des données d'entrée pour prévenir les attaques d'injection et de script malveillant. Il faut aussi utiliser des fonctions de validation et d'échappement appropriées pour filtrer les données avant de les stocker.

## 1.3 - Évoquez les choix techniques que vous avez choisis concernant votre application et justifiez-les (tout en faisant référence au besoin client)

1 - Langage de programmation : Java

Java est un langage de programmation populaire et bien établi, largement utilisé dans le développement d'applications web. Il offre une grande stabilité, une vaste communauté de développeurs, ainsi qu'un large éventail de bibliothèques et de frameworks pour faciliter le développement. Java est également connu pour sa sécurité, ce qui est un aspect crucial pour l'application de Mercadona, qui doit protéger les données des utilisateurs et éviter les vulnérabilités.

2 - Framework Web : Spring Boot

Spring Boot est un framework Java qui simplifie le développement d'applications web en fournissant des fonctionnalités prêtes à l'emploi et en encourageant les bonnes pratiques de développement. Il offre des fonctionnalités telles que l'injection de dépendances, la configuration automatique, la sécurité intégrée, la gestion des sessions, la gestion des erreurs, etc. Spring Boot facilite également l'intégration avec d'autres technologies et services, ce qui est bénéfique pour le développement de l'application de Mercadona.

3 - Base de données : PostgreSQL

PostgreSQL est un système de gestion de base de données relationnelle robuste et open source. Il est réputé pour sa fiabilité, sa performance et sa capacité à gérer des volumes de données importants. PostgreSQL offre également des fonctionnalités avancées de sécurité, telles que le chiffrement des données, les connexions sécurisées et la gestion fine des autorisations. Étant donné que l'application de Mercadona stocke des informations sensibles, PostgreSQL est un choix approprié pour garantir la sécurité et l'intégrité des données.

4 - Interface utilisateur : HTML, CSS et JavaScript

Pour l'interface utilisateur de l'application web, l'utilisation des technologies standard du web, telles que HTML, CSS et JavaScript, est recommandée. Ces technologies offrent une compatibilité élevée avec les navigateurs et permettent de créer une interface conviviale et réactive pour les utilisateurs. L'utilisation de bibliothèques et de frameworks JavaScript, tels que React ou Angular, peut également faciliter le développement de l'interface utilisateur interactive et dynamique.

# Dossier 2 : Développement de la solution

## 2.1 - Dossier de Spécification Technique

Dossier de Spécification Technique - Application Web de Promotions pour Mercadona

1 - Introduction

Le présent dossier de spécification technique décrit les exigences et les fonctionnalités de l'application web de promotions pour Mercadona. L'objectif de cette application est de permettre aux utilisateurs de consulter les promotions en cours et aux administrateurs de gérer les produits et de créer des promotions.

2 - Analyse du projet

2.1 - Objectifs

Les objectifs principaux de l'application web de promotions sont les suivants :

* Permettre aux utilisateurs de consulter les promotions disponibles.
* Autoriser les administrateurs à créer des produits, à les mettre en promotion et à gérer les informations liées.
* Assurer une authentification pour les administrateurs.

2.2 - Contraintes

* L'application doit être développée en utilisant Java pour la partie back-end.
* La base de données utilisée sera PostgreSQL.

3 - Développement de l'application

3.1 - Front-end

Le front-end de l'application sera responsable de l'interface utilisateur et des fonctionnalités accessibles aux utilisateurs finaux. Il devrait inclure les éléments suivants :

* Page d’accueil (catalogue) : Affiche tous les produits disponibles avec un filtre de catégorie.
* Pages produits et promotions : Respectivement ajout et modification des produits.
* Authentification (Spring Security) : Permet aux administrateurs de s'authentifier pour accéder aux fonctionnalités d'administration.

Les technologies les plus appropriées pour le front-end peuvent inclure :

* HTML/CSS pour la structure et la mise en page.
* JavaScript pour la logique interactive.

3.2 - Back-end

Le back-end de l'application sera responsable de la gestion des données, de la logique métier et de la communication avec la base de données. Il devrait inclure les éléments suivants :

* API RESTful : Fournit des endpoints pour la création, la mise à jour et la récupération des données des produits et des promotions.
* Gestion de l'authentification : Vérifie les informations d'identification des administrateurs.

Les technologies les plus appropriées pour le back-end peuvent inclure :

* Java pour le développement de l'API RESTful.
* Frameworks tels que Spring Boot ou JAX-RS pour faciliter le développement et la gestion de l'API.

3.3 - Base de données

La base de données stockera toutes les informations relatives aux produits et leur promotion éventuelle. Elle devrait inclure la table products pour ces produits.

La technologie la plus appropriée pour la base de données est PostgreSQL, en raison de ses capacités relationnelles et de sa compatibilité avec Java.

4 - Évolutions futures

Le développement de l'application web de promotions pour Mercadona peut être amélioré et étendu dans le futur. Quelques suggestions pour les évolutions futures sont les suivantes :

* Implémentation d'une vue détaillée des produits pour afficher des informations supplémentaires.
* Ajout de fonctionnalités de recherche pour permettre aux utilisateurs de trouver des produits spécifiques.
* Mise en place d'une gestion des utilisateurs pour permettre à plusieurs administrateurs d'accéder à l'application.

5 - Tâches du projet et estimations de temps

Les tâches principales du projet et leurs estimations de temps sont les suivantes :

* Analyse des besoins et rédaction de la spécification technique : 2 semaines
* Conception du modèle de données : 1 semaine
* Développement du front-end de l'application : 4 semaines
* Développement du back-end de l'application : 6 semaines
* Mise en place de la base de données : 1 semaine
* Tests et débogage : 2 semaines
* Déploiement et préparation pour la mise en production : 1 semaine

## 2.2 – Tableau Kanban (Trello)

1 - Analyse des besoins et rédaction de la spécification technique :

* Description : Analyser les exigences du projet et rédiger un document de spécification technique détaillé.
* Étiquettes : Analyse, Spécification technique
* Date d'échéance : 01/07/23 – 14/07/23

2 - Conception du modèle de données :

* Description : Concevoir le modèle de données pour la base de données en définissant les tables, les relations et les attributs nécessaires.
* Étiquettes : Conception, Modèle de données
* Date d'échéance : 15/07/23 – 21/07/23

3 - Développement du front-end de l'application :

* Description : Développer l'interface utilisateur et les fonctionnalités du front-end de l'application.
* Étiquettes : Front-end, Développement
* Date d'échéance : 22/07/23 – 18/08/23

4 - Développement du back-end de l'application :

* Description : Développer la logique métier, les API RESTful et la communication avec la base de données pour le back-end de l'application.
* Étiquettes : Back-end, Développement
* Date d'échéance : 19/08/23 – 29/09/23

5 - Mise en place de la base de données :

* Description : Configurer et mettre en place la base de données PostgreSQL.
* Étiquettes : Base de données, Configuration
* Date d'échéance : 30/09/23 – 06/10/23

6 - Tests et débogage :

* Description : Effectuer des tests pour vérifier le bon fonctionnement de l'application et résoudre les problèmes éventuels.
* Étiquettes : Tests, Débogage
* Date d'échéance : 07/10/23 – 20/10/23

7 - Déploiement et préparation pour la mise en production :

* Description : Préparer l'application pour le déploiement en production, y compris les configurations et les tests finaux.
* Étiquettes : Déploiement, Production
* Date d'échéance : 21/10/23 – 27/10/23

Lien tableau Kanban (via Trello, tableau Public pour accessibilité) : <https://trello.com/b/yvke2H8n/mercadona/>

## 2.3 – MCD

Une image contenant texte, capture d’écran, Police, nombre

Description générée automatiquement

|  |
| --- |
| id: Cette colonne est définie comme une séquence (SERIAL) et servira de clé primaire (PRIMARY KEY). Elle est généralement utilisée comme identifiant unique pour chaque produit dans la table.  label: Cette colonne est de type VARCHAR(255) et stocke le libellé ou le nom du produit. Elle ne peut pas être nulle (NOT NULL), ce qui signifie que chaque produit doit avoir un libellé.  description: Cette colonne est de type TEXT et stocke la description du produit. Elle ne peut pas être nulle, ce qui signifie que chaque produit doit avoir une description.  price: Cette colonne est de type DECIMAL(10, 2) et stocke le prix du produit avec une précision de deux décimales. Elle ne peut pas être nulle.  image: Cette colonne est de type VARCHAR(255) et stocke le lien ou le chemin vers l'image du produit. Elle ne peut pas être nulle.  category: Cette colonne est de type VARCHAR(255) et stocke la catégorie à laquelle appartient le produit. Elle ne peut pas être nulle.  promotion\_start\_date: Cette colonne est de type DATE et stocke la date de début de la promotion du produit. Elle peut être nulle, ce qui signifie que certains produits peuvent ne pas avoir de promotion.  promotion\_end\_date: Cette colonne est également de type DATE et stocke la date de fin de la promotion du produit. Elle peut être nulle pour les produits sans promotion.  promotion\_discount: Cette colonne est de type DECIMAL(5, 2) et stocke le pourcentage de remise appliqué au produit pendant la promotion. Elle peut être nulle pour les produits sans promotion. |

## 2.5 – Développement

Étape 1 : Environnement et création de l’infrastructure

Le choix s’est porté sur un serveur dédié virtuel (VPS) chez OVH basé sur l’OS CentOS 7 :

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

La configuration est la suivante : Vcores 1, RAM 2Go, stockage 20 Go.

Une image contenant texte, capture d’écran, Police, nombre

Description générée automatiquement

Un disque additionnel de 50 Go et l’option Snapshot ont été ajoutés :

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

Sur cet environnement de production a été mise en place une infrastructure robuste pour la création et le déploiement d'une API performante. Ce serveur est composé de trois systèmes de fichiers essentiels, chacun avec un rôle spécifique :

|  |
| --- |
| 1. /soft/postgres/data 2. /soft/apache/data 3. /soft/tomcat/data |

Chacun de ces répertoires abrite un composant clé de notre architecture :

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

Le répertoire /soft/postgres/data est dédié à PostgreSQL, un système de gestion de base de données relationnelles open-source. Il s'agit de la pierre angulaire de l’infrastructure, stockant et gérant les données de manière efficace pour alimenter notre API.

Dans le répertoire /soft/apache/data, un serveur web Apache a été installé, une pièce maîtresse pour la gestion des requêtes HTTP et HTTPS. Apache joue un rôle crucial en tant que passerelle entre l'utilisateur final et les services sous-jacents, contribuant ainsi à la disponibilité et à la sécurité de l’API.

Le répertoire /soft/tomcat/data accueille Tomcat, un serveur d'applications Java puissant. Tomcat est le moteur qui exécute l’API, développée en Java. Cette combinaison offre une flexibilité exceptionnelle et une performance optimale pour les applications.

Voici la procédure en image pour la mise en place de cette architecture :

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

Une image contenant texte, capture d’écran, Police, nombre

Description générée automatiquement

Une image contenant texte, capture d’écran, Police, conception

Description générée automatiquement

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

Une image contenant texte, capture d’écran, Police, logiciel

Description générée automatiquement

Une image contenant texte, capture d’écran, logiciel

Description générée automatiquement

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

Une image contenant texte, capture d’écran

Description générée automatiquement

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

Une image contenant texte, capture d’écran

Description générée automatiquement

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

Une image contenant texte, capture d’écran, logiciel, Police

Description générée automatiquement

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

Une image contenant texte, capture d’écran, Police, logiciel

Description générée automatiquement

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

Une image contenant texte, capture d’écran, logiciel, Logiciel multimédia

Description générée automatiquement

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

Une image contenant texte, capture d’écran, conception

Description générée automatiquement

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

Une image contenant texte, capture d’écran, logiciel

Description générée automatiquement

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

Une image contenant texte, capture d’écran, Police, logiciel

Description générée automatiquement

Une image contenant texte, capture d’écran, logiciel, Page web

Description générée automatiquement

Étape 2 : Création d'une application JAVA test avec Spring Boot pour tester les composants postegreSQL, apache et tomcat installés sur le serveur dédié virtuel

Un nouveau projet Spring Boot nommé Studi.zip avec les dépendances requises a été créé en utilisant Spring Initializr, un outil en ligne qui génère la structure de base d'un projet Spring Boot : <https://start.spring.io/>

Une image contenant texte, capture d’écran, logiciel, Page web

Description générée automatiquement

Une classe principale appelée StudiApplication.java a été créée permettant l’ajout d’un user sur une base postgreSQL :

Une image contenant texte, capture d’écran, nombre, Police

Description générée automatiquement

|  |
| --- |
| package vps9d5e18f7.vps.ovh.net.studi;  import org.springframework.boot.SpringApplication;  import org.springframework.boot.autoconfigure.SpringBootApplication;  import org.springframework.boot.web.servlet.support.SpringBootServletInitializer;  import org.springframework.web.bind.annotation.\*;  import java.util.List;  import java.util.Map;  import org.springframework.beans.factory.annotation.Autowired;  import org.springframework.jdbc.core.JdbcTemplate;  import org.springframework.jdbc.support.GeneratedKeyHolder;  import org.springframework.jdbc.support.KeyHolder;  import java.sql.Connection;  import java.sql.PreparedStatement;  import java.sql.Statement;  @SpringBootApplication  public class StudiApplication extends SpringBootServletInitializer {  public static void main(String[] args) {  SpringApplication.run(StudiApplication.class, args);  }  }  @RestController  @RequestMapping("/api/users")  class UserController {  private final JdbcTemplate jdbcTemplate;  @Autowired  public UserController(JdbcTemplate jdbcTemplate) {  this.jdbcTemplate = jdbcTemplate;  }  @PostMapping  public User addUser(@RequestBody User user) {  String sql = "INSERT INTO users (name, email) VALUES (?, ?)";  KeyHolder keyHolder = new GeneratedKeyHolder();  jdbcTemplate.update((Connection con) -> {  PreparedStatement ps = con.prepareStatement(sql, Statement.RETURN\_GENERATED\_KEYS);  ps.setString(1, user.getName());  ps.setString(2, user.getEmail());  return ps;  }, keyHolder);  List<Map<String, Object>> keyList = keyHolder.getKeyList();  if (keyList != null && !keyList.isEmpty()) {  Map<String, Object> keyMap = keyList.get(0);  Integer generatedId = (Integer) keyMap.get("id");  user.setId(generatedId.longValue());  }  return user;  }  @GetMapping  public List<User> getUsers() {  String sql = "SELECT \* FROM users";  return jdbcTemplate.query(sql, (rs, rowNum) -> {  User user = new User();  user.setId(rs.getLong("id"));  user.setName(rs.getString("name"));  user.setEmail(rs.getString("email"));  return user;  });  }  }  class User {  private Long id;  private String name;  private String email;  // getters and setters  public Long getId() {  return id;  }  public void setId(Long id) {  this.id = id;  }  public String getName() {  return name;  }  public void setName(String name) {  this.name = name;  }  public String getEmail() {  return email;  }  public void setEmail(String email) {  this.email = email;  }  } |

Une autre classe appelée CorsConfig.java a été créée pour configurer la politique CORS de l'application Spring Boot afin de permettre à n'importe quelle origine d'accéder aux ressources de l'API tout en autorisant certaines méthodes HTTP et en exposant l'en-tête "Access-Control-Allow-Origin" dans les réponses. Cela facilite l'accès à l'API depuis des domaines différents, ce qui est souvent nécessaire pour les applications web modernes qui communiquent avec des clients web sur des domaines différents :

Une image contenant texte, capture d’écran, logiciel, nombre

Description générée automatiquement

|  |
| --- |
| package vps9d5e18f7.vps.ovh.net.studi;  import org.springframework.context.annotation.Configuration;  import org.springframework.web.servlet.config.annotation.CorsRegistry;  import org.springframework.web.servlet.config.annotation.EnableWebMvc;  import org.springframework.web.servlet.config.annotation.WebMvcConfigurer;  @Configuration  @EnableWebMvc  public class CorsConfig implements WebMvcConfigurer {  @Override  public void addCorsMappings(CorsRegistry registry) {  registry.addMapping("/api/\*\*")  .allowedOrigins("\*")  .allowedMethods("GET", "POST", "PUT", "DELETE")  .allowedHeaders("\*")  .exposedHeaders("Access-Control-Allow-Origin"); // Ajoutez cette ligne  }  } |

Le fichier application.properties a été modifié pour faire la connexion à une base postgreSQL installé en local :

Une image contenant texte, capture d’écran, logiciel, Icône d’ordinateur

Description générée automatiquement

|  |
| --- |
| spring.datasource.url=jdbc:postgresql://localhost:5432/studi  spring.datasource.username=postgres  spring.datasource.password=postgres  spring.datasource.platform=postgres  spring.jpa.hibernate.ddl-auto=none |

Enfin le fichier pom.xml a été modifié pour pouvoir générer un .war et non pas un .jar pour ensuite le déployer sur le tomcat :

Une image contenant texte, capture d’écran, logiciel, affichage

Description générée automatiquement

Une image contenant texte, capture d’écran, logiciel, affichage

Description générée automatiquement

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?>  <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">  <modelVersion>4.0.0</modelVersion>  <parent>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-parent</artifactId>  <version>3.1.3</version>  <relativePath/> <!-- lookup parent from repository -->  </parent>  <groupId>vps-9d5e18f7.vps.ovh.net</groupId>  <artifactId>studi</artifactId>  <version>0.0.1-SNAPSHOT</version>  <packaging>war</packaging>  <name>studi</name>  <description>Application project with Spring Boot</description>  <properties>  <java.version>17</java.version>  </properties>  <dependencies>  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-data-jdbc</artifactId>  </dependency>  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-data-jpa</artifactId>  </dependency>  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-jdbc</artifactId>  </dependency>  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-web</artifactId>  </dependency>  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-devtools</artifactId>  <scope>runtime</scope>  <optional>true</optional>  </dependency>  <dependency>  <groupId>org.postgresql</groupId>  <artifactId>postgresql</artifactId>  <scope>runtime</scope>  </dependency>  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-configuration-processor</artifactId>  <optional>true</optional>  </dependency>  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-test</artifactId>  <scope>test</scope>  </dependency>  <dependency>  <groupId>javax.persistence</groupId>  <artifactId>javax.persistence-api</artifactId>  <version>2.2</version> <!-- Ou une version appropriée -->  </dependency>  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-tomcat</artifactId>  <scope>provided</scope>  </dependency>  </dependencies>  <build>  <plugins>  <plugin>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-maven-plugin</artifactId>  </plugin>  <plugin>  <groupId>org.apache.maven.plugins</groupId>  <artifactId>maven-war-plugin</artifactId>  <configuration>  <failOnMissingWebXml>false</failOnMissingWebXml>  </configuration>  </plugin>  </plugins>  </build>  </project> |

Une base PostgreSQL studi a ensuite été créée via pgadmin avec sa table users :

Une image contenant texte, capture d’écran, logiciel, Icône d’ordinateur

Description générée automatiquement

Une image contenant texte, capture d’écran, logiciel, affichage

Description générée automatiquement

|  |
| --- |
| CREATE DATABASE studi;  CREATE TABLE users (  id SERIAL PRIMARY KEY,  name VARCHAR(255) NOT NULL,  email VARCHAR(255) NOT NULL  ); |

La base et la table étant créées, le projet peut ensuite être lancé avec l’IDE Eclipse pour vérifier si l’API communique bien avec la base et si le projet se lance correctement :

Une image contenant texte, Appareils électroniques, capture d’écran, nombre

Description générée automatiquement

Un script HTML index.html et Javascript script.js sont ensuite créés pour tester l’ajout des users dans la base via l’API :

Une image contenant texte, capture d’écran, affichage, Police

Description générée automatiquement

Une image contenant texte, capture d’écran, affichage, logiciel

Description générée automatiquement

Une image contenant texte, capture d’écran, logiciel, Icône d’ordinateur

Description générée automatiquement

|  |
| --- |
| <!DOCTYPE html>  <html>  <head>  <title>Ajouter un utilisateur</title>  </head>  <body>  <h1>Ajouter un utilisateur</h1>  <form id="userForm">  <label for="name">Nom :</label>  <input type="text" id="name" name="name" required><br><br>  <label for="email">Email :</label>  <input type="email" id="email" name="email" required><br><br>  <button type="submit">Ajouter</button>  </form>  <div id="message"></div>  <script src="script.js"></script>  </body>  </html> |

|  |
| --- |
| document.getElementById("userForm").addEventListener("submit", function (e) {  e.preventDefault();  const name = document.getElementById("name").value;  const email = document.getElementById("email").value;  const user = {  name: name,  email: email  };  // Envoie les données de l'utilisateur au serveur via une requête AJAX  fetch("http://localhost:8080/api/users", {  method: "POST",  headers: {  "Content-Type": "application/json"  },  body: JSON.stringify(user)  })  .then(response => response.json())  .then(data => {  document.getElementById("message").textContent = "Utilisateur ajouté avec succès.";  // Réinitialise les champs du formulaire  document.getElementById("name").value = "";  document.getElementById("email").value = "";  })  .catch(error => {  console.error("Erreur lors de l'ajout de l'utilisateur :", error);  document.getElementById("message").textContent = "Erreur lors de l'ajout de l'utilisateur.";  });  }); |

Maintenant que l’on sait que le projet fonctionne, il nous faut l’exporter au format .war pour le déployer ensuite sur le tomcat installé précedemment sur le serveur dédié virtuel OVH - en n’oubliant pas de changer tous les localhost par l’adresse IP du serveur 92.222.181.53 et le chemin pour l’API <http://92.222.181.53:8080/studi-0.0.1-SNAPSHOT/api/users> avant export. Le chemin de l’API étant un choix personnel :

Une image contenant texte, capture d’écran, logiciel, affichage

Description générée automatiquement

Une image contenant texte, logiciel, Icône d’ordinateur, nombre

Description générée automatiquement

Une image contenant texte, capture d’écran, logiciel, Icône d’ordinateur

Description générée automatiquement

Sur le serveur dédié virtuel OVH où est installé la base postgreSQL, est créé la base studi comme précédemment :

Une image contenant texte, capture d’écran, logiciel, Police

Description générée automatiquement

Le fichier .war est ensuite déployer sur le tomcat du serveur dédié virtuel OVH :

Une image contenant texte, capture d’écran, logiciel, Icône d’ordinateur

Description générée automatiquement

Une image contenant texte, capture d’écran, logiciel, affichage

Description générée automatiquement

Une image contenant texte, capture d’écran

Description générée automatiquement

Les scripts HTML et Javascript eux sont déployés sur le serveur apache installé sur le serveur dédié virtuel OVH :

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

La création d’un user peut alors être testé sur le lien : <http://92.222.181.53:80/index.html>

Une image contenant texte, capture d’écran, Police, logiciel

Description générée automatiquement

Une image contenant texte, capture d’écran, Police, logiciel

Description générée automatiquement

Les données renvoyées par l’API - récupérées depuis la base postgreSQL - peuvent être vu sur le lien : <http://92.222.181.53:8080/studi-0.0.1-SNAPSHOT/api/users>

Une image contenant texte, capture d’écran, logiciel, Police

Description générée automatiquement

Une image contenant texte, capture d’écran, Police, logiciel

Description générée automatiquement

C’est ainsi que peut être validé la mise en place de l’infrastructure de l’environnement de production.

Étape 3 : Développement et déploiement de l’application Mercadona

fff

Lien GitHub : <https://github.com/ovnie/mercadona>