

Arquitectura para Mobile Apps Nativas - Fase 3

Visión General

Este documento describe la arquitectura propuesta para el desarrollo de aplicaciones móviles nativas (iOS y Android) que complementen la plataforma web de INMOVA.

Objetivos Estratégicos

1. **Experiencia Mobile-First:** Proporcionar experiencia optimizada para dispositivos móviles
2. **Funcionalidades Offline:** Permitir uso básico sin conexión a internet
3. **Notificaciones Push:** Engagement en tiempo real con usuarios
4. **Integración Nativa:** Aprovechar capacidades del dispositivo (cámara, GPS, etc.)
5. **Performance:** Experiencia rápida y fluida

Aplicaciones Propuestas

1. INMOVA Manager (Gestores)

Público Objetivo: Administradores y gestores de propiedades

Funcionalidades Principales:

- Dashboard con métricas clave
- Gestión de solicitudes de mantenimiento
- Aprobación de gastos
- Comunicación con inquilinos y propietarios
- Visitas e inspecciones con geolocalización
- Captura y subida de fotos/documentos
- Firma digital móvil

2. INMOVA Tenant (Inquilinos)

Público Objetivo: Inquilinos de propiedades gestionadas

Funcionalidades Principales:

- Portal del inquilino
- Solicitudes de mantenimiento con fotos
- Pagos móviles
- Documentos y contratos
- Mensajería con gestor
- Notificaciones push
- Guía de la comunidad

3. INMOVA Owner (Propietarios)

Público Objetivo: Propietarios de inmuebles

Funcionalidades Principales:

- Dashboard financiero
- Reportes y estadísticas
- Vista de propiedades y unidades
- Documentos financieros
- Notificaciones importantes
- Comunicación con gestor

Stack Tecnológico

Opción 1: React Native (Recomendada)

Ventajas:

- Código compartido entre iOS y Android (70-90%)
- Ecosistema maduro y grande comunidad
- Performance cercano a nativo
- Integración con la codebase web existente (React)
- Reutilización de lógica de negocio
- Hot reload para desarrollo rápido

Desventajas:

- Algunas limitaciones en funcionalidades muy específicas de plataforma
- Dependencia de puentes nativos para funcionalidades avanzadas

Librerías Clave:

```
{
  "react-native": "^0.73.0",
  "@react-navigation/native": "^6.1.0",
  "@react-navigation/stack": "^6.3.0",
  "react-native-maps": "^1.8.0",
  "react-native-camera": "^4.2.0",
  "react-native-push-notification": "^8.1.0",
  "@react-native-async-storage/async-storage": "^1.19.0",
  "react-native-biometrics": "^3.0.0",
  "react-native-document-picker": "^9.0.0",
  "react-native-pdf": "^6.7.0",
  "react-native-signature-capture": "^0.4.0",
  "axios": "^1.6.0",
  "react-query": "^3.39.0"
}
```

Opción 2: Flutter

Ventajas:

- Excelente performance
- UI consistente entre plataformas
- Hot reload ultra-rápido
- Gran conjunto de widgets

Desventajas:

- Lenguaje diferente (Dart) de la web (TypeScript/JavaScript)
- Menor reutilización de código con la web
- Curva de aprendizaje para el equipo

Opción 3: Desarrollo Nativo (iOS + Android)

Ventajas:

- Máxima performance
- Acceso completo a APIs de plataforma
- Mejor experiencia de usuario nativa

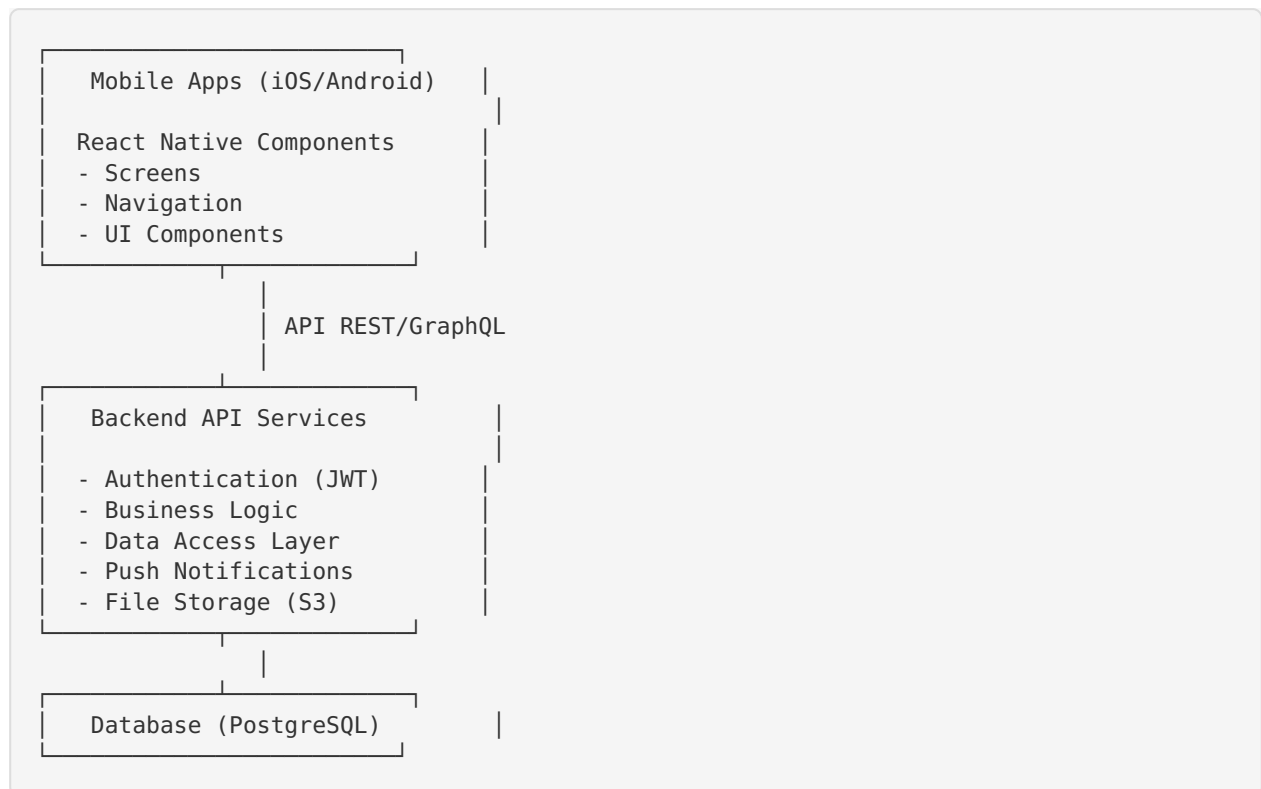
Desventajas:

- Doble esfuerzo de desarrollo
- Mayor costo y tiempo
- Mantenimiento de dos codebases separadas

Recomendación: React Native por balance costo/beneficio y sinergia con stack actual.

Arquitectura Técnica

Arquitectura de Alto Nivel



Patrón de Arquitectura: MVVM (Model-View-ViewModel)

```
// Model
interface Property {
  id: string;
  nombre: string;
  direccion: string;
  imagenes: string[];
}

// ViewModel
class PropertiesViewModel {
  private api: ApiService;
  properties: Property[] = [];
  loading: boolean = false;
  error: string | null = null;

  async fetchProperties() {
    this.loading = true;
    try {
      this.properties = await this.api.getProperties();
    } catch (error) {
      this.error = error.message;
    } finally {
      this.loading = false;
    }
  }
}

// View (React Native Component)
const PropertiesScreen = () => {
  const viewModel = usePropertiesViewModel();

  useEffect(() => {
    viewModel.fetchProperties();
  }, []);

  if (viewModel.loading) return <LoadingSpinner />;
  if (viewModel.error) return <ErrorMessage message={viewModel.error} />;

  return (
    <FlatList
      data={viewModel.properties}
      renderItem={({ item }) => <PropertyCard property={item} />}
    />
  );
};
```

Estructura de Proyecto

```

innova-mobile/
├── src/
│   ├── screens/           # Pantallas de la app
│   │   ├── auth/
│   │   ├── dashboard/
│   │   ├── properties/
│   │   ├── maintenance/
│   │   └── settings/
│   ├── components/       # Componentes reutilizables
│   │   ├── common/
│   │   └── domain/
│   ├── navigation/       # Configuración de navegación
│   ├── services/         # Servicios (API, Storage, etc.)
│   │   ├── api/
│   │   ├── auth/
│   │   ├── storage/
│   │   └── notifications/
│   ├── models/           # Modelos de datos
│   ├── viewmodels/       # Lógica de negocio
│   ├── utils/            # Utilidades
│   ├── constants/        # Constantes
│   └── theme/            # Tema y estilos
├── android/              # Código nativo Android
├── ios/                  # Código nativo iOS
├── __tests__/            # Tests
├── package.json
└── tsconfig.json
  
```

Funcionalidades Clave

1. Autenticación y Seguridad

Implementación:

```
// services/auth/AuthService.ts
export class AuthService {
  private readonly TOKEN_KEY = '@inmova:auth_token';
  private readonly REFRESH_TOKEN_KEY = '@inmova:refresh_token';

  async login(email: string, password: string) {
    const response = await api.post('/auth/login', { email, password });
    await this.storeTokens(response.data.token, response.data.refreshToken);
    return response.data;
  }

  async biometricLogin() {
    const hasCredentials = await Keychain.hasInternetCredentials('inmova');
    if (!hasCredentials) return false;

    const biometricAuth = await ReactNativeBiometrics.simplePrompt({
      promptMessage: 'Autenticar con huella/Face ID',
    });

    if (biometricAuth.success) {
      const credentials = await Keychain.getInternetCredentials('inmova');
      return this.login(credentials.username, credentials.password);
    }
  }

  private async storeTokens(token: string, refreshToken: string) {
    await AsyncStorage.multiSet([
      [this.TOKEN_KEY, token],
      [this.REFRESH_TOKEN_KEY, refreshToken],
    ]);
  }
}
```

2. Sincronización Offline

Estrategia:

- Cache local con AsyncStorage/SQLite
- Cola de sincronización para operaciones pendientes
- Resolución de conflictos

```
// services/sync/SyncService.ts
export class SyncService {
  private queue: SyncQueue;

  async queueOperation(operation: Operation) {
    await this.queue.add(operation);
    if (await NetInfo.fetch().isConnected) {
      await this.processPendingOperations();
    }
  }

  async processPendingOperations() {
    const pending = await this.queue.getPending();
    for (const operation of pending) {
      try {
        await this.executeOperation(operation);
        await this.queue.markCompleted(operation.id);
      } catch (error) {
        await this.queue.markFailed(operation.id, error);
      }
    }
  }
}
```

3. Notificaciones Push

Configuración:

```
// services/notifications/PushNotificationService.ts
import PushNotification from 'react-native-push-notification';
import messaging from '@react-native-firebase/messaging';

export class PushNotificationService {
  async initialize() {
    // Request permission (iOS)
    const authStatus = await messaging().requestPermission();
    const enabled = authStatus === messaging.AuthorizationStatus.AUTHORIZED;

    if (enabled) {
      // Get FCM token
      const token = await messaging().getToken();
      await this.registerDeviceToken(token);

      // Handle foreground notifications
      messaging().onMessage(async remoteMessage => {
        this.showNotification(remoteMessage);
      });

      // Handle background notifications
      messaging().setBackgroundMessageHandler(async remoteMessage => {
        console.log('Background message:', remoteMessage);
      });
    }
  }

  private async registerDeviceToken(token: string) {
    await api.post('/api/push/register', {
      token,
      platform: Platform.OS,
    });
  }
}
```

4. Cámara y Galería

```
// components/ImageCapture.tsx
import { Camera } from 'react-native-camera';
import ImagePicker from 'react-native-image-picker';

export const ImageCapture = ({ onImageSelected }) => {
  const takePhoto = async () => {
    const photo = await camera.takePictureAsync({
      quality: 0.8,
      base64: false,
    });
    await uploadPhoto(photo.uri);
  };

  const selectFromGallery = () => {
    ImagePicker.launchImageLibrary({
      mediaType: 'photo',
      quality: 0.8,
    }, response => {
      if (!response.didCancel && response.assets) {
        onImageSelected(response.assets[0]);
      }
    });
  };

  return (
    <View>
      <Button onPress={takePhoto}>Tomar Foto</Button>
      <Button onPress={selectFromGallery}>Seleccionar de Galería</Button>
    </View>
  );
};
```

5. Geolocalización

```
// services/location/LocationService.ts
import Geolocation from '@react-native-community/geolocation';

export class LocationService {
  async getCurrentLocation(): Promise<Coordinates> {
    return new Promise((resolve, reject) => {
      Geolocation.getCurrentPosition(
        position => resolve({
          latitude: position.coords.latitude,
          longitude: position.coords.longitude,
        }),
        error => reject(error),
        { enableHighAccuracy: true, timeout: 20000 }
      );
    });
  }

  watchLocation(callback: (coords: Coordinates) => void) {
    return Geolocation.watchPosition(
      position => callback({
        latitude: position.coords.latitude,
        longitude: position.coords.longitude,
      }),
      error => console.error(error),
      { enableHighAccuracy: true, distanceFilter: 10 }
    );
  }
}
```

API Backend Requerida

Endpoints Móviles Específicos

```
// Autenticación
POST /api/mobile/auth/login
POST /api/mobile/auth/refresh
POST /api/mobile/auth/logout

// Perfil
GET /api/mobile/profile
PUT /api/mobile/profile

// Dashboard
GET /api/mobile/dashboard
GET /api/mobile/dashboard/widgets

// Propiedades
GET /api/mobile/properties
GET /api/mobile/properties/:id
GET /api/mobile/properties/:id/units

// Mantenimiento
GET /api/mobile/maintenance
POST /api/mobile/maintenance
PUT /api/mobile/maintenance/:id
POST /api/mobile/maintenance/:id/photos

// Notificaciones
GET /api/mobile/notifications
PUT /api/mobile/notifications/:id/read
POST /api/mobile/push/register
DELETE /api/mobile/push/unregister

// Pagos
GET /api/mobile/payments
POST /api/mobile/payments/initiate

// Documentos
GET /api/mobile/documents
GET /api/mobile/documents/:id/download

// Sync
GET /api/mobile/sync/changes?since=timestamp
POST /api/mobile/sync/batch
```

Testing

Estrategia de Testing

1. **Unit Tests:** Jest + React Native Testing Library
2. **Integration Tests:** Detox (end-to-end)
3. **Manual Testing:** TestFlight (iOS) + Firebase App Distribution (Android)

```
// __tests__/screens/PropertiesScreen.test.tsx
import { render, waitFor } from '@testing-library/react-native';
import { PropertiesScreen } from '@screens/properties';

describe('PropertiesScreen', () => {
  it('should load and display properties', async () => {
    const { getByText } = render(<PropertiesScreen />);

    await waitFor(() => {
      expect(getByText('Edificio Central')).toBeTruthy();
    });
  });
});
```

Despliegue

CI/CD Pipeline

```
# .github/workflows/mobile-deploy.yml
name: Mobile Deploy

on:
  push:
    branches: [main]

jobs:
  ios:
    runs-on: macos-latest
    steps:
      - uses: actions/checkout@v3
      - uses: ruby/setup-ruby@v1
      - run: bundle install
      - run: bundle exec fastlane ios beta

  android:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v3
      - uses: actions/setup-java@v3
      - run: ./gradlew bundleRelease
      - run: fastlane android beta
```

App Store & Google Play

iOS (App Store):

1. Apple Developer Program (€99/año)
2. Certificados y provisioning profiles
3. App Store Connect
4. Review process (2-3 días)

Android (Google Play):

1. Google Play Console (€25 one-time)
2. Signing key
3. Play Store listing
4. Review process (1-2 días)

Roadmap de Implementación

Fase 1: Fundación (Mes 1-2)

- Setup del proyecto React Native
- Configuración de navegación
- Autenticación básica
- Integración con API backend
- UI components library

Fase 2: Funcionalidades Core (Mes 3-4)

- Dashboard
- Listado de propiedades
- Mantenimiento
- Notificaciones push
- Cámara y fotos

Fase 3: Funcionalidades Avanzadas (Mes 5-6)

- Modo offline
- Firma digital
- Pagos móviles
- Geolocalización
- Biometría

Fase 4: Testing y Lanzamiento (Mes 7-8)

- Beta testing
- Optimización de performance
- Publicación en stores
- Marketing y onboarding

Métricas de Éxito

1. **Adopción:** % de usuarios que descargan la app
2. **Engagement:** DAU/MAU (Daily/Monthly Active Users)
3. **Retención:** % de usuarios que continúan usando después de 30 días
4. **Performance:** Tiempo de carga, crash rate
5. **Satisfacción:** Rating en stores, reviews

Conclusiones

La implementación de apps móviles nativas con React Native proporciona:

1. Experiencia optimizada para usuarios móviles
2. Acceso a capacidades nativas del dispositivo
3. Mayor engagement mediante push notifications
4. Funcionalidad offline
5. Diferenciación competitiva

Se recomienda comenzar con INMOVA Manager (gestores) por ser el público principal y luego expandir a inquilinos y propietarios.