# REPUBLIQUE DU BENIN

*********


**UNIVERSITE NATIONALE DES SCIENCES TECHNOLOGIQUES, INGENIUERIES ET MATHEMATIQUE**

**FACULTE DES SCIENCES ET TECHNIQUES**


**FILIERE : MI3** **GROUPE : 6**

**PROJET**

# GESTION D'UNE LISTE DE CONTACTS

**MEMBRES DU GROUPES**

1- **LOKO Oladé Ange Sèivè**
2- **OUSMANE Salia**
3- **OLOU Agué Martin**
4- **ISSIFOU Abdou kabirou**
5- **DONOUGBO Ignace**
6- **LOKOSSOU Césard Festus**
7- **ABALLO CHOUTA Kondé Aballo Ezékiel**
8- **SERO Tamou**

**SOUS LA SUPERVISION DE:**

**Dr Gilbert CAPO CHICHI**

# PLAN DU PROJET

## Introduction

## I- Analyse des besoins

1- Identification des parties prenantes
2- Cahier des charges
3- Modélisation du système

## II- Conception de l'application

1- Architecture logicielle
2- Choix des technologies et outils
3- Schéma de la base des données

## III- Développement de l'application

1- Implémentation des fonctionalities principales
2- Gestion des erreurs et exceptions
3- Tests et validation

## IV- Déploiement et maintenance

1- Stratégie de déploiement
2- Plan de maintenance et mises à jour

## V- Conclusion et perspectives

1- Bilan du projet
2- Améliorations et évolutions possibles

# Introduction

Dans un monde de plus en plus connecté, la gestion efficace des contacts personnels et professionnels est essentielle. La capacité de retrouver rapidement les informations d'une personne, de communiquer efficacement et d'organiser les relations est cruciale tant dans la vie professionnelle que personnelle.

Traditionnellement, les contacts étaient conservés dans des carnets ou sur des cartes, mais avec l'avènement des technologies numériques, il est devenu nécessaire de disposer d'outils plus adaptés à la gestion des informations. Les entreprises, par exemple, doivent gérer un grand nombre de contacts, incluant des clients, des fournisseurs et des partenaires, ce qui rend difficile la tâche sans un système structuré. Par ailleurs, la multiplication des appareils mobiles et des plateformes de communication nécessite une solution flexible et accessible. Le besoin d'une application de gestion de contacts se justifie par plusieurs facteurs :

Avoir tous les contacts centralisés permet une meilleure organisation et facilite la recherche d'informations. Une application numérique accessible depuis divers appareils garantit que les utilisateurs peuvent rapidement retrouver un contact où qu'ils soient. Les informations des contacts peuvent changer (nouvelles adresses, numéros de téléphone, etc.). Une application permet d'actualiser facilement ces données en quelques clics. Une application peut également intégrer des fonctionnalités pour suivre les interactions avec les contacts, permettant d'améliorer les relations professionnelles et personnelles. Stocker les contacts dans une application offre un niveau de sécurité supérieur par rapport à un carnet papier, notamment grâce à des options de sauvegarde et de chiffrement.

Elle permet de gagner du temps et d'augmenter la productivité, en facilitant l'accès rapide aux informations nécessaires. Les entreprises peuvent mieux gérer leurs relations et prendre des décisions éclairées basées sur des données actualisées et facilement accessibles. Une telle application peut servir de base pour des développements futurs, comme l'intégration d'une fonctionnalité de messagerie .

# II- ANALYSE DES BESOINS

## 1- Identification des parties prenantes

### → Utilisateurs finaux

Profil: Personnes utilisant l'application pour gérer leurs contacts personnels ou professionnels.

Besoins:

-Ajouter, modifier, et supprimer des contacts facilement.

- Rechercher des contacts par divers critères (nom, numéro, etc.).

- Afficher les détails des contacts de manière claire et organisée.

- Accéder à la liste depuis des appareils variés (ordinateurs, smartphones).

- Synchroniser les contacts avec d'autres applications (calendriers, e-mails).

### →Administrateurs

Profil: Personnes responsables de la gestion et de la maintenance de l'application.

Besoins:

- Gérer les comptes utilisateurs (création, modification, suppression).

- Accéder à des statistiques d'utilisation (nombre d'utilisateurs, contacts ajoutés).

- Assurer la sécurité des données (sauvegardes, gestion des accès).

- Mettre à jour l'application pour ajouter de nouvelles fonctionnalités ou corriger des bugs.

### → Développeurs

* Profil: Équipe technique responsable de la conception et du développement de l'application.

* Besoins:

  - Créer une architecture logicielle robuste et modulaire pour faciliter les mises à jour.

  - Produire une interface utilisateur intuitive pour améliorer l'expérience utilisateur.

  - Intégrer une base de données sécurisée pour le stockage des contacts.

  - Assurer la compatibilité de l'application sur différentes plateformes (web, mobile).

  - Mettre en place des tests pour garantir le bon fonctionnement de l'application.

# 2- Cahier des charges

### → Contexte et objectifs

Développer une application permettant aux utilisateurs de gérer efficacement une liste de contacts, incluant des fonctionnalités pour ajouter, modifier, supprimer et rechercher des contacts.

### → **Fonctionnalités principals**:

- Ajouter un nouveau contact (nom, prénom, téléphone, email, ).

- Modifier les informations d'un contact existant.

- Supprimer un contact.

- Afficher la liste des contacts avec des options de tri.

### → Recherche et filtrage:

- Rechercher des contacts par nom, prénom ou numéro de téléphone.

- Filtrer par catégorie (amis, famille, travail).

### → Sécurité:

- Authentification par mot de passe.

- Sauvegarde des données.

## 3. Modélisation du système

- Langages de programmation: Java (backend), HTML/CSS/JavaScript (frontend).

- Base de données: MySQL ou SQLite pour le stockage des contacts.

- Plateforme: Application web accessible depuis différents appareils (ordinateurs, tablettes, smartphones).

# III- CONCEPTION DE L'APPLICATION

## 1 - Architecture logicielle

Choix de l'architecture : MVC (Modèle-Vue-Contrôleur)

Justification du choix

### →Séparation des préoccupations

Le modèle (logique de données) est distinct de la vue (interface utilisateur) et du contrôleur (gestion des interactions), facilitant la compréhension et la maintenance.

### → Facilité de développement

Les développeurs peuvent travailler sur différentes parties de l'application en parallèle, ce qui accélère le processus de développement.

### →Tests simplifieés

Les unités de code peuvent être testées indépendamment, permettant une identification rapide des bugs et une meilleure qualité du code.

### →Scalabilité

Le modèle MVC permet d'ajouter facilement de nouvelles fonctionnalités ou de modifier des existantes sans perturber l'ensemble du système.

### →Amélioration de l'expérience utilisateur

La vue peut être améliorée et modifiée sans affecter la logique métier, permettant d'optimiser constamment l'expérience utilisateur.

# 2 Choix des technologies et outils

## →Langages

- Java: Langage principal pour le développement de l'application, offrant robustesse et portabilité.

- SQL : Utilisé pour l'interaction avec la base de données (récupération, insertion, mise à jour et suppression des contacts).

## →Frameworks

-Spring Boot

- Justification : Simplifie la configuration et le déploiement d'applications Java. Idéal pour créer des API REST et gérer l'architecture MVC.

### → Hibernate

Justification : Framework ORM qui facilite la gestion de la base de données en mappant des objets Java à des tables SQL, simplifiant ainsi les opérations CRUD.

## → Java FX :

Justification : Permet de créer des interfaces graphiques riches si une application de bureau est envisagée. Sinon, pour une application web, cela pourrait être remplacé par des technologies web standards.

## → Environnement de Développement Intégré (IDE)

- Eclipse IDEA:

Justification : IDE performant et riche en fonctionnalités pour Java, avec

# 3- Schéma de la base de données pour la gestion d'une liste de contacts

## → Tables

- Contacts

-Champs :

- `id` (INT, PK, AUTO_INCREMENT) : Identifiant unique du contact.

- `nom` : Nom du contact.

- `prenom` : Prénom du contact.

- `telephone` : Numéro de téléphone.

- `email` : Adresse email.

→ **Relations et contraintes**

**Contraintes :**

- `nom`, `prénom`, `téléphone`, `email` ne peuvent pas être NULL.

- `téléphone` doit être unique (OPTIONNEL selon les besoins).

- `email` peut également être unique (OPTIONNEL selon les besoins

# IV- DÉVELOPPEMENT DE L'APPLICATION

## 1. Implémentation des fonctionnalités principales

### a. Authentification et gestion des utilisateurs

**- Inscription** :

- Formulaire d'inscription avec validation des données.

- Stockage des informations (`username`, `password`, etc.) dans une table `Users`.

- **Connexion** :

- Authentification par mot de passe (hashé pour la sécurité).

- Création de sessions pour les utilisateurs connectés.

**- Gestion des rôles** :

- Implémentation de rôles (administrateur, utilisateur) pour contrôler l'accès aux fonctionnalités.

## b. Interface utilisateur

**- Web** :

- Utilisation de **HTML/CSS/JavaScript** pour créer des pages dynamiques.

- Frameworks : **React** ou **Vue.js** pour construire des interfaces réactives.

- Pages principales : liste des contacts, ajout/modification de contacts, page d'authentification.

- **Desktop** (si nécessaire) :

- Utilisation de **Java FX** pour créer une application de bureau intuitive.

- Concevoir des fenêtres pour la gestion des contacts avec des boutons, formulaires, etc

# 2-Gestion des erreurs et exceptions

## a- Stratégie de gestion des erreurs

**- Exceptions personnalisées** :

- Créer des classes d'exception spécifiques (ex. `ContactNotFoundException`, `UserAlreadyExistsException`) pour gérer des cas particuliers.

**- Middleware de gestion des erreurs :**

- Utiliser un `@ControllerAdvice` dans Spring pour capturer les exceptions globalement et retourner des réponses appropriées (HTTP status et messages).

**- Logs d'erreurs** :

- Utiliser un framework comme **Log4j** ou **SLF4J** pour enregistrer les erreurs et les exceptions, facilitant le débogage.

**- Réponses structurées :**

- Retourner des objets JSON normalisés contenant un code d'erreur, un message, et éventuellement une description pour les erreurs.

## b- Sécurité et validation des entrées

**- Validation côté serveur**

- Utiliser des annotations comme `@NotNull`, `@Email`, et `@Size` dans les modèles pour s'assurer que les données respectent les contraintes.

- Validation côté client :

- Implémenter des vérifications simples en JavaScript

# c- Tests et validation

## - Tests unitaires :

- Utilisation de **JUnit** et **Mockito** pour tester les fonctionnalités principales. Cela comprend la vérification des méthodes de service (ajout, modification, suppression de contacts) et des contrôleurs (authentification, gestion des sessions).

## - Tests d'intégration et fonctionnels :

- Mise en place de tests d'intégration pour s'assurer que les différentes couches de l'application fonctionnent correctement ensemble.

- Tests fonctionnels pour valider les scénarios utilisateur (ex. : création d'un contact, recherche, connexion).

## - Débogage et optimisation du code :

- Utilisation d'outils de profilage pour identifier les goulets d'étranglement et optimiser les performances (ex. : requêtes SQL, gestion de la mémoire).

- Révision du code pour éliminer les bugs et améliorer la lisibilité et la maintenabilité.

Ces étapes garantissent que l'application fonctionne comme prévu, est sécurisée et répond aux besoins des utilisateurs.

# V- DÉPLOIEMENT ET MAINTENANCE

## 1. Stratégie de déploiement

**- Serveur d'hébergement**: Déploiement sur un serveur cloud (par exemple, AWS ou Azure) pour assurer une haute disponibilité.

- **Conteneurisation** : Utilisation de **Docker** pour faciliter le déploiement et la scalabilité de l'application, permettant d'exécuter des instances isolées et reproductibles.

## 2. Plan de maintenance et mises à jour

**- Surveillance des performances** : Mise en place d'outils de monitoring (comme Prometheus) pour suivre l'utilisation des ressources et détecter les anomalies.

**- Gestion des mises à jour** : Élaboration d'un calendrier de mise à jour régulier pour intégrer de nouvelles fonctionnalités et patcher les bugs.

**- Support et documentation** : Création d'une documentation utilisateur et développeur pour faciliter la prise en main et la résolution des problèmes.

Ce plan assure la durabilité de l'application, tout en garantissant que les besoins des utilisateurs sont continuellement satisfaits.

# VI- CONCLUSION ET PERSPECTIVES

## 1. Bilan du projet

**- Résumé des réalisations** : Le projet a abouti à une application robuste de gestion de contacts, intégrant des fonctionnalités d'authentification, de CRUD, et une interface utilisateur intuitive. Les tests complets ont validé la stabilité et la sécurité de l'application.

**- Difficultés rencontrées** : Des défis ont été rencontrés dans la gestion des erreurs et la validation des entrées. Des solutions efficaces ont été mises en place, notamment la création d'exceptions personnalisées et la sécurisation des données.

## 2. Améliorations et évolutions possibles

**- Fonctionnalités futures** : Intégration de fonctionnalités avancées comme la synchronisation avec des services tiers (ex. : Google Contacts) et l'ajout de notifications.

**- Adaptabilité et scalabilité :** L'architecture mise en place (microservices) permettra d'ajouter facilement de nouvelles fonctionnalités et d'augmenter la capacité de l'application selon la demande des utilisateurs.

Ce projet a posé une base solide pour un développement futur continu et l'amélior