Documentation Détaillée

Application Médicale

Réalisé par

Said Mazen
Ben Brahim Nazih
Ben Hmida Hamza

7 mai 2025

Table des matières

Chapitre 1

Vue d'ensemble

L'application MediConnect est une plateforme complète de gestion de rendez-vous médicaux, développée avec Ionic/Angular pour le frontend et Python/Flask pour le backend. Elle permet aux patients de prendre des rendez-vous avec des médecins, aux médecins de gérer leur pratique, et aux administrateurs de superviser les utilisateurs et les données. Cette application offre une expérience utilisateur fluide et intuitive, accessible sur mobile et web.

Première partie

Manuel d'Utilisation

Chapitre 2

Guide de l'Utilisateur

2.1 Premiers Pas

2.1.1 Inscription

1. Accédez à la page d'inscription
2. Remplissez le formulaire avec vos informations
3. Choisissez votre type de compte (Patient/Médecin)
4. Validez votre inscription

Placeholder pour une capture d'écran : Une capture d'écran de la page d'inscription serait affichée ici. Elle montrerait un formulaire avec des champs pour l'email, le mot de passe, un menu déroulant pour sélectionner le type de compte (Patient ou Médecin), et un bouton "S'inscrire" en bas. Le design utilise une palette de couleurs claires avec des éléments centrés pour une navigation intuitive.

2.1.2 Connexion

1. Accédez à la page de connexion
2. Entrez vos identifiants
3. Cliquez sur "Se connecter"

Placeholder pour une capture d'écran : Une capture d'écran de la page de connexion serait affichée ici. Elle présenterait une interface simple avec deux champs de saisie (un pour l'email, un pour le mot de passe), un bouton "Se connecter" en bleu, et un lien "Mot de passe oublié ?" en dessous. L'arrière-plan pourrait inclure un dégradé subtil pour une apparence moderne.

2.2 Fonctionnalités Principales

2.2.1 Gestion des Rendez-vous

— Consultation du calendrier
— Prise de rendez-vous
— Modification/Annulation
— Rappels automatiques

Placeholder pour une capture d'écran : Une capture d'écran de la page de gestion des rendez-vous serait affichée ici. Elle montrerait un calendrier mensuel interactif à gauche, où les jours avec des rendez-vous sont surlignés en vert. À droite, une liste

de rendez-vous à venir serait visible, chaque entrée affichant la date, l'heure, le nom du médecin, et des boutons pour modifier ou annuler le rendez-vous. Une icône de notification rappellerait les rendez-vous imminents.

2.2.2 Messagerie

— Envoi de messages

— Partage de documents

— Historique des conversations

Placeholder pour une capture d'écran : Une capture d'écran de l'interface de messagerie serait affichée ici. Elle illustrerait une conversation entre un patient et un médecin, avec des bulles de texte (bleues pour l'utilisateur, grises pour le correspondant). En bas, une zone de texte pour écrire un message, un bouton pour joindre des documents (icône de trombone), et un bouton d'envoi (icône d'avion en papier) seraient visibles. À gauche, une liste des conversations récentes permettrait de naviguer entre différents échanges.

Deuxième partie

Rapport Technique

Chapitre 3

Structure du Projet

3.1 Frontend (`src/`)

La structure du frontend est organisée comme suit :

- `src/app/`
  - `pages/` : Contient les pages principales comme `doctor-dashboard/`, `doctor-details/`, `doctor-register/`, `home/`, `login/`, `appointments/`, `profile/`
  - `services/` : Services pour gérer les appels API et la logique métier
  - `components/` : Composants réutilisables
  - `guards/` : Gardes pour la sécurité des routes
- `assets/` : Ressources statiques (images, icônes)
- `theme/` : Fichiers de style global (SCSS)
- `environments/` : Configurations d'environnement

3.2 Backend (`backend/`)

La structure du backend est organisée comme suit :

- `routes/` : Routes API pour les différentes fonctionnalités
- `models/` : Schémas de données MongoDB
- `services/` : Logique métier et services
- `utils/` : Utilitaires (gestion des erreurs, logs)
- `logs/` : Journaux d'application
- `app.py` : Point d'entrée principal du serveur Flask
- `config.py` : Configuration de l'application
- `requirements.txt` : Dépendances Python

Chapitre 4

Architecture Technique

4.1 Stack Technologique

4.1.1 Frontend

— **Framework** : Ionic 7, Angular 16
— **Langage** : TypeScript
— **CSS** : SCSS avec Tailwind
— **État** : NgRx
— **Tests** : Jasmine/Karma

4.1.2 Backend

— **Langage** : Python
— **Framework** : Flask
— **Base de données** : MongoDB
— **Authentification** : JWT
— **Tests** : pytest (recommandé)

Chapitre 5

Fonctionnalités Principales et Code

5.1 Authentification

5.1.1 Service d'Authentification

```
// auth.service.ts
@Injectable({
  providedIn: 'root'
})
export class AuthService {
  constructor(private http: HttpClient) {}

  login(credentials: LoginDTO): Observable<any> {
    return this.http.post(`${API_URL}/auth/login`, credentials);
  }

  register(userData: RegisterDTO): Observable<any> {
    return this.http.post(`${API_URL}/auth/register`, userData);
  }

  logout(): void {
    localStorage.removeItem('token');
  }

  isAuthenticated(): boolean {
    return !!localStorage.getItem('token');
  }
}
```

5.1.2 Guard d'Authentification

```
// auth.guard.ts
@Injectable({
  providedIn: 'root'
})
export class AuthGuard implements CanActivate {
  constructor(private authService: AuthService, private router:
     Router) {}

  canActivate(): boolean {
    if (this.authService.isAuthenticated()) {
      return true;
```

```
    }
    this.router.navigate(['/login']);
    return false;
  }
}
```

5.2 Gestion des Rendez-vous

5.2.1 Service des Rendez-vous

```
// appointment.service.ts
@Injectable({
  providedIn: 'root'
})
export class AppointmentService {
  constructor(private http: HttpClient) {}

  createAppointment(appointment: AppointmentDTO): Observable<any> {
    return this.http.post(`${API_URL}/appointments`, appointment);
  }

  getAppointments(): Observable<Appointment[]> {
    return this.http.get<Appointment[]>(`${API_URL}/appointments`);
  }

  updateAppointment(id: string, appointment: AppointmentDTO): Observable<any> {
    return this.http.put(`${API_URL}/appointments/${id}`, appointment);
  }

  deleteAppointment(id: string): Observable<any> {
    return this.http.delete(`${API_URL}/appointments/${id}`);
  }
}
```

5.2.2 Modèle de Rendez-vous

```
// appointment.model.ts
export interface Appointment {
  id: string;
  patientId: string;
  doctorId: string;
  date: Date;
  status: 'scheduled' | 'completed' | 'cancelled';
```

```
  type: string;
  notes?: string;
}
```

5.3 Messagerie

5.3.1 Service de Messagerie

```
// message.service.ts
@Injectable({
  providedIn: 'root'
})
export class MessageService {
  constructor(private http: HttpClient) {}

  sendMessage(message: MessageDTO): Observable<any> {
    return this.http.post(`${API_URL}/messages`, message);
  }

  getConversation(userId: string): Observable<Message[]> {
    return this.http.get<Message[]>(`${API_URL}/messages/
       conversation/${userId}`);
  }

  markAsRead(messageId: string): Observable<any> {
    return this.http.put(`${API_URL}/messages/${messageId}/read`,
        {});
  }
}
```

5.3.2 Modèle de Message

```
// message.model.ts
export interface Message {
  id: string;
  senderId: string;
  receiverId: string;
  content: string;
  timestamp: Date;
  read: boolean;
}
```

5.4 Base de Données

5.4.1 Schémas MongoDB

Les schémas MongoDB définissent la structure des données pour les utilisateurs, les rendez-vous et les messages.

```
// user.model.js
const userSchema = new Schema({
  email: { type: String, required: true, unique: true },
  password: { type: String, required: true },
  role: { type: String, enum: ['patient', 'doctor', 'admin'] },
  profile: {
    name: String,
    phone: String,
    address: String,
    specialization: String // Pour les m decins
  },
  createdAt: { type: Date, default: Date.now }
});

// appointment.model.js
const appointmentSchema = new Schema({
  patientId: { type: Schema.Types.ObjectId, ref: 'User' },
  doctorId: { type: Schema.Types.ObjectId, ref: 'User' },
  date: { type: Date, required: true },
  status: { type: String, enum: ['scheduled', 'completed', '
     cancelled'] },
  type: { type: String, required: true },
  notes: String,
  createdAt: { type: Date, default: Date.now }
});

// message.model.js
const messageSchema = new Schema({
  senderId: { type: Schema.Types.ObjectId, ref: 'User' },
  receiverId: { type: Schema.Types.ObjectId, ref: 'User' },
  content: { type: String, required: true },
  timestamp: { type: Date, default: Date.now },
  read: { type: Boolean, default: false }
});
```

5.5 API Endpoints

5.5.1 Routes des Médecins

```
# doctors.py
@app.route('/api/doctors', methods=['GET'])
def list_doctors():
    # Logique pour lister les m decins
```

```
    return jsonify(doctors)

@app.route('/api/doctors/<id>', methods=['GET'])
def get_doctor(id):
    # Logique pour obtenir les d tails d'un m decin
    return jsonify(doctor)

@app.route('/api/doctors', methods=['POST'])
def create_doctor():
    data = request.get_json()
    # Logique pour cr er un m decin
    return jsonify({"message": "M decin cr    "})

@app.route('/api/doctors/<id>', methods=['PUT'])
def update_doctor(id):
    data = request.get_json()
    # Logique pour mettre    jour un m decin
    return jsonify({"message": "M decin mis    jour"})

@app.route('/api/doctors/<id>', methods=['DELETE'])
def delete_doctor(id):
    # Logique pour supprimer un m decin
    return jsonify({"message": "M decin supprim "})
```

Placeholder pour une capture d'écran : Une capture d'écran de la réponse API `GET /api/doctors` serait affichée ici. Elle montrerait un résultat JSON avec une liste de médecins, chaque entrée incluant des champs comme `id`, `name`, `specialization`, et `address`. La réponse serait formatée avec une indentation claire, affichée dans un outil comme Postman ou un navigateur.

5.5.2 Routes des Utilisateurs

```
# auth.py
@app.route('/api/auth/register', methods=['POST'])
def register():
    data = request.get_json()
    # Logique d'inscription
    return jsonify({"message": "Utilisateur inscrit"})

@app.route('/api/auth/login', methods=['POST'])
def login():
    data = request.get_json()
    # Logique de connexion
    return jsonify({"token": "jwt_token"})

@app.route('/api/users/profile', methods=['GET'])
def get_profile():
    # Logique pour obtenir le profil
    return jsonify(profile)

@app.route('/api/users/profile', methods=['PUT'])
```

```
def update_profile():
    data = request.get_json()
    # Logique pour mettre    jour le profil
    return jsonify({"message": "Profil mis    jour"})
```

Placeholder pour une capture d'écran : Une capture d'écran de la réponse API `GET /api/users/profile` serait affichée ici. Elle montrerait un objet JSON contenant les détails du profil utilisateur, tels que `name`, `email`, `phone`, et `address`, avec une mise en forme soignée dans un outil de test API.

5.5.3 Routes des Rendez-vous

```
# appointments.py
@app.route('/api/appointments', methods=['POST'])
def create_appointment():
    data = request.get_json()
    # Logique de cr ation
    return jsonify({"message": "Rendez-vous  c r    "})

@app.route('/api/appointments', methods=['GET'])
def get_all_appointments():
    # Logique de r cup ration
    return jsonify(appointments)

@app.route('/api/appointments/<id>', methods=['PUT'])
def update_appointment(id):
    data = request.get_json()
    # Logique de mise    jour
    return jsonify({"message": "Rendez-vous mis    jour"})

@app.route('/api/appointments/<id>', methods=['DELETE'])
def delete_appointment(id):
    # Logique de suppression
    return jsonify({"message": "Rendez-vous supprim "})
```

Placeholder pour une capture d'écran : Une capture d'écran de la réponse API `GET /api/appointments` serait affichée ici. Elle présenterait une liste JSON de rendez-vous, chaque entrée incluant `id`, `patientId`, `doctorId`, `date`, et `status`, affichée dans un format clair et lisible.

5.6 Sécurité

5.6.1 Mesures de Sécurité

- **Authentification JWT** : Utilisation de jetons JWT pour sécuriser les points de terminaison API.
- **Validation des Données** : Validation des entrées pour éviter les erreurs et les attaques.
- **Protection contre les Injections** : Sanitisation des données pour empêcher les injections SQL ou autres.

— **Chiffrement des Mots de Passe** : Utilisation de bcrypt pour chiffrer les mots de passe.

— **Gestion des Sessions** : Sessions sécurisées via jetons JWT.

Troisième partie

Installation et Configuration

Chapitre 6

Guide d'Installation

6.1 Prérequis

- Node.js (v14+)
- npm (v6+)
- Python 3.8+
- MongoDB (v4+)
- Git
- Ionic CLI

6.2 Installation

6.2.1 Frontend

```
# Installer les d pendances
npm install

# Lancer l'application en mode d veloppement
ionic serve
```

6.2.2 Backend

```
# Cr er un environnement virtuel Python
python -m venv venv
source venv/bin/activate  # Linux/Mac
venv\Scripts\activate     # Windows

# Installer les d pendances Python
pip install -r backend/requirements.txt

# Lancer le serveur
python backend/app.py
```

6.3 Configuration

6.3.1 Variables d'Environnement

Créer un fichier `.env` pour le frontend et le backend :

```
# Frontend
API_URL=http://localhost:5000

# Backend (dans backend/.env)
MONGODB_URI=mongodb://localhost:27017/medical_app
JWT_SECRET=votre_secret_jwt
```

Chapitre 7

Démarrage

7.1 Environnement de Développement

```
# D marrer le backend
cd backend
python app.py

# D marrer le frontend (dans un nouveau terminal)
cd ..
ionic serve
```

7.2 Environnement de Production

```
# Build de l'application (frontend)
ionic build --prod

# Copier le contenu du dossier www/ vers votre serveur web

# Backend : Installer les d pendances de production
cd backend
pip install -r requirements.txt

# Utiliser gunicorn pour la production
pip install gunicorn
gunicorn --workers 4 app:app
```

Chapitre 8

Maintenance

8.1 Logs

— Les journaux sont disponibles dans `logs/`.
— Utiliser pour le dépannage et le suivi des performances.

8.2 Sauvegardes

— Sauvegarde automatique de la base de données recommandée.
— Configurer des sauvegardes régulières de MongoDB.

8.3 Surveillance

— Monitoring des performances via des outils comme Prometheus ou New Relic.
— Gestion proactive des erreurs grâce aux logs.

Annexe A

Annexes

A.1 Commandes Utiles

```
# Développement (frontend)
npm run start         # Démarre le serveur de développement
npm run test          # Lance les tests

# Production (frontend)
npm run build:prod    # Build de production

# Backend
python app.py         # Démarre le serveur Flask
gunicorn app:app      # Démarre en production avec Gunicorn

# Tests Backend
python -m pytest      # Lance les tests backend
```

A.2 Support

Pour toute question ou problème :

- Ouvrir une issue sur GitHub
- Contacter l'équipe de support

A.3 Licence

Ce projet est sous licence MIT.

A.4 Auteurs

- Said Mazen
- Ben Brahim Nazih
- Ben Hmida Hamza

A.5 Version

A.6 Quelques Capture

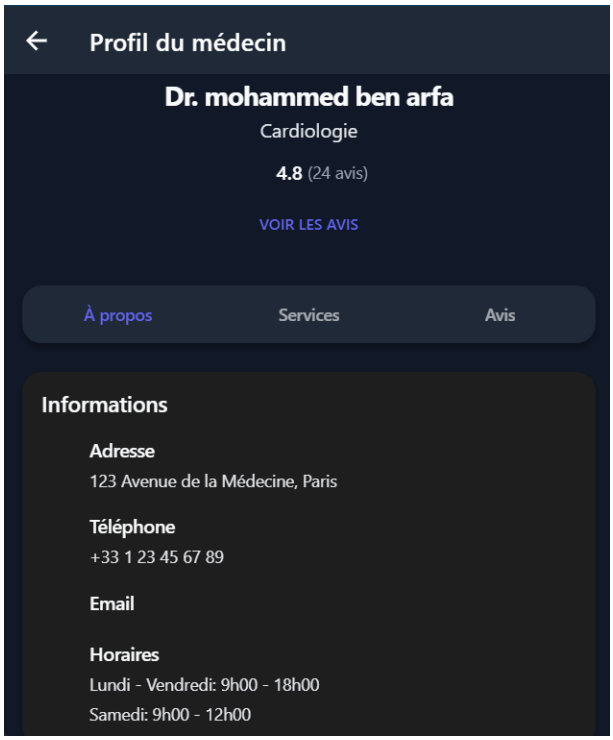

FIGURE A.1 – Profile de medecin

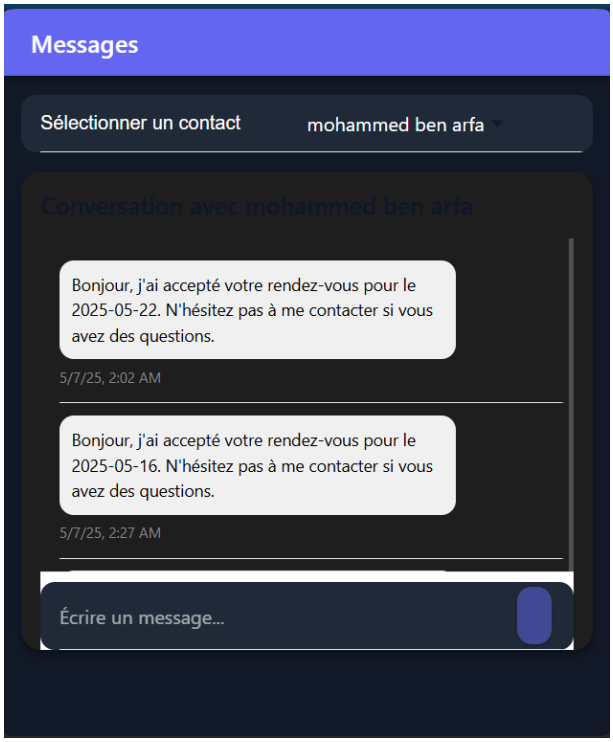

FIGURE A.2 – Messagerie

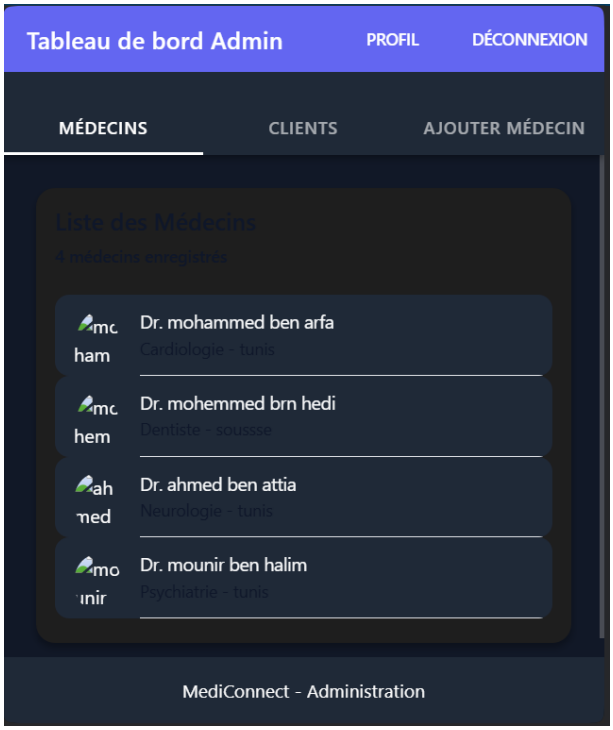

FIGURE A.3 – Liste des medecins

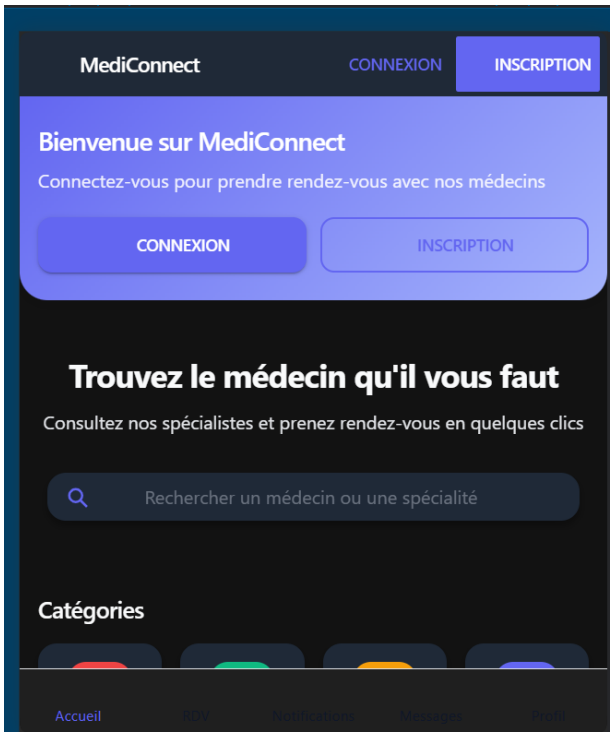

FIGURE A.4 – Home page

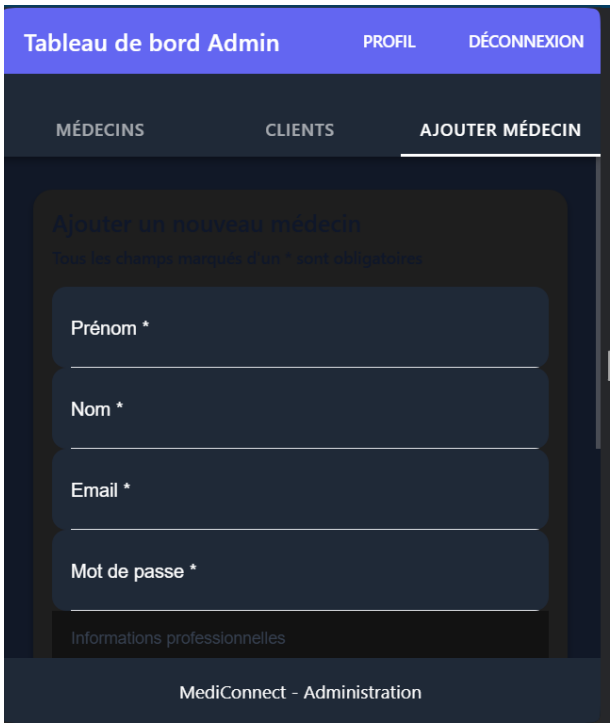

FIGURE A.5 – Ajouter medecin

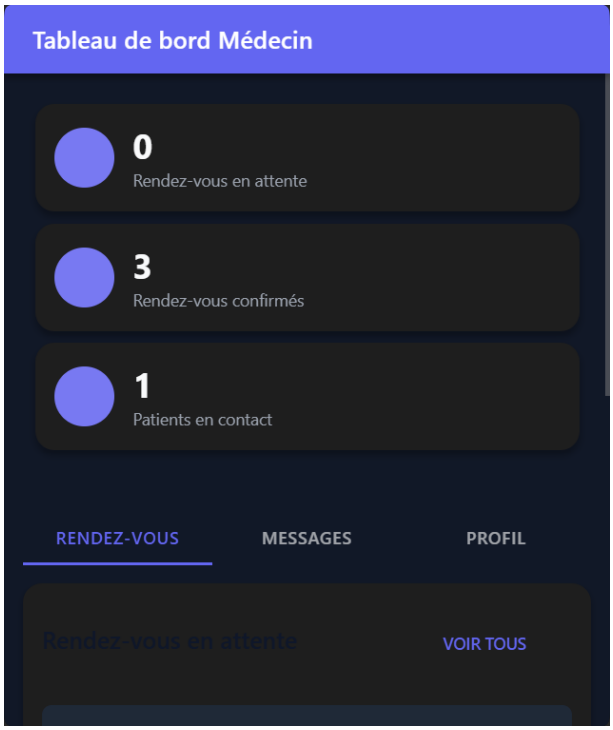

FIGURE A.6 – Tablea de bord de medecin

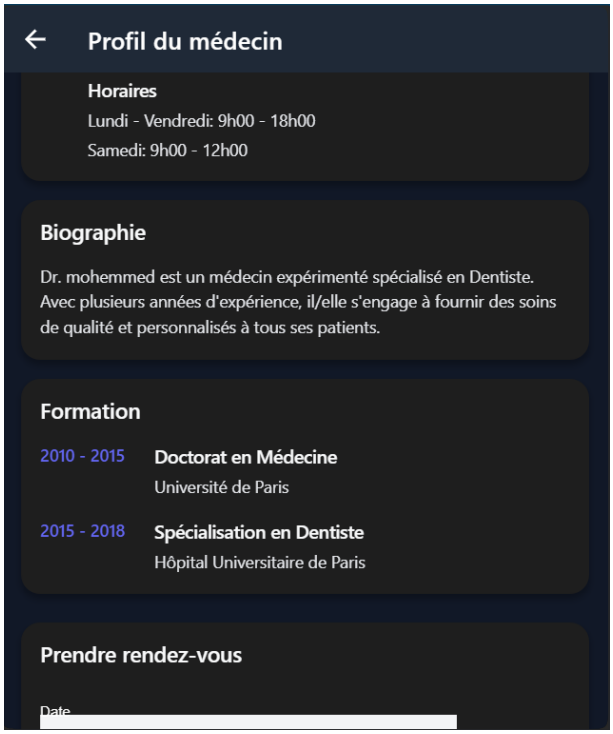

FIGURE A.7 – Profile de medecin

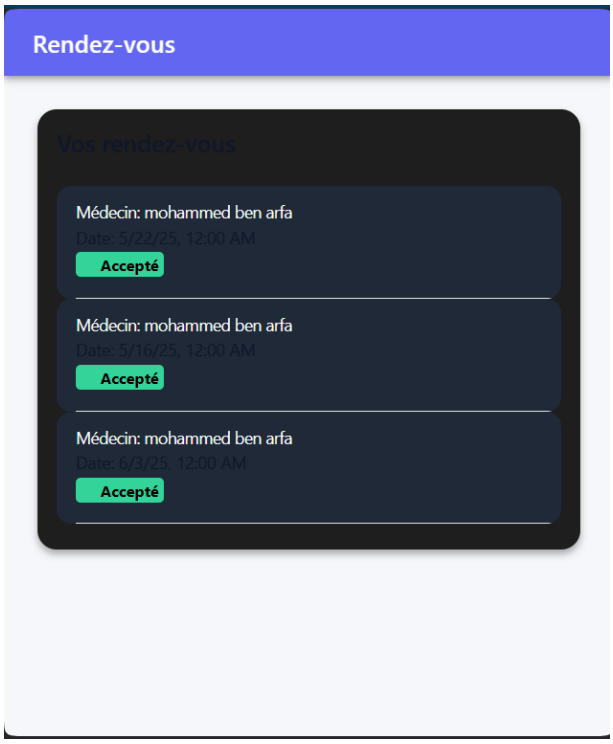

FIGURE A.8 – Rendez vous