Documentation Technique - Fès Route Finder

Auteur : Manus AI
Date : 6 juin 2025
Version : 1.0

Table des matières

Introduction

Fès Route Finder est une application web end-to-end conçue pour aider les utilisateurs à trouver le meilleur chemin entre deux points dans la ville de Fès, au Maroc. L'application utilise une architecture moderne combinant un frontend React optimisé pour mobile, un backend Django, une base de données MongoDB et un modèle de machine learning pour optimiser les itinéraires.

Objectifs du projet

- Fournir une interface utilisateur intuitive et réactive pour la recherche d'itinéraires
- Permettre aux utilisateurs de sélectionner des points de départ et d'arrivée sur une carte interactive
- Calculer et afficher le meilleur chemin entre deux points
- Utiliser un modèle de machine learning pour optimiser les itinéraires en fonction des données historiques de trafic
- Offrir une expérience utilisateur optimisée pour les appareils mobiles

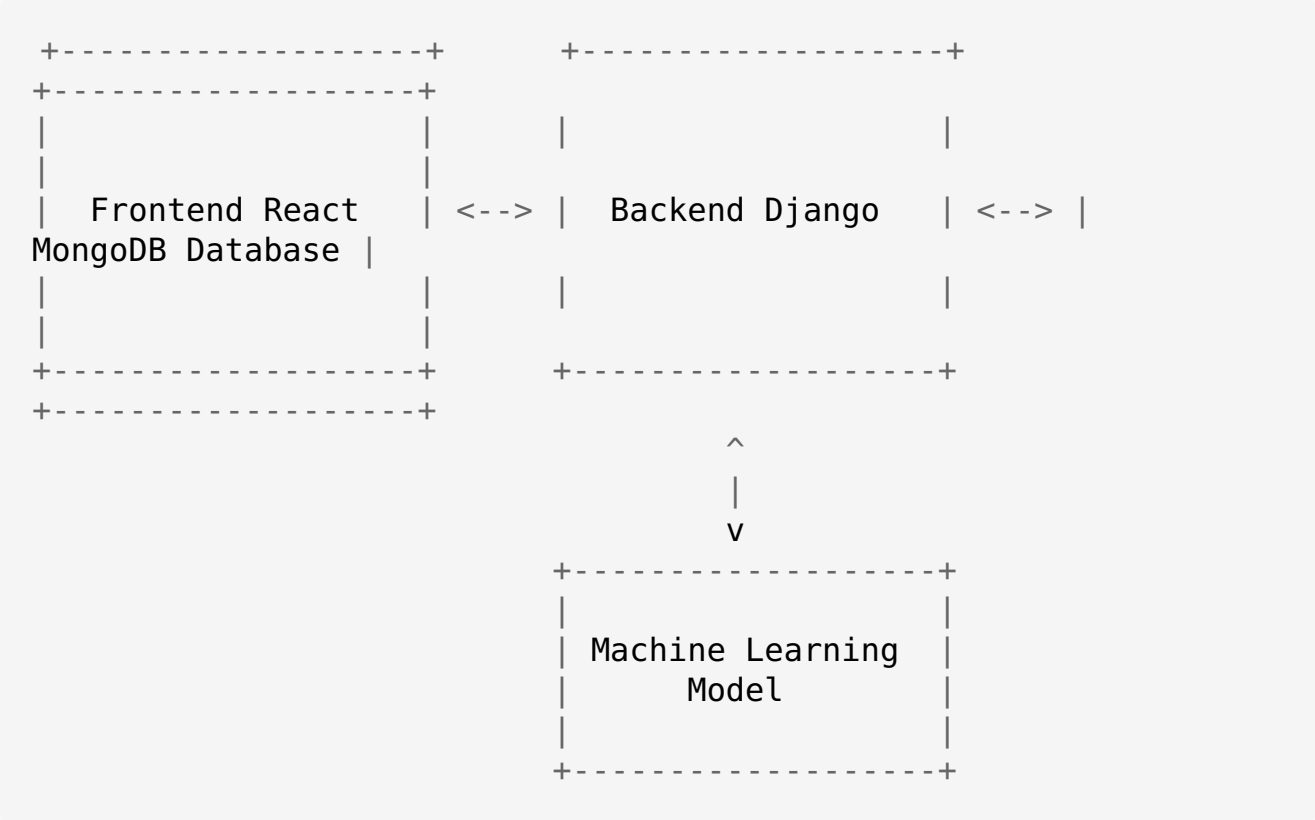
Technologies utilisées

- **Frontend** : React, Leaflet (pour la cartographie)
- **Backend** : Django, Django REST Framework
- **Base de données** : MongoDB
- **Machine Learning** : scikit-learn, pandas, numpy
- **Cartographie** : OpenStreetMap
- **Déploiement** : Services de déploiement statique et backend

Architecture du système

L'architecture de Fès Route Finder suit un modèle client-serveur classique avec une séparation claire entre le frontend et le backend. Le système est composé des éléments suivants :

Vue d'ensemble



Flux de données

1. L'utilisateur interagit avec l'interface React pour sélectionner des points sur la carte
2. Le frontend envoie des requêtes API au backend Django
3. Le backend traite les requêtes et interagit avec la base de données MongoDB
4. Pour l'optimisation d'itinéraire, le backend utilise le modèle de machine learning
5. Les résultats sont renvoyés au frontend pour affichage à l'utilisateur

Communication entre composants

La communication entre le frontend et le backend se fait via des API REST. Les données sont échangées au format JSON. Les principaux endpoints sont :

- `/api/locations/search/` : Recherche de lieux par nom
- `/api/routes/calculate/` : Calcul d'itinéraire standard
- `/api/routes/optimize/` : Calcul d'itinéraire optimisé avec ML

Backend (Django)

Le backend est développé avec Django et Django REST Framework, offrant une API RESTful pour le frontend.

Structure du projet

```
backend/
├── api/
│   ├── __init__.py
│   ├── apps.py
│   ├── models.py
│   ├── serializers.py
│   ├── urls.py
│   ├── utils.py
│   ├── views.py
│   └── ml_model/
│       ├── data/
│       ├── data_generator.py
│       ├── model_trainer.py
│       ├── predictor.py
│       ├── route_predictor_model.joblib
│       └── features.txt
├── route_finder/
│   ├── __init__.py
│   ├── asgi.py
│   ├── settings.py
│   ├── urls.py
│   └── wsgi.py
├── venv/
├── manage.py
└── run_server.sh
```

Modèles de données

Le backend utilise MongoDB comme base de données, avec PyMongo pour l'interaction. Les principales collections sont :

- `locations` : Stocke les informations sur les lieux (nom, coordonnées)
- `routes` : Stocke les itinéraires calculés (points de départ et d'arrivée, chemin, distance, durée)
- `traffic_data` : Stocke les données de trafic pour l'entraînement du modèle ML

API REST

Le backend expose plusieurs endpoints API :

SearchLocationView

```python
class SearchLocationView(APIView):
    """
    API pour rechercher des lieux par nom
    """
    def post(self, request):
        # Recherche de lieux par nom
        # Retourne une liste de lieux correspondants
```

RouteView

```python
class RouteView(APIView):
    """
    API pour calculer un itinéraire entre deux points
    """
    def post(self, request):
        # Calcul d'itinéraire standard
        # Retourne le chemin, la distance et la durée
```

OptimizedRouteView

```python
class OptimizedRouteView(APIView):
    """
    API pour calculer un itinéraire optimisé entre deux points
    en utilisant le modèle de machine learning
    """
    def post(self, request):
        # Calcul d'itinéraire optimisé avec ML
        # Retourne le chemin, la distance et la durée optimisés
```

Services utilitaires

Le backend comprend plusieurs services utilitaires :

- `GeocodingService` : Service de géocodage utilisant Nominatim (OpenStreetMap)
- `RoutingService` : Service de calcul d'itinéraire
- `TrafficDataService` : Service de gestion des données de trafic

Frontend (React)

Le frontend est développé avec React et utilise Leaflet pour l'affichage de la carte interactive.

Structure du projet

```
frontend/
├── public/
│   ├── index.html
│   ├── favicon.ico
│   └── ...
├── src/
│   ├── App.js
│   ├── App.css
│   ├── index.js
│   ├── index.css
│   └── components/
│       └── Map.js
├── package.json
└── ...
```

Composants principaux

App.js

Composant principal de l'application qui définit la structure globale de la page.

```
function App() {
  return (
    <div className="App">
      <header className="App-header">
        <h1>Fès Route Finder</h1>
        <p>Trouvez le meilleur chemin dans la ville de Fès</p>
      </header>
      <main>
```

```
          <Map />
        </main>
        <footer>
          <p>&copy; 2025 Fès Route Finder</p>
        </footer>
      </div>
    );
  }
```

Map.js

Composant central qui gère la carte interactive et toutes les fonctionnalités associées.

Principales fonctionnalités : - Affichage de la carte avec Leaflet - Recherche de lieux - Sélection des points de départ et d'arrivée - Calcul et affichage de l'itinéraire - Gestion des erreurs et des états de chargement

Interface utilisateur

L'interface utilisateur est conçue pour être intuitive et réactive, avec une attention particulière à l'expérience mobile :

- Barre de recherche en haut de l'écran
- Affichage des points de départ et d'arrivée sélectionnés
- Boutons d'action (calculer, inverser, effacer)
- Carte interactive occupant la majeure partie de l'écran
- Affichage des informations d'itinéraire (distance, durée)

Optimisation mobile

L'interface est optimisée pour les appareils mobiles avec :

- Design responsive s'adaptant à différentes tailles d'écran
- Contrôles tactiles pour la carte
- Disposition verticale des éléments pour une meilleure utilisation sur mobile
- Taille des boutons adaptée à l'utilisation tactile

Base de données (MongoDB)

MongoDB a été choisi comme base de données pour sa flexibilité et ses performances avec les données géospatiales.

Collections

Collection `locations`

Stocke les informations sur les lieux.

```
{
  "name": "Médina de Fès",
  "coordinates": [-5.0, 34.0],
  "type": "Point"
}
```

Collection `routes`

Stocke les itinéraires calculés.

```
{
  "start_point": [-5.0, 34.0],
  "end_point": [-4.9, 34.1],
  "path": [[-5.0, 34.0], [-4.95, 34.05], [-4.9, 34.1]],
  "distance": 5000,
  "duration": 600,
  "created_at": "2025-06-06T12:00:00Z"
}
```

Collection `traffic_data`

Stocke les données de trafic pour l'entraînement du modèle ML.

```
{
  "segment": "segment_id",
  "timestamp": "2025-06-06T12:00:00Z",
  "speed": 45,
  "congestion_level": 2
}
```

Indexation

Des index sont créés pour optimiser les requêtes :

```
 # Index géospatial pour les recherches de proximité
locations_collection.create_index([("coordinates", "2dsphere")])

# Index pour les recherches de lieux par nom
locations_collection.create_index([("name", "text")])
```

```
# Index pour les recherches d'itinéraires
routes_collection.create_index([("start_point", 1),
("end_point", 1)])

# Index pour les données de trafic
traffic_data_collection.create_index([("segment", 1),
("timestamp", -1)])
```

Modèle de Machine Learning

Le modèle de machine learning est utilisé pour optimiser les itinéraires en fonction des données historiques de trafic.

Données d'entraînement

Les données d'entraînement sont générées synthétiquement pour simuler des conditions de trafic réelles :

- Points de départ et d'arrivée aléatoires dans la zone de Fès
- Distances calculées avec la formule de Haversine
- Facteurs de trafic variables selon l'heure de la journée et le jour de la semaine
- Vitesses moyennes et durées de trajet calculées en fonction des facteurs de trafic

Prétraitement des données

Les données sont prétraitées avant l'entraînement :

- Normalisation des coordonnées et des distances
- Encodage des heures et des jours de la semaine
- Division en ensembles d'entraînement et de test (80/20)

Modèle utilisé

Le modèle utilisé est un RandomForestRegressor de scikit-learn, choisi pour sa capacité à capturer des relations non linéaires et sa robustesse.

```
pipeline = Pipeline([
    ('scaler', StandardScaler()),
    ('model', RandomForestRegressor(
        n_estimators=100,
        max_depth=10,
        random_state=42
```

```
    ))
])
```

Caractéristiques (features)

Les caractéristiques utilisées pour la prédiction sont :

- Coordonnées du point de départ (latitude, longitude)
- Coordonnées du point d'arrivée (latitude, longitude)
- Distance à vol d'oiseau
- Heure de la journée (0-23)
- Jour de la semaine (0-6)

Évaluation du modèle

Le modèle est évalué sur l'ensemble de test avec les métriques suivantes :

- MAE (Mean Absolute Error) : environ 0.80 minutes
- RMSE (Root Mean Squared Error) : environ 1.20 minutes
- R^2 : environ 0.86

Intégration avec le backend

Le modèle entraîné est sauvegardé avec joblib et chargé par le backend pour les prédictions en temps réel :

```python
class MLIntegration:
    """
    Classe pour l'intégration du modèle de machine learning
    """
    def __init__(self):
        # Initialiser le prédicteur
        self.predictor = RoutePredictor(model_dir)

    def predict_optimal_route(self, start_point, end_point):
        # Prédire le meilleur itinéraire
        # ...
```

API et Intégration

API Backend

Le backend expose plusieurs endpoints API RESTful :

Endpoint	Méthode	Description	Paramètres	Réponse
`/api/locations/search/`	POST	Recherche de lieux par nom	`query` : Texte de recherche	Liste de lieux avec noms et coordonnées
`/api/routes/calculate/`	POST	Calcul d'itinéraire standard	`start_point` : [lon, lat], `end_point` : [lon, lat]	Chemin, distance, durée
`/api/routes/optimize/`	POST	Calcul d'itinéraire optimisé	`start_point` : [lon, lat], `end_point` : [lon, lat]	Chemin, distance, durée optimisés

Intégration Frontend-Backend

Le frontend communique avec le backend via des requêtes HTTP utilisant axios :

```
// Exemple de requête pour calculer un itinéraire optimisé
const calculateRoute = async () => {
  try {
    const backendUrl = window.location.hostname === 'localhost'
      ? 'http://localhost:8000'
      : 'https://8000-
if1lru4hrm22pfu00ju94-5bc75307.manus.computer';

    const response = await axios.post(`${backendUrl}/api/routes/
optimize/`, {
      start_point: startPoint.lngLat,
      end_point: endPoint.lngLat
    });

    // Traitement de la réponse
    // ...
  } catch (err) {
    console.error('Erreur lors du calcul de l\'itinéraire:',
err);
    setError('Erreur lors du calcul de l\'itinéraire. Veuillez
réessayer.');
  }
};
```

Gestion des erreurs

Le système implémente une gestion des erreurs robuste :

- Côté backend : Validation des données d'entrée, gestion des exceptions, codes d'état HTTP appropriés
- Côté frontend : Gestion des erreurs de requête, affichage de messages d'erreur conviviaux, retentatives automatiques

Déploiement

Frontend

Le frontend est déployé en tant qu'application statique :

1. Construction de l'application avec `npm run build`
2. Déploiement des fichiers statiques sur un service d'hébergement

URL de déploiement : `https://tsllfako.manus.space`

Backend

Le backend est déployé en tant que service web :

1. Configuration de l'environnement virtuel Python
2. Installation des dépendances
3. Configuration de la base de données MongoDB
4. Lancement du serveur Django

URL de déploiement : `https://8000-if1lru4hrm22pfu00ju94-5bc75307.manus.computer`

Configuration CORS

Le backend est configuré pour accepter les requêtes cross-origin du frontend :

```
# settings.py
CORS_ALLOW_ALL_ORIGINS = True
```

Sécurité et Performance

Sécurité

L'application étant un prototype, les mesures de sécurité sont basiques :

- Validation des données d'entrée côté backend
- Protection contre les injections MongoDB
- Utilisation de HTTPS pour les communications

Pour une version de production, il faudrait ajouter : - Authentification et autorisation - Rate limiting - Protection contre les attaques CSRF - Audit de sécurité complet

Performance

Plusieurs optimisations de performance sont mises en place :

- Indexation MongoDB pour des requêtes rapides
- Mise en cache des itinéraires calculés
- Chargement optimisé des tuiles de carte
- Lazy loading des composants React

Limites actuelles

- L'application est conçue pour un nombre limité d'utilisateurs
- Le modèle ML est entraîné sur des données synthétiques
- Les calculs d'itinéraire sont simplifiés

Maintenance et Évolution

Maintenance

Pour maintenir l'application :

1. Surveiller les logs d'erreur
2. Mettre à jour régulièrement les dépendances
3. Sauvegarder la base de données
4. Vérifier périodiquement les performances

Évolutions possibles

Plusieurs améliorations pourraient être apportées :

- Intégration de données de trafic en temps réel
- Ajout de fonctionnalités comme les favoris ou l'historique
- Support pour différents modes de transport
- Extension à d'autres villes
- Amélioration du modèle ML avec des données réelles
- Ajout d'une authentification utilisateur
- Développement d'applications mobiles natives

Roadmap suggérée

1. **Court terme** (1-3 mois) :
2. Correction des bugs identifiés
3. Amélioration de l'interface utilisateur
4. Optimisation des performances
5. **Moyen terme** (3-6 mois) :
6. Intégration de données de trafic réelles
7. Ajout de fonctionnalités utilisateur
8. Extension à d'autres villes marocaines
9. **Long terme** (6-12 mois) :
10. Développement d'applications mobiles
11. Intégration avec d'autres services (météo, événements)
12. Monétisation (version premium, API pour entreprises)