Spécifications Techniques - HabitatsConnect

1. Architecture Générale

1.1 Architecture applicative

- **Type** : Application web monopage (SPA) avec SSR
- **Pattern** : Architecture hexagonale
- **Approche** : Micro-frontend modulaire

1.2 Technologies principales

- **Frontend Framework** : Next.js 14 (React 18)
- **Language** : TypeScript
- **Styling** : Tailwind CSS
- **State Management** : Zustand
- **Backend as a Service** : Firebase
- **Base de données** : Firestore (NoSQL)
- **Authentification** : Firebase Auth
- **Hébergement** : Vercel
- **CI/CD** : GitHub Actions + Vercel

2. Architecture Technique Détaillée

2.1 Frontend Architecture

Composants

```
src/
├── app/                    # Next.js App Router
│   ├── layout.tsx         # Layout racine
│   ├── page.tsx           # Page d'accueil
│   ├── globals.css        # Styles globaux
│   └── (routes)/          # Routes groupées
├── components/            # Composants réutilisables
│   ├── ui/                # Composants de base (shadcn/ui)
│   ├── forms/             # Composants formulaires
│   └── layouts/           # Layouts spécialisés
├── hooks/                 # Hooks personnalisés
├── lib/                   # Utilitaires
│   ├── firebase.ts        # Configuration Firebase
│   ├── utils.ts           # Fonctions utilitaires
│   └── validations.ts     # Schémas validation
├── store/                 # State management
│   ├── authStore.ts       # Authentification
│   ├── propertyStore.ts   # Propriétés
│   └── reservationStore.ts # Réservations
└── types/                 # Définitions TypeScript
```

Patterns utilisés

- **Custom Hooks** : Logique métier réutilisable
- **Compound Components** : Composants complexes modulaires
- **Render Props** : Partage logique entre composants
- **Higher-Order Components** : Injection dépendances

2.2 Backend Architecture

Firebase Services

- **Authentication** : Gestion utilisateurs et sessions
- **Firestore** : Base de données documents
- **Storage** : Stockage fichiers (photos)
- **Functions** : API serverless (optionnel)
- **Hosting** : Hébergement statique

Structure base de données

```
// Collections Firestore
interface Collections {
  users: User[];
  properties: Property[];
  reservations: Reservation[];
  messages: Message[];
  conversations: Conversation[];
  transactions: Transaction[];
}
```

2.3 API Design

Endpoints REST

```
GET    /api/properties           # Liste propriétés
GET    /api/properties/[id]      # Détail propriété
POST   /api/properties           # Créer propriété
PUT    /api/properties/[id]      # Modifier propriété
DELETE /api/properties/[id]      # Supprimer propriété

POST   /api/reservations         # Créer réservation
PUT    /api/reservations/[id]    # Modifier réservation

POST   /api/messages             # Envoyer message
GET    /api/messages/[convId]    # Messages conversation
```

WebSockets (Temps réel)

- Notifications nouvelles réservations
- Messages chat instantané
- Mises à jour statuts réservations

3. Spécifications Fonctionnelles Techniques

3.1 Gestion des utilisateurs

Authentification

- **Provider** : Firebase Auth
- **Méthodes** : Email/password, Google, Facebook
- **Sécurité** : JWT tokens, refresh automatique
- **Sessions** : Persistées localStorage

Autorisation

- **RBAC** : Role-Based Access Control
- **Rôles** : client, owner, admin
- **Permissions** : Granulaires par fonctionnalité

3.2 Gestion des propriétés

Stockage données

- **Format** : Documents Firestore
- **Indexation** : Champs recherchés indexés
- **Cache** : React Query pour optimisation

Upload médias

- **Service** : Firebase Storage
- **Formats** : JPG, PNG, WebP
- **Optimisation** : Redimensionnement automatique
- **CDN** : Distribution globale

3.3 Système de recherche

Algorithme

- **Full-text search** : Firebase search
- **Filtres** : Prix, dates, équipements, localisation
- **Tri** : Pertinence, prix, date, popularité
- **Pagination** : Curseur-based

Géolocalisation

- **API** : Google Maps Platform
- **Précision** : Adresse complète
- **Rayon recherche** : Configurable (km)

3.4 Gestion des réservations

Workflow états

Concurrence

- **Verrous optimistes** : Firestore transactions
- **Gestion conflits** : Retry automatique
- **Timeouts** : Expiration réservations temporaires

3.5 Système de paiement

Intégration Stripe

- **Mode** : Connect (marketplace)
- **Flux** : Payment Intent
- **Sécurité** : 3D Secure
- **Devises** : EUR, USD, GBP

Gestion dépôts

- **Autorisation** : Pré-autorisation carte
- **Capture** : À confirmation propriétaire
- **Remboursement** : Conditions d'annulation

4. Exigences Non Fonctionnelles

4.1 Performance

Métriques cibles

- **First Contentful Paint** : < 1.5s
- **Largest Contentful Paint** : < 2.5s
- **Time to Interactive** : < 3s
- **Core Web Vitals** : Tous "Good"

Optimisations

- **Code splitting** : Routes dynamiques
- **Image optimization** : Next.js Image component
- **Caching** : HTTP cache + service worker
- **CDN** : Distribution globale Vercel

4.2 Sécurité

Authentification

- **Hashing** : bcrypt (côté serveur)
- **JWT** : Expiration 24h, refresh tokens
- **2FA** : Optionnel pour propriétaires

Autorisation

- **Middleware** : Vérification routes protégées
- **CORS** : Configuration restrictive
- **Rate limiting** : Protection DDoS

Données sensibles

- **Chiffrement** : AES-256 au repos
- **Transmission** : TLS 1.3
- **RGPD** : Conformité complète

4.3 Disponibilité

SLA

- **Uptime** : 99.9% (8h downtime/mois)
- **RTO** : 4 heures récupération
- **RPO** : 1 heure données

Monitoring

- **Application** : Sentry error tracking
- **Performance** : Vercel Analytics
- **Infrastructure** : Firebase monitoring

4.4 Maintenabilité

Code quality

- **Linting** : ESLint + Prettier
- **Testing** : Jest + React Testing Library
- **Coverage** : > 80% lignes
- **Documentation** : JSDoc + Storybook

Déploiement

- **CI/CD** : GitHub Actions
- **Environnements** : Development, Staging, Production
- **Rollback** : Déploiement instantané
- **Feature flags** : Lancement progressif

5. Évolutivité

5.1 Architecture scalable

- **Horizontal scaling** : Serverless Firebase
- **Database sharding** : Firestore automatique
- **Caching** : Redis (futur)
- **Microservices** : API modulaires

5.2 Performance monitoring

- **Métriques** : Response times, error rates
- **Alertes** : Seuils configurables
- **Optimisation** : Continuous profiling

5.3 Migration future

- **API versioning** : REST v1, GraphQL v2
- **Database migration** : Scripts automatisés
- **Zero downtime** : Blue/green deployment