République Algérienne democratique et Populaire

Ministère de l'Enseignement Supérieure et de la Recherche Scientifique

Université Mouloud Mammeri de Tizi-Ouzou
Faculté de Génie Electrique et Informatique
Département d'Informatique

Mini projet

Thème

Développement et conception d'une application WEB de déploiement optimal de dispositifs de pompiers en cas de situation d'urgence: feux de forêt .

➢ ***Membres du groupe:***

★MEZRAGUE Lamia

★MEZRAGUE Yasmine

➢ ***Module:*** Assurance qualité logiciel.

➢ ***Spécialité*** : ISI

Introduction Générale:

Dans un monde en perpétuelle mutation, où les données numériques se multiplient de manière exponentielle, les défis de la gestion des situations d'urgence nécessitent des solutions à la fois innovantes et adaptées aux besoins spécifiques des organisations et des populations. **La Business Intelligence** devient un outil indispensable pour les organisations, leur permettant de convertir de vastes quantités de données en informations pertinentes et exploitables pour prendre des décisions éclairées. Grâce aux outils modernes d'**ETL** (Extract, Transform, Load) et aux entrepôts de données, les organisations peuvent centraliser, analyser et offrir une vue d'ensemble cohérente des situations à partir de données hétérogènes. Les bases de données relationnelles sont essentielles pour structurer et gérer ces données, assurant leur intégrité, disponibilité et cohérence, même dans des environnements complexes et dynamiques.

Simultanément, les catastrophes telles que les incendies de forêt, les inondations, ou autres crises majeures posent des défis significatifs aux services de secours, notamment pour la localisation rapide des ressources, l'optimisation des trajets, et l'utilisation en temps réel d'informations provenant de diverses sources, y compris les réseaux sociaux, qui bien que fournissent des données précieuses en période de crise. Pour relever ces défis, le développement d'une application web intégrant des technologies avancées comme les Systèmes d'Information Géographique (SIG), les algorithmes d'optimisation et l'intelligence artificielle (IA), offre de nouvelles perspectives. Une telle solution pourrait révolutionner la gestion des secours, rendant les interventions plus rapides, ciblées et efficaces.

Chapitre 1 : Business intelligence

☐ *1 . Définition du B1*

La Business Intelligence (BI) désigne un ensemble de technologies, de processus et d'outils qui permettent de transformer les données brutes en informations utiles pour les organisations. Elle soutient la prise de décision stratégique et opérationnelle.

❖ *1.1.Objectifs de la BI*

Les objectifs de la BI reflètent ses principaux usages :

Centralisation des données :

- La BI permet de regrouper des informations provenant de diverses sources (bases de données relationnelles, fichiers, etc.) pour fournir une vue unifiée facilitant ainsi une meilleure analyse.

Analyse et compréhension :

- En identifiant des tendances, des modèles , la BI permet de révéler des insights cachés qui ne sont pas toujours apparents dans les ensembles de données bruts.

Aide à la décision :

- Grâce à des analyses basées sur des faits, la BI aide les managers et les dirigeants à prendre des décisions éclairées, réduisant ainsi le risque d'erreurs liées à l'intuition.

Automatisation des rapports :

- La BI minimise le travail manuel en fournissant des tableaux de bord dynamiques et automatisés, ce qui permet de gagner du temps et d'améliorer la précision des rapports.

❖ 1.2.Caractéristiques clés de la BI

Données historiques et en temps réel :

- La BI intègre des analyses pour comprendre les événements passés ainsi que des données en temps réel permettant d'agir sur les situations présentes.

Multidimensionnalité :

- Cette capacité permet d'analyser les données sous plusieurs angles (produit, région, période, etc.), offrant ainsi une vision complète et contextualisée des informations.

Accessibilité :

- La BI propose des rapports et des visualisations faciles à comprendre, même pour les utilisateurs qui n'ont pas de compétences techniques avancées.

❖ *1.3. Exemples d'utilisation*

Ces exemples montrent comment la BI s'applique dans divers secteurs :

- Une entreprise de e-commerce ajuste ses stratégies marketing grâce à l'analyse des ventes mensuelles.
- Un hôpital optimise la gestion de ses ressources en étudiant les temps d'attente des patients.
- Une banque utilise la BI pour détecter des risques ou proposer des produits personnalisés en fonction des comportements des clients.

☐ *2. Technologies d'implémentation de la BI*

L'implémentation de la BI repose sur des technologies qui couvrent tout le cycle de vie des données : collecte, transformation, stockage, analyse et visualisation.

❖ 2.1 Collecte et intégration des données

Ces technologies permettent d'extraire les données de diverses sources pour les intégrer dans un système centralisé :

- **Bases de données relationnelles** :
  - *MySQL* : Bases open-source pour des projets standards.
  - *Microsoft SQL Server* : Intégré à l'écosystème Windows, adapté aux grandes entreprises.
  - *Oracle Database* : Robuste, idéal pour les environnements complexes.
- **Bases NoSQL** :
  - *MongoDB* : Stocke des données semi-structurées.
  - *Cassandra* : Spécialisée dans les big data et le stockage distribué.
  - *Elasticsearch* : Conçue pour des recherches rapides sur des données textuelles.
- **Connecteurs et intégrateurs** :
  - Les API permettent la récupération de données externes.

❖ *2.2 Transformation et intégration des données (ETL/ELT)*

Le processus ETL (Extract, Transform, Load) ou ELT extrait les données brutes, les transforme pour les rendre utilisables, puis les charge dans un entrepôt.

- **Outils ETL/ELT** :
    - *Talend* : Solution open-source pour des projets d'intégration complexes.
    - *Informatica PowerCenter* : Idéal pour les environnements professionnels.
    - *Apache Nifi* : Automatisation des flux de données en open-source.
    - *SSIS (SQL Server Integration Services)* : Intégré à SQL Server pour une compatibilité accrue.
- **Langages et bibliothèques** :
    - *Python* (*pandas*, *PySpark*) : Manipulation de données et traitement à grande échelle.
    - *R* : Utilisé pour des analyses statistiques avancées.

❖ ***2.3 Stockage des données***

Les données sont centralisées dans des entrepôts ou des lacs de données selon leurs caractéristiques.

- **Entrepôts de données (Data Warehouse)** :
  Conçus pour des analyses rapides sur des données structurées. Exemples :
    - *Snowflake*, *Google BigQuery*

      Lacs de données (Data Lake) :
      Adaptés pour les données semi-structurées ou non structurées. Exemples :

    - *Amazon S3*

❖ ***2.4 Analyse et traitement des données***

Ces outils permettent une exploration approfondie et une exécution d'analyses complexes.

- **Systèmes OLAP (Online Analytical Processing)** :
    - *SSAS*, *SAP BW/4HANA*, *Mondrian*.
- **Outils d'analyse Big Data** :
    - *Apache Spark*, *Hive*, *Elasticsearch*.

❖ ***2.5 Visualisation et reporting***

Ces outils transforment les résultats d'analyses en graphiques ou tableaux de bord faciles à comprendre.

- **Outils de visualisation BI** :

    - *Tableau*, *Microsoft Power BI*, *Qlik Sense*, *Looker*, *Google Data Studio*.
- **Bibliothèques de visualisation** :
    - *D3.js* pour des visualisations personnalisées.
    - *Plotly* pour des graphiques interactifs en Python.

☐ ***3.Composants d'un entrepôt de données (Data Warehouse)***

Un entrepôt de données est une base centralisée et optimisée pour l'analyse, le reporting, et la prise de décisions stratégiques. Il regroupe et structure les données provenant de multiples sources, permettant ainsi leur exploitation efficace. Voici une description détaillée de ses principaux composants :

1. Sources de données

L'entrepôt de données centralise des informations issues de diverses sources pour les rendre exploitables. Ces sources incluent :

- **Bases de données transactionnelles** : Systèmes ERP (Enterprise Resource Planning) et CRM (Customer Relationship Management), qui stockent des données relatives aux opérations commerciales quotidiennes.
    - *Exemple : Les ventes quotidiennes d'un magasin ou les interactions clients dans un CRM sont extraites pour analyse.*
- **Fichiers plats** : Formats simples comme CSV ou Excel, souvent utilisés pour transférer des données ponctuelles ou provenant de systèmes non connectés.
    - *Exemple : Une liste de prospects enregistrée dans un fichier Excel est intégrée dans l'entrepôt pour enrichir les analyses marketing.*
- **API** : Interfaces de programmation qui permettent de collecter des données depuis des services web ou des plateformes tierces, comme les réseaux sociaux.
    - *Exemple : Une API Facebook qui fournit des métriques d'engagement sur une campagne publicitaire.*

Ces données hétérogènes alimentent l'entrepôt pour fournir une vision consolidée.

2. Zone de staging (zone de transit)

La zone de staging agit comme une zone tampon où les données brutes provenant des sources sont stockées temporairement avant leur transformation. Cette étape est essentielle pour minimiser l'impact sur les systèmes sources et éviter les perturbations lors de l'extraction des données.

- ***Caractéristiques :***
    - Les données restent dans leur format brut et non traité.
    - Stockage temporaire, souvent sous forme de tables SQL ou de fichiers plats.

- ***Exemple*** **:** Une table temporaire SQL est créée pour stocker les données brutes importées d'un fichier CSV contenant les ventes journalières avant leur traitement.

3. Zone d'intégration

Dans cette étape, les données brutes sont transformées, nettoyées, et harmonisées pour garantir leur qualité et leur cohérence. C'est ici que les données provenant de différentes sources sont consolidées dans une structure uniforme.

- ***Processus clés :***
    - Nettoyage des données : Suppression des valeurs aberrantes ou des doublons.
    - Transformation : Conversion des formats de dates, harmonisation des devises, standardisation des noms de colonnes, etc.
    - Intégration : Fusion des données similaires issues de sources distinctes.
- ***Exemple*** **:** Les dates enregistrées dans différents formats (MM/JJ/AAAA, JJ-MM-AAAA) sont uniformisées en un format unique (AAAA-MM-JJ). Les ventes dans différentes devises sont converties dans une devise commune.

4. Schémas et structures de données

Une fois les données intégrées, elles sont organisées selon des modèles spécifiques adaptés aux besoins d'analyse et de reporting. Les deux modèles les plus courants sont :

- ***Modèle en étoile :***
    - Comprend une table centrale appelée *table de faits*, qui contient les données quantitatives (ex : ventes, revenus).
    - Cette table est reliée à des *tables de dimensions*, qui fournissent un contexte (ex : clients, produits, dates).
    - *Exemple : Une table de faits "ventes" est connectée aux dimensions "produits", "clients", et "temps" pour analyser les ventes selon différents axes.*
- ***Modèle en flocon de neige :***
    - Étend le modèle en étoile en normalisant davantage les tables de dimensions. Chaque dimension est subdivisée en sous-dimensions pour minimiser la redondance.
    - *Exemple : La dimension "produits" est subdivisée en "catégories de produits" et "types de produits" pour une meilleure granularité.*

Ces schémas simplifient l'accès aux données et facilitent les requêtes complexes.

5. Outils d'analyse et de visualisation

Une fois les données organisées et chargées dans l'entrepôt, des outils spécialisés permettent de les exploiter. Ces logiciels sont conçus pour aider les utilisateurs finaux à extraire, explorer, et visualiser les données sans nécessiter une expertise technique approfondie.

- **Caractéristiques :**
    - Extraction et requêtes interactives : Les utilisateurs peuvent interroger les données pour répondre à des besoins spécifiques.
    - Visualisation avancée : Les outils proposent des graphiques, tableaux de bord, et rapports interactifs pour rendre les données compréhensibles.
- **Exemples d'outils populaires :**
    - **Power BI** : Solution Microsoft largement utilisée pour créer des rapports interactifs.
    - **Tableau** : Logiciel performant pour l'analyse visuelle et la création de tableaux de bord.
    - **QlikView** : Plateforme axée sur l'intelligence d'entreprise et l'exploration de données.
    - *Exemple : Un tableau de bord montrant les ventes par région et par catégorie de produit, mis à jour en temps réel.*

☐ *4.Processus ETL (Extract, Transform, Load)*

Le processus **ETL (Extract, Transform, Load)** est un élément fondamental de la Business Intelligence (BI). Il permet de collecter, préparer, et charger les données dans un entrepôt pour les rendre exploitables.

Ses Étapes :

1. Extraction (Extract)

Cette première étape consiste à collecter les données depuis diverses sources, souvent hétérogènes, pour les centraliser dans un environnement commun. L'objectif est de récupérer les données brutes sans perturber les systèmes opérationnels.

- ***Caractéristiques :***
    - Les données peuvent provenir de bases de données transactionnelles, de fichiers plats (CSV, Excel) ou d'API.
    - L'extraction peut être réalisée de manière incrémentale (récupération des nouvelles données uniquement) ou complète (récupération de l'ensemble des données).
- ***Exemple :***
    - Extraire un fichier CSV contenant les ventes quotidiennes d'un magasin ou utiliser une API pour récupérer les métriques d'une campagne publicitaire en ligne.

2. Transformation (Transform):

Après l'extraction, les données brutes sont préparées pour l'analyse. Cette étape consiste à nettoyer, harmoniser et enrichir les données afin qu'elles soient cohérentes et conformes aux besoins de l'entrepôt de données.

- **Principales opérations :**
  - **Nettoyage** : Suppression des doublons, des valeurs aberrantes ou des données manquantes.
  - **Conversion** : Transformation des formats (ex : convertir une date de JJ/MM/AAAA en AAAA-MM-JJ).
  - **Standardisation** : Uniformisation des unités de mesure ou des devises (ex : convertir des montants dans différentes devises en euros).
  - **Agrégation** : Regroupement des données selon des périodes ou des dimensions (ex : calcul des ventes mensuelles à partir des ventes journalières).
  - **Fusion et enrichissement** : Combinaison des données provenant de plusieurs sources pour obtenir une vue complète.
- **Exemples :**
  - Suppression des doublons dans une liste de clients.
  - Conversion des ventes exprimées en dollars et livres sterling en euros.
  - Regroupement des transactions par mois pour obtenir une vue consolidée des performances.

3. Chargement (Load)

La dernière étape consiste à intégrer les données transformées dans l'entrepôt de données. C'est ici que les données sont rendues accessibles pour les analyses et les requêtes.

- **Types de chargement :**
  - **Chargement initial** : Insertion d'un ensemble de données complet dans l'entrepôt pour la première fois.
  - **Chargement incrémental** : Ajout ou mise à jour des nouvelles données afin de maintenir l'entrepôt à jour.
  - **Chargement en temps réel** : Synchronisation continue des données pour répondre aux besoins d'analyses en temps réel.
- **Exemple :**
  - Charger les ventes consolidées dans une table *Faits Ventes*, reliée aux dimensions comme *Produits*, *Clients*, et *Temps*.

4. Composants transversaux

- **Orchestration et Planification :**
  - Gestion des workflows ETL.
  - Ordonnancement des tâches d'extraction, transformation et chargement.
- **Surveillance et Gestion des erreurs :**
  - Suivi des performances.

    - Alertes en cas de problème.
- **Documentation :**
    - Métadonnées pour décrire les données et leur cycle de vie.
- **Sécurité :**
    - Gestion des accès et des autorisations.
    - Chiffrement des données sensibles.

Importance du processus ETL

Le processus ETL garantit que les données sont non seulement collectées mais également transformées en une structure compréhensible et cohérente pour répondre aux besoins d'analyse. C'est grâce à ce processus que les entreprises peuvent tirer des insights fiables et prendre des décisions éclairées, en s'appuyant sur des données de qualité.

☐ *Exemples techniques :*

1. Analyse des ventes dans une entreprise de commerce électronique

Cas pratique : Analyse des performances de produits et segmentation des clients.

★ **Technologies utilisées :**

- **Collecte des données :** Les données des ventes, clics des utilisateurs et comportements d'achat sont extraites depuis une base relationnelle (PostgreSQL) et des fichiers journaux des serveurs.
- **Transformation des données (ETL) :**
    - Avec *Talend* ou *Apache Nifi*, les données sont nettoyées pour corriger les erreurs (par ex: des catégories mal orthographiées).
    - Segmentation des clients basée sur leurs comportements avec *Python* (*pandas*, *scikit-learn* pour le clustering).
- **Stockage :**
    - Les données sont chargées dans *Snowflake* pour permettre des requêtes rapides.
- **Analyse et visualisation :**
    - Création de tableaux de bord interactifs dans *Power BI* pour visualiser les produits les plus performants et analyser les ventes par région et période.
    - Analyse des tendances via des séries temporelles avec *R*.

2. Optimisation logistique pour une entreprise de transport:

***Cas pratique* : Réduction des coûts de carburant et optimisation des itinéraires.**

★ **Technologies utilisées :**

- **Collecte des données :** Les données GPS des véhicules sont collectées en temps réel via des API et des flux MQTT.
- **Transformation des données :**
  - Les données brutes sont enrichies avec des informations météo et des cartes géographiques grâce à des API externes (Google Maps API).
  - Les données sont transformées et agrégées dans un entrepôt avec *Apache Spark*.
- **Analyse et traitement :**
  - Modélisation des itinéraires optimaux avec des algorithmes d'optimisation (algorithme A* ou Dijkstra) en utilisant *Python* ou *Pyomo*.
  - Calcul des KPI tels que le coût moyen par kilomètre et l'efficacité énergétique avec *SQL* dans *BigQuery*.
- **Visualisation :**
  - Tableaux de bord dynamiques dans *Tableau* pour suivre les itinéraires et les performances en temps réel.

3. Détection de fraudes dans une banque

Cas pratique : Identification des transactions suspectes.

Technologies utilisées :

- **Collecte des données :** Les transactions sont enregistrées dans une base relationnelle (Oracle Database).
- **Transformation des données :**
  - Transformation des formats (par ex., convertion des devises) et nettoyage des anomalies via *Informatica PowerCenter*.
  - Extraction de caractéristiques pour le modèle de fraude, comme la fréquence des transactions par carte ou le montant moyen, via *Python*.
- **Analyse des données :**
  - Un modèle de machine learning basé sur des forêts aléatoires (*scikit-learn*) ou des réseaux neuronaux (*TensorFlow*) est entraîné sur les données historiques pour détecter les anomalies.
  - Analyse des transactions suspectes avec des visualisations OLAP dans *Qlik Sense*.
- **Visualisation et reporting :**
  - Alertes automatiques générées et affichées dans un tableau de bord en temps réel dans *Microsoft Power BI*.

4. Optimisation des performances d'un hôpital

Cas pratique : Réduction des temps d'attente aux urgences.

Technologies utilisées :

- **Collecte des données** : Les données patient sont extraites depuis un système ERP médical (SAP HANA).
- **Transformation des données :**
    - Les données sur les temps d'attente sont croisées avec les horaires du personnel et les pics d'admission via *SSIS* ou *Talend*.
- **Analyse et traitement :**
    - Analyse prédictive avec *Python* (modèles de régression pour estimer les pics d'affluence).
    - Agrégation des données par périodes pour identifier les goulets d'étranglement dans *Google BigQuery*.
- **Visualisation :**
    - Un tableau de bord dans *Looker* montre les temps d'attente par heure, la charge du personnel et les prévisions pour la journée.

5. Analyse des campagnes marketing dans une entreprise

Cas pratique : Mesurer le ROI des campagnes numériques.

Technologies utilisées :

- **Collecte des données :**
    - Les clics et conversions des campagnes publicitaires sont récupérés via des connecteurs API (Google Ads, Facebook Ads).
- **Transformation des données :**
    - Les données sont standardisées et consolidées dans un entrepôt de données cloud (*Amazon Redshift*).
- **Analyse et traitement :**
    - Calcul des KPIs tels que le coût par acquisition (CPA) et le retour sur investissement (ROI) à l'aide de *SQL*.
    - Création de modèles pour prévoir l'impact des campagnes futures avec *Python*.
- **Visualisation :**
    - Rapports automatisés avec *Google Data Studio* pour surveiller les performances en temps réel.

Données relationnelles vs Entrepôt de données

Finalité :

- Les **données relationnelles** sont utilisées pour **gérer des transactions** quotidiennes, comme la gestion des paiements bancaires ou le suivi des commandes clients.
- Un **entrepôt de données** est conçu pour **supporter l'analyse stratégique**, en fournissant des informations pour des prises de décision à long terme (comme l'analyse des tendances de ventes ou des performances régionales).

Structure :

- Les **données relationnelles** utilisent des **tables normalisées** pour éviter la redondance et garantir l'intégrité des données. Cela signifie que les données sont divisées en tables distinctes pour minimiser la duplication d'informations.
- Un **entrepôt de données**, en revanche, utilise des **modèles étoile ou flocon**, qui sont **dénormalisés** pour simplifier les requêtes d'analyse et d'agrégation des données.

Volatilité :

- Les **données relationnelles** sont souvent **fréquemment mises à jour** pour refléter les transactions courantes (ex : modifications dans le statut des commandes, enregistrement de paiements).
- Les données dans un **entrepôt de données** sont **historisées**, ce qui signifie qu'elles sont conservées pour l'analyse à long terme et rarement mises à jour une fois qu'elles ont été collectées.

Type de requêtes :

- Les **données relationnelles** traitent des **requêtes simples et rapides**, généralement centrées sur les transactions courantes (ex : vérifier le statut d'une commande ou effectuer une opération bancaire).
- Les **entrepôts de données** traitent des **requêtes complexes**, impliquant des agrégations (calculs de totaux, moyennes) et des analyses historiques pour des rapports stratégiques (ex : analyses des performances par région ou comparaison des ventes sur plusieurs années).

Exemple d'usage :

- Les **données relationnelles** sont utilisées dans des systèmes comme la gestion des **paiements bancaires** ou le **suivi des commandes clients**, où des mises à jour fréquentes et des transactions rapides sont nécessaires.
- Les **entrepôts de données** sont utilisés pour des **analyses des tendances de ventes annuelles** ou des rapports sur les **performances par région**, qui nécessitent des analyses plus complexes basées sur des données historiques.

Cahier des Charges (C.C.)

1. Introduction et Contexte

Les incendies de forêt représentent une menace croissante et nécessitent une réponse rapide et coordonnée des services de secours. Le projet vise à développer une application web permettant d'optimiser le déploiement des ressources des pompiers (véhicules, personnel, ravitaillement, etc.). Cette solution s'appuiera sur des technologies modernes, telles que les systèmes d'information géographique (SIG), et l'exploitation des données.

2. Objectifs du Projet

- Fournir aux citoyens un moyen de signaler un incendie en précisant son type et son degré de gravité.
- Fournir des consignes de sécurité adaptées aux citoyens en fonction de leur emplacement.
- Intégrer une carte géographique permettant aux pompiers d'accéder à l'itinéraire le plus rapide pour atteindre les zones sinistrées.
- permettre au chef pompier de gérer d'une manière efficace les ressources des unités des pompiers : personnels , et véhicules.

3. Besoins Fonctionnels

3.1. Gestion des données spatiales et localisation en temps réel

- Localisation précise des zones à risque, des ressources disponibles et des événements en cours.
- Mise à jour dynamique des cartes en fonction des données collectées (ex : propagation des feux).

3.2. Optimisation du déploiement des ressources

- Utilisation d'algorithmes d'optimisation pour sélectionner les zones prioritaires.
    - Exemple : Algorithmes basés sur les graphes (A*, etc.) .
    - Prise en compte des contraintes de temps, distance et disponibilité des ressources.

3.3. Interfaces utilisateur

- Interface web intuitive pour les pompiers et les responsables des opérations.
    - Tableau de bord interactif affichant les zones touchées, et les ressources déployées.
    - Indicateurs clés : niveaux de danger, état des ressources.
    - Notifications des évolutions critiques (nouveaux feux).

4. Besoins Techniques

4.1. Technologies envisagées

- **SIG** : Outils comme API cartographiques (Google Maps).
- **Algorithmes d'optimisation** : Développement en Python , avec des bibliothèques comme render.
- **Backend** : Frameworks comme Django pour gérer la logique applicative.
- **Frontend**: comme html , css ...
- **Base de données** db browser sqlLite pour des données.

4.2. Architecture du Système

- Application web basée sur une architecture client-serveur.

5. Contraintes et Limitations

- **Temps de réponse** : L'application doit fonctionner en quasi-temps réel pour maximiser son efficacité, en particulier pour la mise à jour des cartes et l'attribution des ressources.
- **Précision des données** : Les données collectées via SIG doivent être fiables pour éviter des erreurs d'intervention.
- **Compatibilité multi-appareils** : L'application doit être accessible à la fois sur ordinateurs et appareils mobiles.
- **Budget et ressources** : Le projet doit être développé avec des outils et technologies accessibles tout en respectant le budget .

6. Critères de Succès

- Réduction du temps de réponse lors des interventions.
- Précision dans l'attribution des ressources aux zones prioritaires.
- Adoption par les utilisateurs grâce à une interface ergonomique et un processus de déploiement fluide.

7. Plan de Réalisation

Phase 1 : Analyse des Besoins (2 semaines)

- Étude des scénarios types de sinistres.
- Collecte des besoins des utilisateurs (pompiers, autorités locales, citoyens).
- Définition des spécifications détaillées.

Phase 2 : Conception et Prototypage (4 semaines)

- Développement du modèle de base de données et intégration SIG.
- Création des interfaces utilisateur (UI/UX).

- Implémentation des premiers algorithmes d'optimisation.

Phase 3 : Développement et Intégration (2 semaines)

- Développement du backend et du frontend.
- Intégration des API externes (cartographiques).
- Implémentation des fonctionnalités de base (cartographie, gestion des alertes, optimisation).

Phase 4 : Tests et Validation (1 semaines)

- Tests fonctionnels et performance en environnement simulé.

Phase 5 : Déploiement (2 semaines)

- Documentation et formation des utilisateurs (pompiers et responsables des opérations).

Phase 6 : Maintenance et Suivi Post-Déploiement (2 semaines)

- Suivi des performances du prototype.
- Ajustements en fonction des retours d'expérience et des problèmes rencontrés.

8. Délais et Livrables

- **Phase 1 : Analyse des besoins** – Livraison : Document d'analyse et spécifications.
- **Phase 2 : Conception et prototypage** – Livraison : Prototype de l'application, maquettes de l'interface utilisateur.
- **Phase 3 : Développement et intégration** – Livraison : Version avec fonctionnalités principales.
- **Phase 4 : Tests et validation** – Livraison : Rapport de tests et validation, corrections des erreurs majeures.
- **Phase 5 : Déploiement** – Livraison : Application en production, documentation et formation.
- **Phase 6 : Maintenance** – Livraison : Suivi et ajustements post-déploiement.

9. Budget et Ressources

9.1. Budget estimé

- **Ressources humaines** :
    - Développeurs (Backend et Frontend), Data Scientist, Designer, Testeurs, Chef de projet.
- **Technologies et licences** :
    - Licences SIG, API externes (cartographiques)..
- **Formation et documentation** :

- Formation des utilisateurs, création de la documentation technique et utilisateur.

9.2. Ressources nécessaires

- **Équipe projet** : 1 Développeur Backend, 1 Développeur Frontend, 1 Data Scientist, 1 Designer UI/UX, 1 Chef de projet, 2 Testeurs.
- **Technologies utilisées** : Django, Sphinx, Google Maps API , UnitTest.

10. Risques et Contingences

- **Retards dans le développement** : Prévoir une marge de 10% du budget pour les imprévus.
- **Problèmes techniques** : En cas de difficulté d'intégration, prévoir des ressources pour des consultations externes.

Conclusion : Ce cahier des charges pose les bases d'une application web permettant d'optimiser la gestion des ressources en situation de crise, en combinant des technologies avancées telles que les SIG .Le projet vise à améliorer l'efficacité et la réactivité des services de secours dans la lutte contre les incendies de forêt.

Objectifs possibles du projet :

1. **Surveillance des feux de forêt** : Collecter des données sur les feux actifs grâce à des API.
2. **Prévention** : Fournir des alertes basées sur des indices de risque.
3. **Sensibilisation** : Visualiser les zones touchées et fournir des informations sur les impacts environnementaux.
4. **Gestion efficace :** permettre au chef pompier de faire une gestion efficace pour les personnels des pompiers et les véhicules .

Étapes principales :

1. **Choisir les API adaptées** :
   - **Google Maps API** : Pour intégrer des cartes interactives.
2. **Développer avec Django** :
   - **Backend** : Créer une base de données pour stocker les données des feux (emplacement, intensité, date, etc.).
   - **Frontend** : Développer une interface utilisateur pour visualiser et interagir avec les données.
3. **Fonctionnalités possibles** :
   - Tableau de bord pour suivre les tendances (historique des feux, prévisions).
   - Un formulaire de déclaration des feux .
   - une carte interactive en sélectionnant la zone d'incendie et en affichant le plus court chemin entre l'unité de pompier et le lieu d'incendie.

L'architecture de l'application en conception globale et détaillée :

L'architecture d'une application décrit la structure globale et la façon dont les différentes parties de l'application interagissent. Elle est souvent présentée sous deux formes : l'architecture globale (ou conception générale) et l'architecture détaillée (ou conception détaillée).

d'architecture globale :

1. Frontend (Vue utilisateur) :

Technologie : HTML, CSS, JavaScript (avec éventuellement des frameworks comme Vue.js, ou bien des templates Django simples).

Rôle : C'est la partie visible de l'application. Les utilisateurs interagissent avec cette interface pour effectuer des actions comme la déclaration des incendies , consulter les consignes de sécurité.. etc. Elle communique avec le backend via des requêtes HTTP.

2. Backend (Logique métier) :

Technologie : Django (Python).

Rôle : Le backend gère la logique de l'application, telles que les alertes des utilisateurs, la gestion des camions, des interactions avec la base de données, etc. Django utilise des vues pour rendre des pages web ou fournir des données via des API pour une communication plus dynamique avec le frontend.

3. Admin (Côté administrateur) :

Technologie : Le panel d'administration par défaut de Django.

Rôle : Cette interface permet aux administrateurs de gérer facilement les déclarants, les pompiers, les véhicules de pompiers, les incendies… etc. Elle est alimentée directement par la base de données et est hautement configurable via le framework Django pour ajouter ou modifier les modèles de données.

3. Conception détaillée

La conception détaillée va se concentrer sur la manière dont chaque composant fonctionnera au niveau technique.

Modèles de données (Database Models) :

Définissez les modèles dans Django pour représenter les entités importantes du système. Par exemple : Incendie : Stocke les informations sur les incendies en cours.

Conception globale de l'architecture de l'application

L'architecture de l'application pour le déploiement optimal des dispositifs de pompiers en situation d'urgence (feux de forêt) repose sur plusieurs entités principales et leur interaction pour permettre une gestion efficace des incidents, des pompiers et des ressources disponibles.

Composants principaux de l'application :

Frontend (Interface utilisateur) :

Objectif : Offrir une interface conviviale pour les citoyens, les pompiers et les administrateurs afin de gérer les incidents, les ressources et les missions.
Technologies : HTML, CSS, JavaScript.
Utilisateurs : Citoyens : Déclarer un incendie, suivre les informations sur l'incendie.
Pompiers : Consulter les missions, suivre l'évolution des incendies, et recevoir des notifications en temps réel.
Administrateurs : Gérer les informations sur les incendies, les pompiers, les véhicules, et les équipes.

Backend (Logique métier et API) :

Objectif : Gérer la logique métier, les interactions avec la base de données et la communication avec le frontend via des API.
Technologies : Django (avec Django Framework), Python.
Utilisateurs : Pompiers et Administrateurs : Reçoivent les informations sur les incendies, les missions et les ressources.
Citoyens : Utilisent l'application pour déclarer un incendie ou consulter des informations publiques.

Base de données (Modèles de données) :

Objectif : Stocker toutes les données nécessaires à l'application, y compris les informations sur les incendies, les pompiers, les véhicules et les missions.
Technologies : DB SQLLITE.
Entités principales : Incendie : Contient des informations sur les incendies en cours, notamment leur emplacement et leur gravité.
UnitePompier : Représente les unités de pompiers disponibles.
VehiculePompier : Gère les informations relatives aux véhicules de pompiers (matricule, type, état).
Citoyen: gérer les déclarants .

Cartographie et gestion des incidents :

Objectif : Visualiser les incendies sur une carte et permettre aux utilisateurs de suivre l'évolution des événements.
Technologies : Google Maps API.

Système de notifications en temps réel :

Objectif : alerter les pompiers des nouveaux incidents et des mises à jour en temps réel.
Technologies : Django ,
Utilisateurs concernés : Pompiers et administrateurs.

Interaction entre les entités :

Citoyen : Les citoyens peuvent déclarer des incendies via la table incendie. Lorsqu'un incident est signalé, il est automatiquement lié à un incendie (table Incendie) dans la base de données, où des informations détaillées telles que l'emplacement et la gravité sont enregistrées.

Pompiers : Les pompiers sont attribués à des missions spécifiques via la table Pompier, avec des informations détaillées sur leur équipe, leur grade et leur mission en cours. Chaque pompier appartient à une unité, qui est enregistrée dans la table UnitePompier.

Véhicules : Les véhicules nécessaires pour les interventions sont enregistrés dans la table VehiculePompier. Chaque véhicule est associé à un type (camion, ambulance, etc.) et un état de disponibilité (en mission ou non).

Missions : Lorsqu'un incendie est signalé, une équipe de pompiers et des véhicules spécifiques sont envoyés pour intervenir. Cette coordination est gérée par la logique métier dans le backend et représente une action dans la base de données.

3. Codage

Le codage est l'implémentation des conceptions détaillées en code fonctionnel. Voici comment procéder :
Étapes pour coder l'application :

Configuration du projet Django :

Créez un projet Django et configurez les fichiers nécessaires (comme settings.py, urls.py, models.py, etc.).

Création des modèles :

Définissez les modèles dans le fichier models.py.

Création des vues :

Créez des fonctions de vue dans le fichier views.py.

Création des templates HTML :

Créez des fichiers HTML dans un dossier templates/

Définition des URLs :

Définissez les routes dans le fichier urls.py .

Tests :

Écrivez des tests unitaires pour vérifier le bon fonctionnement de vos vues et modèles.

Outil de test

Une application web développée avec Python et Django, on utilise généralement unittest, qui est intégré dans Python, ou des frameworks spécialisés comme Pytest.

1. Créer des tests

Dans une application Django, les tests sont définis dans un fichier appelé tests.py à l'intérieur de chaque application.

Types de tests à réaliser

a) Tests unitaires

Ce type de test s'assure que le formulaire fonctionne comme prévu, en vérifiant que les données valides sont acceptées et que les erreurs sont renvoyées pour des entrées incorrectes.
Type de test :
Test unitaire (Unit Test) : Ce type de test vérifie une unité de code spécifique, dans ce cas, le formulaire Django (CitoyenForm). Chaque test s'assure que le formulaire répond comme prévu lorsque des données valides ou invalides sont envoyées.
Chaque test est isolé et vérifie une partie précise du comportement du formulaire : validation de l'email, correspondance des mots de passe, etc.
Test de formulaire :Plus précisément, ce test est un test de formulaire dans Django. Il teste le comportement du formulaire CitoyenForm en s'assurant qu'il valide correctement les champs en fonction des règles définies dans le formulaire.

b) Tests d'intégration

Tester les interactions entre plusieurs composants, comme les vues, les modèles, et les templates.
Objectif :
Ce test vérifie si l'API renvoie correctement les incendies enregistrés dans la base de données. Il s'agit d'une vérification de l'intégration entre la vue de l'API (probablement une vue basée sur une classe ou une fonction) et le modèle de données, pour s'assurer que les données sont correctement récupérées et renvoyées.

c) Tests fonctionnels

Simuler les actions d'un utilisateur sur le site, comme remplir un formulaire ou accéder à une page.
Test de formulaire valide (test_valid_form) :
Objectif : Vérifier que le formulaire est valide lorsque toutes les données sont correctement remplies.
Vérification : Le formulaire doit être valide et, si applicable, un objet Citoyen est créé dans la base de données.
Test d'email invalide (test_invalid_email) :
Objectif : Vérifier que l'email invalide est rejeté par la validation du formulaire.
Vérification : Le formulaire doit être invalide et afficher un message d'erreur pour l'email ("Enter a valid email address.").
Test de non-correspondance des mots de passe (test_password_mismatch) :
Objectif : Vérifier que le formulaire rejette les mots de passe qui ne correspondent pas.
Vérification : Le formulaire doit être invalide et afficher un message d'erreur pour la non-correspondance des mots de passe ("Les mots de passe ne correspondent pas.").

La documentation avec Sphinx:

Sphinx est un générateur de documentation utilisé principalement pour des projets Python. Il permet de créer de la documentation en plusieurs formats (HTML, PDF, LaTeX, etc.) à partir de fichiers sources écrits en **reStructuredText (reST)** ou **Markdown (MD)**. Sphinx est particulièrement populaire pour les projets logiciels, car il peut intégrer automatiquement la documentation du code source grâce à des commentaires et des docstrings.

1. Installer Sphinx et les dépendances

Avant tout, vous devez installer Sphinx ainsi que des extensions utiles pour intégrer Django.

a. Installer Sphinx

Dans votre environnement virtuel ou global, exécutez la commande suivante pour installer

Sphinx :

bash

pip install sphinx

b. Installer sphinx_rtd_theme (optionnel mais recommandé)

Il s'agit d'un thème populaire pour Sphinx qui est bien adapté pour la documentation Django.

bash

pip install sphinx_rtd_theme

c. Installer sphinx-autodoc (pour générer automatiquement la documentation à partir du code)

Cette extension est utilisée pour extraire automatiquement la documentation des docstrings

de votre code source.

bash

pip install sphinx-autodoc

d. Installer django-sphinx (si nécessaire)

Si vous avez besoin de générer de la documentation avec des modèles Django et des vues,

vous pouvez installer cette extension, mais elle est parfois facultative :

bash

pip install django-sphinx

3. Configurer le fichier conf.py de Sphinx

Ouvrez le fichier conf.py généré par Sphinx dans le répertoire docs/ et apportez les modifications suivantes :

a. Ajouter le chemin vers votre projet Django

Dans conf.py, vous devez ajouter le chemin vers le répertoire de votre projet Django. Cela permet à Sphinx de charger correctement les modules et les modèles Django pour générer la documentation.

Ajoutez cette ligne avant toute autre configuration dans le fichier conf.py :

python

import os

import sys

sys.path.insert(0, os.path.abspath('../')) # Chemin vers le

répertoire principal du projet Django

b. Spécifier le module de paramètres Django

Toujours dans conf.py, définissez la variable DJANGO_SETTINGS_MODULE pour que Sphinx puisse accéder à votre configuration Django (par exemple, le fichier settings.py de votre projet).

Ajoutez ceci dans conf.py :

python

import django

```
import os
os.environ['DJANGO_SETTINGS_MODULE'] = 'myproject.settings' #
Remplacez par le chemin de votre fichier settings.py
django.setup()
```

Cela configure Sphinx pour qu'il charge votre projet Django et puisse récupérer les informations nécessaires sur les modèles et les vues.

c. Ajouter des extensions utiles

Dans conf.py, ajoutez les extensions suivantes pour activer autodoc (pour extraire la documentation des docstrings) et le thème pour améliorer l'apparence.

```
python
extensions = [
'sphinx.ext.autodoc', # Extraction automatique de la
documentation des docstrings

'sphinx.ext.viewcode', # Lien vers le code source dans la
documentation
'sphinx_rtd_theme', # Thème pour la documentation
]
# Choisissez un thème (optionnel, mais recommandé)
html_theme = 'sphinx_rtd_theme'
```

4. Ajouter la documentation de votre code source avec autodoc

Pour inclure automatiquement la documentation des modules Python (comme vos modèles, vues, etc.), vous devez ajouter des directives autodoc dans vos fichiers .rst.

Exemple pour index.rst :

.. automodule:: home.models

:members:

:undoc-members:

:show-inheritance:

.. automodule:: home.views

:members:

:undoc-members:

:show-inheritance:

.. automodule:: home.urls

:members:

:undoc-members:

:show-inheritance:

.. automodule:: home.forms

:members:

:undoc-members:

:show-inheritance:

.. automodule:: home.admin

:members:

:undoc-members:

:show-inheritance:

Cette directive automodule va générer la documentation pour tous les membres de votre

module Python, en incluant les classes et les fonctions, avec leur docstring.

5. Générer la documentation

Une fois votre fichier conf.py configuré et vos fichiers .rst préparés, vous pouvez générer de la documentation.

a. Accédez au répertoire source :

bash

cd source

b. Générer la documentation HTML :

Exécutez la commande suivante pour générer la documentation au format HTML :

bash

sphinx-build -b html . build/html -v

Cela va créer la documentation dans un répertoire _build/html/.

c. Ouvrir la documentation :

Une fois la documentation générée, vous pouvez l'ouvrir dans votre navigateur :

bash

open build/html/index.html

Chapitre 2 : Analyse et conception.

Introduction:

Avant la réalisation de tout projet informatique, il faut passer par une étape cruciale qui est l'étape d'analyse et de conception.

Dans ce chapitre, on a opté pour la démarche UP et la modélisation UML pour représenter la dynamique de notre site. Pour cela on a réalisé notre travail en deux phases:

La phase d'analyse permet de définir les besoins fonctionnels et non fonctionnels du système ainsi que les différents acteurs qui interagissent avec notre système et leurs tâches associées.

La phase de conception permet quant à elle de décrire le fonctionnement du système en utilisant les différents diagrammes UML afin de faciliter la réalisation de ce dernier.

1.Définition du langage UML:

UML (Unified Modeling Language) est un langage de modélisation graphique utilisé principalement dans le domaine de l'ingénierie logicielle pour visualiser, spécifier, concevoir et documenter les systèmes logiciels. Il fournit un ensemble de notations standardisées et de conventions de représentation graphique pour décrire les différentes perspectives d'un système logiciel, y compris sa structure, son comportement, ses interactions et son architecture.

Dans notre projet, nous avons opté pour la modélisation de quelques diagrammes qui sont :

- Diagramme de contexte .
- Diagrammes de cas d'utilisation.
- diagramme de classe

2.Identifications des acteurs :

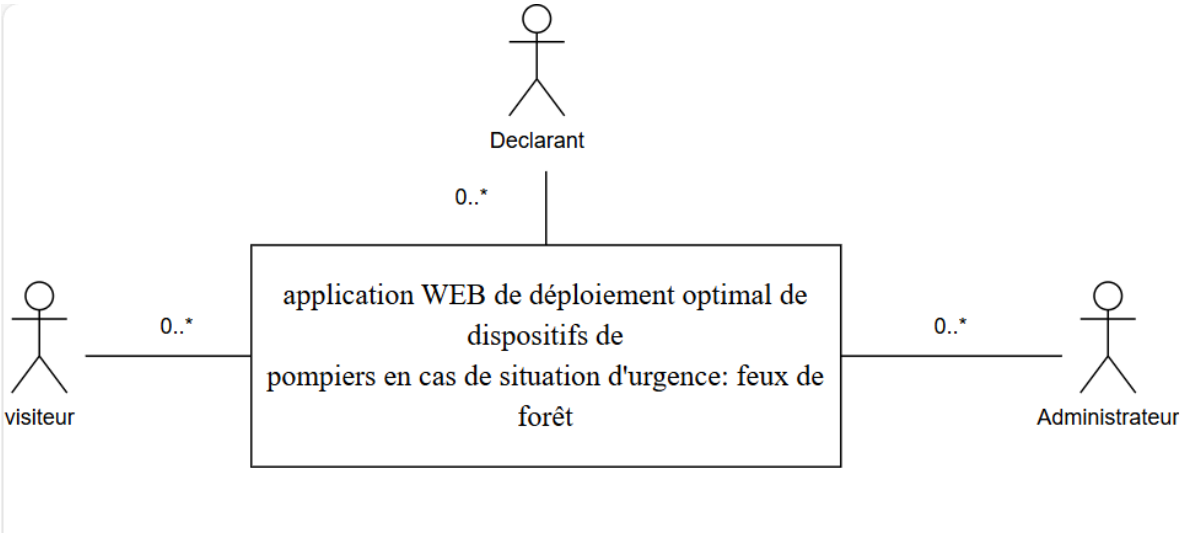
- ***Le visiteur***: toute personne pouvant consulter le site dans le but de prendre connaissance sur le site, cet acteur peut devenir un client en s'inscrivant sur le site .

- ***Le client :*** c'est un acteur qui a déjà créé un compte sur le site, il peut donc consulter les consignes de sécurité , déclarer des incendies, a propos et tout ce que le site offre comme fonctionnalités en toute sécurité et confidentialité .
- ***L'administrateur,:*** appelé aussi webmaster, est responsable de gérer toutes les fonctionnalités du site.

3-Diagramme de contexte:

Un diagramme de contexte est un type de diagramme de modélisation qui représente les interactions entre un système et son environnement, il permet de délimiter le champ d'étude du système, il est généralement le premier diagramme à définir dans la phase conception. Il répond à la question: "quels sont les acteurs et éléments environnants au système ?

Après identification des acteurs de notre système on obtient le diagramme suivant :



4-Identification des tâches :

- L'utilisateur peut se rendre sur le site pour déclarer un incendie de forêt.

1.1. Depuis la page d'accueil, il peut accéder directement à la section "Déclarer un incendie". Dans cette section, il remplit un formulaire avec son nom, son numéro de téléphone, le type d'incendie, la gravité estimée et la localisation de l'incendie, puis soumet la déclaration.

1.2. L'utilisateur a également la possibilité de créer un compte en accédant à la page d'authentification. Pour s'inscrire, il doit fournir son nom, son adresse e-mail, son numéro de téléphone et choisir un mot de passe, puis soumettre le formulaire d'inscription.

1.3. En cliquant sur l'icône "Appels d'urgence" située à droite de la barre de navigation, l'utilisateur peut accéder aux coordonnées téléphoniques des unités de pompiers et les contacter directement.

1.4. Il peut consulter la page "Consignes de sécurité" pour obtenir des instructions sur les mesures à prendre en cas d'incendie de forêt.

1.5. Enfin, l'utilisateur peut visiter la page "À propos" pour en savoir plus sur les pompiers de Tizi Ouzou et leur mission.

Gestion des incendies et des ressources par l'administrateur

- En tant que chef des pompiers, l'administrateur accède à son interface après s'être authentifié en entrant son nom d'utilisateur et son mot de passe.

2.1. Il peut consulter la table "Incendies", qui regroupe toutes les déclarations soumises par les citoyens.

2.2. L'administrateur accède à la table "Citoyens" pour visualiser les inscriptions des utilisateurs. Il peut également gérer les comptes des citoyens en supprimant un utilisateur ou en modifiant ses informations.

2.3. Dans la table "Pompiers", l'administrateur peut :

- Ajouter un nouveau membre du personnel.
- Supprimer un pompier existant.
- Mettre à jour l'état d'un pompier.

2.3.1. Lors de la mise à jour d'un pompier, l'administrateur peut :

- Modifier l'équipe à laquelle il est affecté.
- Marquer le pompier comme "En mission" en cochant la case appropriée (ou laisser la case décochée si le pompier n'est pas en mission).
- Préciser la mission à laquelle il participe.

2.4. L'administrateur peut également gérer les véhicules via la table "Véhicules de pompier". Il a la possibilité d'ajouter un nouveau véhicule, d'en supprimer un existant ou de modifier son état.

2.4.1. La modification de l'état d'un véhicule inclut :

- Indiquer s'il est opérationnel ou en panne.
- Préciser s'il est disponible ou indisponible (indisponible signifie qu'il est actuellement en mission).

5:Conception:

5-1:Diagrammes de cas d'utilisation :

Les cas d'utilisation sont des outils de modélisation pour décrire les interactions entre un système et ses utilisateurs ou d'autres systèmes externes. Ils se présentent sous forme de descriptions textuelles ou graphiques, décrivant comment un utilisateur ou un système externe interagit avec le système pour atteindre un objectif spécifique. Chaque cas d'utilisation représente un scénario ou flux d'actions possibles, mettant en lumière les besoins et les objectifs de l'utilisateur, ainsi que les étapes ou actions impliquées dans l'interaction.

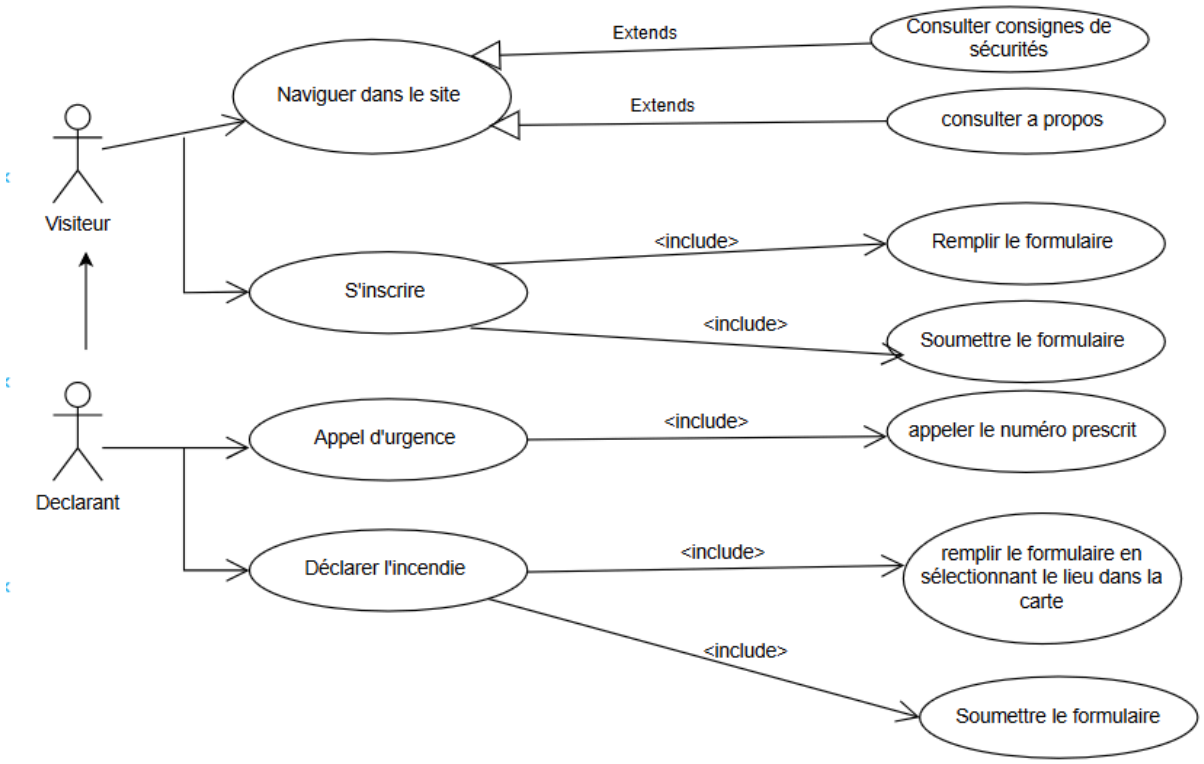


diagramme de cas d'utilisation visiteur , déclarant .

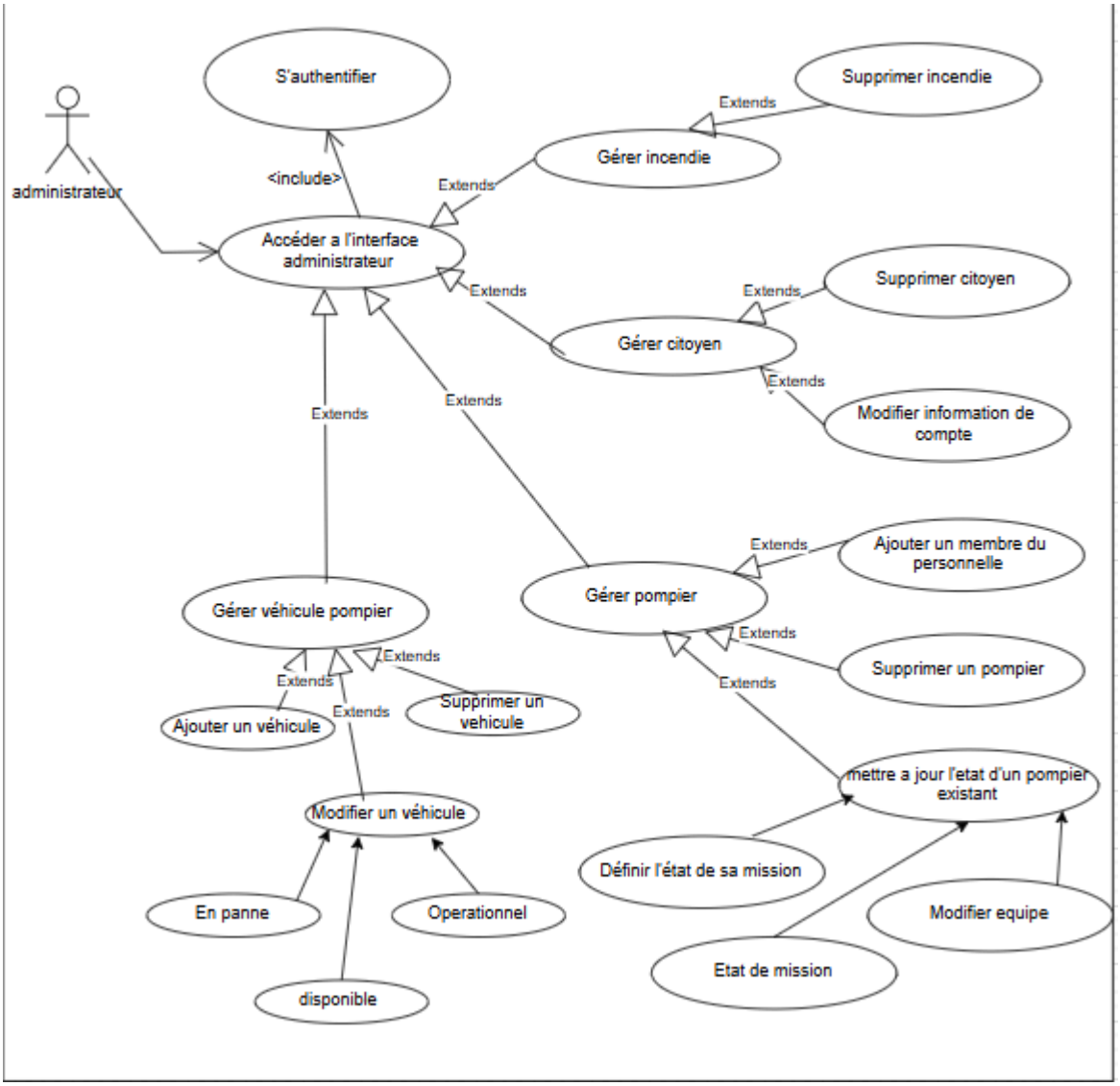


diagramme de cas d'utilisation : administrateur .

5.2: Diagramme de classe:

Un diagramme de classe est une représentation visuelle de la structure statique d'un système logiciel, mettant en évidence les classes, les attributs, les méthodes et les relations entre les objets. Il montre comment les classes interagissent les unes avec les autres, avec des boîtes pour représenter les classes et des lignes pour les relations. Ces diagrammes sont cruciaux pour concevoir et documenter la structure des logiciels orientés objet.

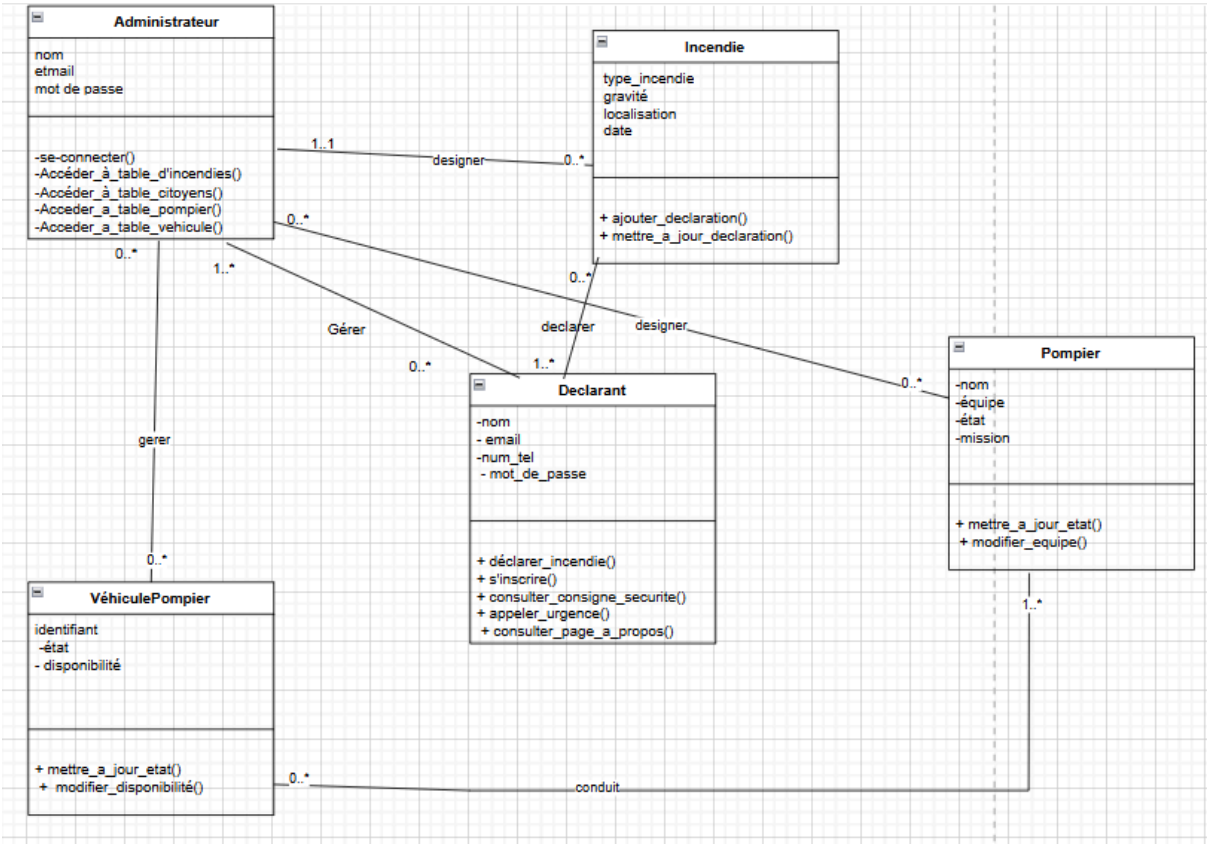


diagramme de classe

5.3 : schémas relationnel :

Administrateur(nom , email, mot de passe , type_incendie*)

Incendie(type incendie , gravité , localisation , date)

pompier(nom , equipe , etat , mission)

Déclarant(nom , email , num tél , mot de passe)

véhicule pompier(identifiant , etat , disponibilité)

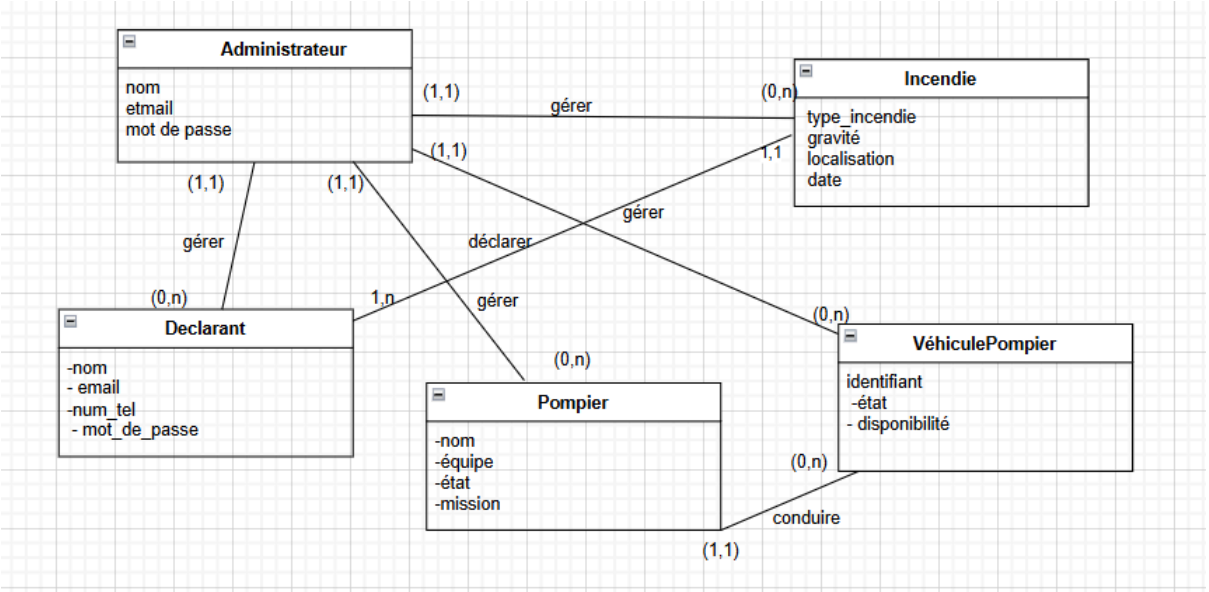
designer (nomPompier* , nomAdmisitrateur*)

Conduire(nomPompier *,Identifiant*)

Gérer(nomAdministrateur*,identifiant*)

Gérer(nomAdministrateur*,nomDeclarant)

Déclarer(type incendie*,nomDeclarant*)



Modèle conceptuel

Utilisation des données pour les besoins du cahier de charges : Algorithme d'optimisation :

1. Données :

Les données peuvent inclure des informations recueillies à partir de diverses sources (capteurs, bases de données, historique, etc.) qui sont utilisées pour résoudre un problème ou prendre des décisions.

2. Cahier des charges :

Le cahier des charges est un document décrivant les besoins, les contraintes et les objectifs du projet. Pour un algorithme, cela pourrait inclure :

Les objectifs à optimiser (par exemple, minimiser les coûts, maximiser la vitesse, réduire les risques).

Les contraintes (par exemple, temps de calcul limité, disponibilité des ressources).

Les métriques de performance attendues.

3. Algorithme d'optimisation :

Un algorithme d'optimisation vise à trouver la meilleure solution possible pour un problème donné. Par exemple :

Trouver l'itinéraire le plus court pour un véhicule (optimisation du chemin).

```
let routeControl = L.Routing.control({
  waypoints: [fireStation.getLatLng()], // Point de départ : l'unité de pompiers
  routeWhileDragging: true,
}).addTo(map);
```

L.Routing.control est une méthode fournie par la bibliothèque Leaflet Routing Machine.

Elle configure le calcul des itinéraires en fonction des points de départ (waypoints), et elle utilise par défaut OSRM en ligne comme moteur de routage.

Le calcul de l'itinéraire est délégué à une bibliothèque externe, appelée Leaflet Routing Machine, qui repose sur des moteurs de routage (comme OSRM, GraphHopper, ou autre).

Quel algorithme est utilisé ?

Le calcul du plus court chemin dans votre exemple dépend du moteur de routage utilisé par la bibliothèque Leaflet Routing Machine :

Par défaut, Leaflet Routing Machine utilise OSRM (Open Source Routing Machine).

OSRM utilise des variantes optimisées des algorithmes de graphes, principalement :

A* (A-star) : une version optimisée de Dijkstra utilisant une fonction heuristique pour accélérer les recherches.

Où cela est-il configuré ?

let routeControl = L.Routing.control({...}).addTo(map);

Le moteur OSRM en ligne est automatiquement utilisé si vous ne spécifiez rien d'autre.

Comment le modifier ?

Vous pouvez définir explicitement le moteur et donc influencer les algorithmes utilisés (OSRM, GraphHopper, etc.).

Chapitre 3 : Réalisation

1.L'environnement du développement :

1.1.Les outils utilisés :

Draw.io :

Draw.io est un outil de création de diagrammes en ligne offrant une variété de modèles comme les organigrammes, les diagrammes de flux et les diagrammes UML. Avec une interface intuitive et des fonctionnalités de glisser-déposer, il permet aux utilisateurs de concevoir facilement des diagrammes professionnels. Utilisé pour la documentation, la planification de projet et la modélisation des processus, Draw.io est polyvalent et adapté à divers besoins de visualisation.

Visual Studio Code :

Visual Studio Code (VS Code) est un éditeur de code source développé par Microsoft, disponible gratuitement et open-source . Il est largement utilisé pour écrire du code dans divers langages de programmation, grâce à sa légèreté, sa performance et sa flexibilité.

1.2.Les langages utilisées :

HTML :

HTML est le langage de balisage standard pour la création de pages Web, permettant de définir la structure et le contenu d'une page en utilisant des balises prédéfinies pour organiser les éléments de manière logique et cohérente.Sa dernière version est HTML5 [5].

CSS :

CSS est un langage de feuilles de style utilisé pour contrôler l'apparence et le formatage des pages Web en HTML et XML, offrant aux développeurs un contrôle précis sur la présentation visuelle de leur contenu.

JavaScript :

c’est un langage de programmation utilisé pour rendre les pages Web dynamiques et interactives, offrant aux développeurs la possibilité de créer des fonctionnalités avancées et des expériences utilisateur riches sur le Web.

Python:

c’est un langage de programmation simple et puissant, utilisé dans de nombreux domaines comme le web, les données et l'IA.

Django :

C'est un framework web Python qui facilite le développement rapide d'applications en offrant des outils prêts à l'emploi pour la gestion des bases de données, la sécurité et l'administration.

2.Le modèle logique de données:

Citoyen:

Nom du champ	Type de données	Description	Clé
FullName	varchar(255)	nom et prénom du citoyen	Primaire
email	varchar(255)	email du citoyen	/
Phone	int(10)	numéro de tél	/
Password	varchar(255)	mot de passe citoyen	/

Structure de la table citoyen

Incendie:

Nom du champ	Type de données	Description	Contraintes
severity	varchar(100)	gravité de l'incendie	/
date_reported	int(15)	date de déclaration	/
type	varchar(20)	type d'incendie	/
latitude	Float	numéro de téléphone	/
longitude	float	a longitude d'une position géographique	/

Structure de la table incendie

Pompier:

Nom du champ	Type de données	Description	Contraintes
Nom	varchar(15)	nom du pompier	/
Prenom	varchar(15)	prenom du pompier	/
numéro de tél	int(10)	numero de telephone du pompier	/
equipe	varchar(255)	l'équipe à laquelle il appartient	/
grade	varchar(255)	définit le grade du pompier	/
dateRec	date	la date quand il a été recruter	/
en mission	case à cocher	son état est ce que en mission ou non	/

mission	varchar(50)	définir la mission à laquelle il participe	/

Structure de la table pompier

Véhicule de pompiers :

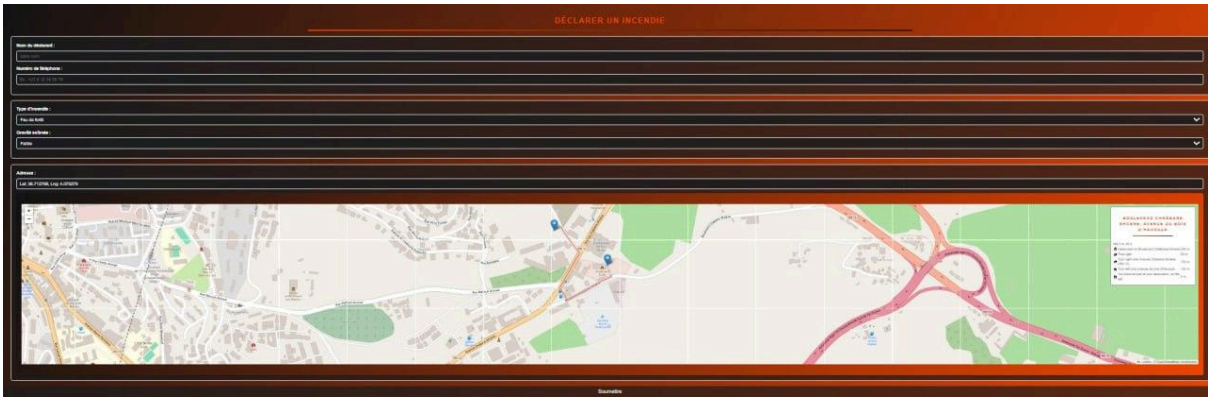
Nom du champ	Type de données	Description	Contraintes
num immat	varchar(50)	numéro immatriculation du camion	/
type de véhicule	varchar(20)	type de véhicule	/
marque	varchar(20)	marque du camion	/
etat	varchar(20)	l'état du camion opérationnel ou en panne	/
en service	case à cocher	est ce que le camion en service ou non	/

Structure de la table vehicule de pompiers:

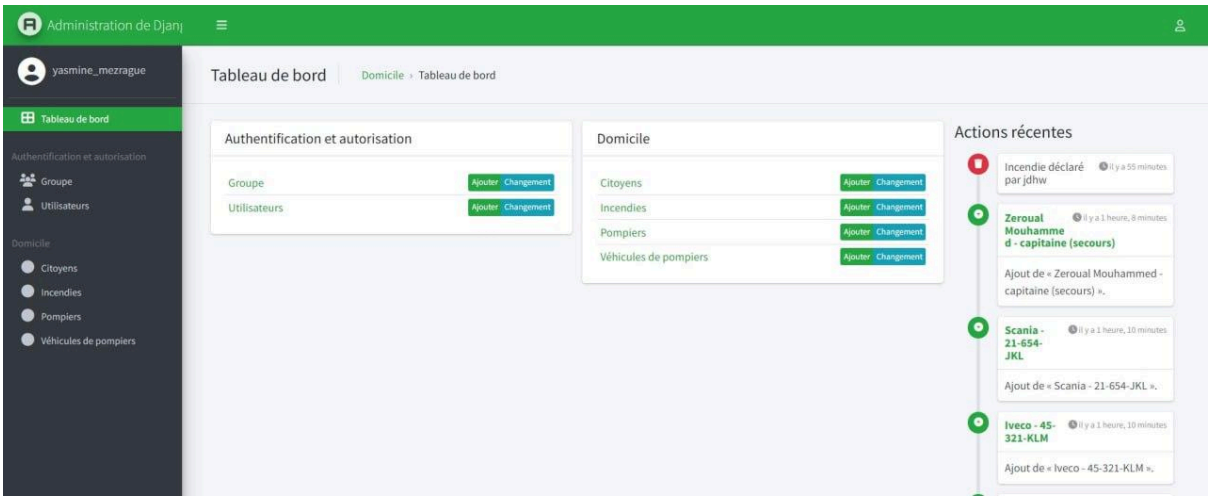
3. Les principales interfaces de notre site :

Page d'accueil 1

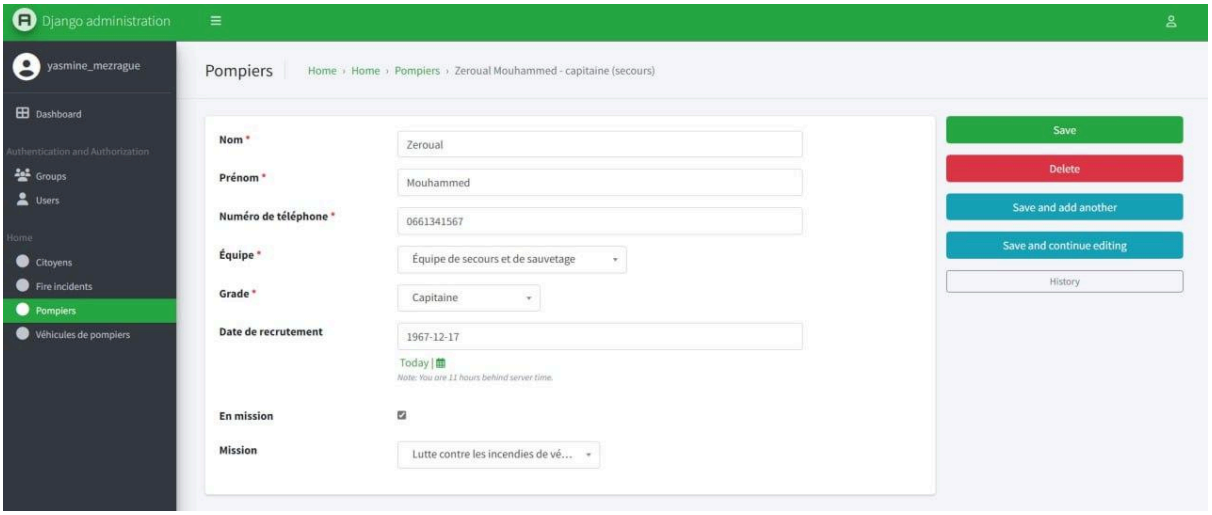
Page d'accueil 2



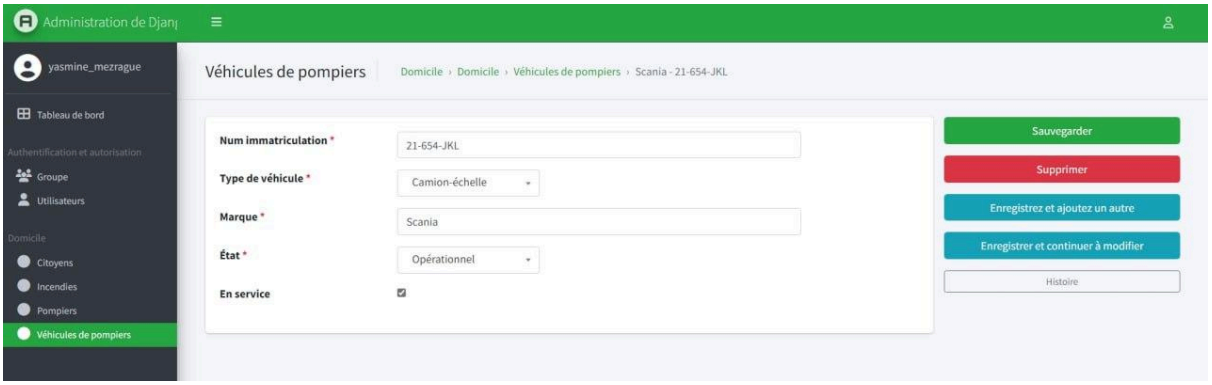
Le formulaire de déclaration



Le tableau de bord de la page administrateur



La page des pompiers



La page des véhicules des pompiers

Conclusion:

Dans ce chapitre nous avons exposé les différents outils et langages employés pour concrétiser notre application, tout en offrant un aperçu de celle-ci à travers quelques interfaces utilisateur.

Conclusion Générale :

En conclusion, l'intégration de la Business Intelligence, des outils ETL, des SIG, de l'optimisation et de l'IA permet de transformer les données en informations exploitables pour améliorer la gestion des situations d'urgence. Ces technologies offrent des solutions innovantes pour optimiser les interventions des secours, rendant les réponses plus rapides et efficaces, notamment dans des contextes complexes comme les incendies de forêt. Cette approche permet une gestion proactive des crises en temps réel, en utilisant des données provenant de sources variées, y compris les réseaux sociaux.

Bibliographie:

BI

Explication de qlq termes :

-NoSQL : fait référence aux **types de base de données non relationnelle** qui stockent des données dans un format différent des tables relationnelles