📱 TĀPE'A - Document de Spécifications Complètes pour Migration Expo/React Native

🎯 OBJECTIF

Reconstruire l'application de VTC "TĀPE'A" actuellement en React Web (PWA) vers **Expo/React Native** pour publication sur **App Store et Google Play Store**, en conservant 100% de la logique métier et du design.

🏗️ ARCHITECTURE ACTUELLE

Stack Technique

- **Frontend**: React 18 + TypeScript + Vite
- **Backend**: Express.js + TypeScript (à conserver tel quel sur le serveur)
- **Base de données**: PostgreSQL avec Drizzle ORM
- **Temps réel**: Socket.IO pour WebSockets
- **Paiement**: Stripe SDK
- **Cartes**: Google Maps API
- **Authentification**: Sessions avec cookies côté client

URL du Backend

L'app Expo communiquera avec le même backend Express déployé sur `https://[domain].replit.app`

👥 DEUX INTERFACES UTILISATEUR

1. Interface CLIENT

Accès normal à l'application pour commander des courses.

Pages à recréer:

- `Accueil` - Carte Google Maps avec géolocalisation, bouton de commande
- `CommandeOptions` - Sélection du type de course, adresses, options
- `CommandeSuccess` - Recherche de chauffeur avec animation
- `CourseEnCours` - Suivi GPS en temps réel du chauffeur
- `CourseDetails` - Détails d'une course terminée + facture PDF
- `Commandes` - Historique des courses
- `Wallet` - Solde et cartes bancaires sauvegardées
- `CartesBancaires` - Ajout/suppression de cartes Stripe
- `Profil`, `InfoPerso` - Gestion du profil
- `Tarifs` - Grille tarifaire
- `Aide`, `Support` - Chat support
- `Documents`, `Contact`, `Confidentialite`

2. Interface CHAUFFEUR

Accès via code à 6 chiffres (ex: 111 111).

Pages à recréer:

- `ChauffeurLogin` - Saisie du code à 6 chiffres
- `ChauffeurAccueil` - Carte + toggle EN LIGNE/HORS LIGNE + réception commandes
- `ChauffeurCourseEnCours` - Navigation GPS vers client puis destination
- `ChauffeurCourses` - Historique des courses effectuées
- `ChauffeurGains` - Dashboard des gains
- `ChauffeurProfil` - Profil et véhicule
- `ChauffeurDocuments` - Documents légaux
- `ChauffeurAide`, `ChauffeurSupport`

📊 MODÈLE DE DONNÉES (PostgreSQL)

Tables principales:

```
// CLIENTS (passagers)
clients {
 id: UUID (PK)
 phone: string // Format +689XXXXXX (Polynésie française)
 hashedPassword: string
 firstName: string
 lastName: string
 email: string?
 isVerified: boolean
 walletBalance: number (XPF)
 averageRating: number?
 totalRides: number
 createdAt: timestamp
}
// DRIVERS (chauffeurs)
drivers {
 id: UUID (PK)
 phone: string
 code: string // Code à 6 chiffres pour login
 firstName: string
 lastName: string
 vehicleModel: string?
 vehicleColor: string?
 vehiclePlate: string?
 isActive: boolean
 averageRating: number?
 totalRides: number
 createdAt: timestamp
}
// ORDERS (courses)
orders {
 id: UUID (PK)
 clientId: UUID (FK -> clients)
 clientName: string
 clientPhone: string
 addresses: JSON // [{type: "pickup"|"stop"|"destination", address:
string, placeId: string, lat, lng}]
 rideOption: JSON // {id, title, price, pricePerKm, ...}
 routeInfo: JSON? // {distance: number (km), duration: string}
 passengers: number
 supplements: JSON // [{id, name, icon, price, quantity}]
 totalPrice: number (XPF)
 driverEarnings: number (XPF)
```

```
 paymentMethod: "cash" | "card"
 scheduledTime: timestamp?
 isAdvanceBooking: boolean
 status: OrderStatus
 assignedDriverId: UUID?
 createdAt: timestamp
 expiresAt: timestamp
}
// STATUTS DE COMMANDE
OrderStatus = "pending" | "accepted" | "driver_arrived" | "in_progress"
|
             "completed" | "payment_pending" | "payment_confirmed" |
             "payment_failed" | "cancelled" | "expired"
// SESSIONS
client_sessions { id, clientId, expiresAt, createdAt, lastSeenAt }
driver_sessions { id, driverId, driverName, isOnline, expiresAt,
createdAt, lastSeenAt }
// STRIPE
stripe_customers { id, clientId, stripeCustomerId, createdAt }
payment_methods { id, clientId, stripePaymentMethodId, last4, brand,
expiryMonth, expiryYear, isDefault }
invoices { id, clientId, orderId, stripePaymentIntentId,
stripeInvoiceId, amount, currency, status, pdfUrl }
```

🌐 API ENDPOINTS (Backend Express existant)

Authentification Client

```
POST /api/auth/register     - Inscription client
POST /api/auth/login        - Connexion client
POST /api/auth/logout       - Déconnexion
GET  /api/auth/me           - Client authentifié actuel
POST /api/auth/forgot-password
POST /api/auth/reset-password
POST /api/auth/resend-code
```

Authentification Chauffeur

```
POST /api/driver/login      - Connexion avec code 6 chiffres
POST /api/driver-sessions   - Créer session chauffeur
PATCH /api/driver-sessions/:id/status - Mettre en ligne/hors ligne
```

Commandes

```
POST /api/orders            - Créer une commande
GET  /api/orders/pending    - Commandes en attente (chauffeurs)
GET  /api/orders/active/client - Commande active du client
GET  /api/orders/active/driver - Commande active du chauffeur
GET  /api/orders/:id        - Détails d'une commande
```

```
GET  /api/orders/:id/driver-location - Position GPS du chauffeur
(polling HTTP)
GET  /api/client/orders      - Historique client
GET  /api/driver/orders/:sessionId - Historique chauffeur
```

Stripe Paiement

```
GET  /api/stripe/publishable-key
POST /api/stripe/customer           - Créer/récupérer client Stripe
POST /api/stripe/setup-intent       - Créer SetupIntent pour ajouter
carte
POST /api/stripe/payment-method     - Sauvegarder carte après setup
GET  /api/stripe/payment-methods/:clientId - Lister les cartes
DELETE /api/stripe/payment-method/:id
POST /api/stripe/payment-method/:id/default - Définir carte par défaut
POST /api/stripe/payment-intent     - Créer PaymentIntent pour paiement
POST /api/stripe/confirm-payment    - Confirmer paiement
POST /api/stripe/generate-invoice/:orderId - Générer facture PDF
GET  /api/stripe/invoice/order/:orderId
```

Profils

```
GET  /api/client/profile
PATCH /api/client/profile
GET  /api/client/wallet
GET  /api/driver/profile/:driverId
```

⚡ ÉVÉNEMENTS WEBSOCKET (Socket.IO)

Événements émis par le CLIENT

```
socket.emit("client:join", { orderId, clientToken })
socket.emit("payment:confirm", { orderId, confirmed, role: 'client',
clientToken })
socket.emit("payment:retry", { orderId, clientToken })
socket.emit("payment:switch-cash", { orderId, clientToken })
socket.emit("ride:cancel", { orderId, role: 'client', reason?,
clientToken })
socket.emit("location:client:update", { orderId, lat, lng, clientToken
})
socket.emit("ride:join", { orderId, role: 'client', clientToken })
```

Événements émis par le CHAUFFEUR

```
socket.emit("driver:join", { sessionId })
socket.emit("driver:status", { sessionId, isOnline: true/false })
socket.emit("order:accept", { orderId, sessionId })
socket.emit("order:decline", { orderId, sessionId })
socket.emit("ride:status:update", { orderId, status, sessionId })
socket.emit("payment:confirm", { orderId, confirmed, role: 'driver',
sessionId })
```

```
socket.emit("ride:cancel", { orderId, role: 'driver', reason?,
sessionId })
socket.emit("location:driver:update", { orderId, lat, lng, heading?,
speed?, sessionId })
socket.emit("ride:join", { orderId, role: 'driver', sessionId })
```

Événements reçus (à écouter)

```
// Nouvelles commandes (chauffeurs en ligne)
socket.on("order:new", (order) => {...})
socket.on("order:expired", ({ orderId }) => {...})
// Course acceptée
socket.on("order:driver:assigned", ({ orderId, driver }) => {...})
socket.on("order:accept:success", (order) => {...})
socket.on("order:accept:error", ({ message }) => {...})
// Statut de course
socket.on("ride:status:changed", ({ orderId, status }) => {...})
socket.on("ride:cancelled", ({ orderId, cancelledBy, reason }) =>
{...})
// Paiement
socket.on("payment:status", ({ orderId, status, paymentMethod, error?
}) => {...})
socket.on("payment:retry:ready", ({ orderId }) => {...})
socket.on("payment:switched-to-cash", ({ orderId }) => {...})
// Position GPS
socket.on("location:driver", ({ orderId, lat, lng, heading, speed,
timestamp }) => {...})
```

💰 TARIFICATION (Devise: XPF - Franc Pacifique)

Options de course

```
const rideOptions = [
 {
   id: "immediate",
   title: "Taxi immédiat",
   duration: "10 - 20 min",
   capacity: "1 - 8 passagers",
   basePrice: 2300, // XPF
   pricePerKm: 150  // XPF/km
 },
 {
   id: "reservation",
   title: "Réservation à l'avance",
   duration: "45 - 1h",
   capacity: "1 - 8 passagers",
   basePrice: 2300,
   pricePerKm: 150
```

```
  },
  {
    id: "tour",
    title: "Tour de l'Île",
    duration: "45 - 1h",
    capacity: "4 - 8 passagers",
    basePrice: 30000, // Prix fixe
    pricePerKm: 0
  }
];
// Suppléments
const supplements = [
  { id: "bagages", name: "Bagages", price: 100 }, // par bagage
  { id: "encombrants", name: "Encombrants", price: 200 }
];
// Calcul du prix
totalPrice = basePrice + (distance * pricePerKm) + supplements
driverEarnings = totalPrice * 0.80 // 80% pour le chauffeur
```

🎨 DESIGN & UI

Thème couleurs

- **Primaire**: Jaune doré (#F5A623 ou similaire)
- **Fond**: Blanc / Gris clair
- **Texte**: Noir / Gris foncé
- **Accent**: Vert pour "en ligne", Rouge pour erreurs

Composants principaux

- Header avec menu hamburger glissant (drawer)
- Cartes Google Maps plein écran avec markers personnalisés
- Bottom sheets pour les options
- Cards pour afficher les informations
- Boutons arrondis style moderne
- Toggle switch pour statut en ligne/hors ligne

Mobile-first

- Design optimisé pour écran 375-428px de large
- Safe areas pour iPhone (notch, home indicator)
- Gestures pour fermer les modals

🔐 AUTHENTIFICATION

Client

- Inscription: téléphone (+689) + mot de passe + nom/prénom
- Connexion: téléphone + mot de passe
- Session via cookie `clientSessionId` (durée: 30 jours)

Chauffeur

- Connexion: code à 6 chiffres uniquement
- Session via `sessionId` en mémoire locale
- Toggle en ligne/hors ligne pour recevoir des courses

📍 GÉOLOCALISATION

Permissions requises

- `ACCESS_FINE_LOCATION` (Android)
- `NSLocationWhenInUseUsageDescription` (iOS)
- `NSLocationAlwaysAndWhenInUseUsageDescription` (iOS) pour chauffeurs

Fonctionnalités

- Position actuelle du client sur la carte
- Suivi GPS continu du chauffeur (envoi toutes les 2-3 secondes)
- Calcul d'itinéraire via Google Directions API
- Affichage du trajet sur la carte

🔔 NOTIFICATIONS PUSH

Pour Expo/React Native

- Utiliser **expo-notifications** + **Firebase Cloud Messaging**
- Le backend supporte déjà VAPID pour web push
- Ajouter endpoints pour tokens FCM si nécessaire

Événements à notifier

- Nouvelle commande disponible (chauffeurs)
- Chauffeur assigné (client)
- Chauffeur arrivé (client)
- Course terminée (client/chauffeur)
- Paiement confirmé/échoué

📦 DÉPENDANCES EXPO SUGGÉRÉES

```
{
 "expo": "~50.x",
 "expo-location": "~16.x",
 "expo-notifications": "~0.27.x",
 "react-native-maps": "1.10.x",
 "@stripe/stripe-react-native": "0.35.x",
 "socket.io-client": "^4.7.x",
 "@tanstack/react-query": "^5.x",
 "expo-secure-store": "~12.x", // Pour stocker les sessions
 "expo-router": "~3.x", // Navigation
 "nativewind": "^2.x" // Tailwind pour RN (optionnel)
}
```

⚠️ POINTS D'ATTENTION

1. **Sessions**: Remplacer les cookies par `SecureStore` ou `AsyncStorage`
2. **Stripe**: Utiliser `@stripe/stripe-react-native` au lieu de `@stripe/react-stripe-js`
3. **WebSocket**: Socket.IO fonctionne tel quel en React Native
4. **Google Maps**: Utiliser `react-native-maps` avec `PROVIDER_GOOGLE`
5. **Permissions**: Demander les permissions GPS au bon moment
6. **Background location**: Pour les chauffeurs, implémenter le suivi en arrière-plan
7. **Deep linking**: Configurer pour les liens de paiement Stripe

---

🚀 ORDRE DE DÉVELOPPEMENT SUGGÉRÉ

1. **Configuration Expo** + navigation de base
2. **Authentification** (client puis chauffeur)
3. **Accueil avec carte** Google Maps
4. **Flux de commande** complet (client)
5. **Interface chauffeur** avec réception de courses
6. **Suivi GPS temps réel** (WebSocket)
7. **Intégration Stripe** paiement + factures
8. **Notifications push** FCM
9. **Polish UI** + animations
10. **Tests** + publication stores

📁 FICHIERS SOURCE CLÉS À ÉTUDIER

Pour comprendre la logique métier existante :

- `server/routes.ts` - Toute l'API REST + WebSocket
- `shared/schema.ts` - Modèle de données complet
- `client/src/pages/CommandeOptions.tsx` - Flux de réservation
- `client/src/pages/CourseEnCours.tsx` - Suivi temps réel client
- `client/src/pages/ChauffeurCourseEnCours.tsx` - Interface chauffeur
- `client/src/lib/socket.ts` - Configuration Socket.IO
- `client/src/hooks/use-auth.tsx` - Gestion authentification