📚 Documentación Técnica - Sistema de Encuestas Davivienda Mobile

Tabla de Contenidos

---

1. Arquitectura del Sistema

1.1 Clean Architecture en Capas

El proyecto implementa una arquitectura limpia dividida en tres capas principales:

```
┌─────────────────────────────────────────┐
│      Presentation Layer                 │
│  (UI, Components, Navigation)           │
```

```
├──────────────────────────────────────┤
│         Data Layer                   │
│  (DataSources, DTOs, Mappers)        │
├──────────────────────────────────────┤
│         Domain Layer                 │
│  (Entities, Business Logic)          │
└──────────────────────────────────────┘
```

1.1.1 Capa de Dominio (Core)

Ubicación: `src/core/domain/`

- **Responsabilidad**: Define las entidades del negocio
- **Independencia**: No depende de frameworks externos
- **Entidades principales**:
- `Survey` : Modelo de encuesta
- `Question` : Modelo de pregunta
- `QuestionType` : Tipos de preguntas permitidos

```
// Ejemplo de entidad
export interface Survey {
  id: string;
  title: string;
  description: string;
  questions: Question[];
  createdBy: string;
  isPublished: boolean;
  expiresAt: Date | null;
  createdAt: Date;
  updatedAt: Date;
}
```

1.1.2 Capa de Datos (Data)

Ubicación: `src/data/`

- **DataSources**: Implementaciones concretas de acceso a datos
- `auth.datasource.ts` : Operaciones de autenticación
- `survey.datasource.ts` : CRUD de encuestas y respuestas

- **DTOs (Data Transfer Objects)**: Modelos para comunicación con API
- `auth.dto.ts`: Modelos de autenticación
- `survey.dto.ts`: Modelos de encuestas

Responsabilidades:
- Comunicación con APIs externas
- Transformación de DTOs a entidades de dominio
- Caché y persistencia local

1.1.3 Capa de Presentación (Presentation)

Ubicación: `src/presentation/`

- **Screens**: Pantallas organizadas por funcionalidad
- **Components**: Componentes reutilizables
- **Stores**: Estado global con Zustand
- **Navigation**: Configuración de navegación
- **Hooks**: Lógica reutilizable

1.2 Organización por Funcionalidad

Cada pantalla sigue el patrón de organización por carpeta:

```
screens/
├── login/
│   ├── LoginScreen.tsx          # Lógica y UI del componente
│   └── LoginScreen.styles.ts    # Estilos separados
```

Ventajas:
- Alta cohesión
- Bajo acoplamiento
- Fácil mantenimiento
- Escalabilidad

---

2. Patrones de Diseño

2.1 Repository Pattern

Los DataSources actúan como repositorios, abstrayendo la lógica de acceso a datos:

```
class SurveyDataSource {
  async getMySurveys(): Promise<SurveyResponse[]> {
    const { data } = await httpClient.get('/surveys');
    return data;
  }
}

export const surveyDataSource = new SurveyDataSource();
```

2.2 Singleton Pattern

Los stores de Zustand y DataSources se implementan como singletons:

```
export const surveyDataSource = new SurveyDataSource();
```

2.3 Observer Pattern

Zustand implementa el patrón Observer para la gestión de estado:

```
const useSurveyStore = create<SurveyStore>((set, get) => ({
  surveys: [],
  setSurveys: (surveys) => set({ surveys }),
}));
```

2.4 Compound Component Pattern

Componentes como CustomInput implementan este patrón:

```
<CustomInput
  placeholder="Email"
  value={email}
```

```
  onChangeText={setEmail}
  error={errors.email}
/>
```

2.5 Custom Hooks Pattern

Encapsulación de lógica reutilizable:

```
export const useAuth = () => {
  const login = useAuthStore((state) => state.login);
  const logout = useAuthStore((state) => state.logout);
  const user = useAuthStore((state) => state.user);
  const isLoading = useAuthStore((state) => state.isLoading);

  return { login, logout, user, isLoading };
};
```

---

3. Gestión de Estado

3.1 Zustand como State Manager

Ventajas sobre Redux:
- Menos boilerplate
- API más simple
- TypeScript nativo
- Mejor performance
- No requiere Context

3.2 Stores Implementados

3.2.1 Auth Store

Ubicación: `src/presentation/stores/authStore.ts`

```
interface AuthStore {
  user: User | null;
  token: string | null;
```

```
  isLoading: boolean;
  login: (email: string, password: string) => Promise<void>;
  register: (name: string, email: string, password: string) => Promise<void>;
  logout: () => Promise<void>;
}
```

Responsabilidades:
- Gestión de sesión de usuario
- Almacenamiento de token
- Estado de autenticación

3.2.2 Survey Store

Ubicación: `src/presentation/stores/surveyStore.ts`

```
interface SurveyStore {
  surveys: Survey[];
  publishedSurveys: Survey[];
  currentSurvey: Survey | null;
  loading: boolean;

  // CRUD operations
  refreshMySurveys: () => Promise<void>;
  refreshPublishedSurveys: () => Promise<void>;
  createSurvey: (data: CreateSurveyDto) => Promise<Survey>;
  updateSurvey: (id: string, data: UpdateSurveyDto) => Promise<Survey>;
  deleteSurvey: (id: string) => Promise<void>;
  publishSurvey: (id: string) => Promise<Survey>;

  // Questions
  addQuestion: (surveyId: string, question: CreateQuestionDto) => Promise<Survey>;
  updateQuestion: (surveyId: string, questionId: string, data: UpdateQuestionDto) => Promise<Sur
  deleteQuestion: (surveyId: string, questionId: string) => Promise<Survey>;
}
```

Características especiales:
- Actualización automática cada 5 segundos
- Sincronización silenciosa
- Optimistic updates

3.2.3 Toast Store

Ubicación: `src/presentation/stores/toastStore.ts`

```
interface ToastStore {
  message: string;
  type: 'success' | 'error' | 'info' | 'warning';
  visible: boolean;
  showToast: (message: string, type: ToastType) => void;
  hideToast: () => void;
}
```

Responsabilidad: Gestión de notificaciones al usuario

3.3 Persistencia de Estado

El token de autenticación se persiste usando AsyncStorage:

```
const saveToken = async (token: string) => {
  await AsyncStorage.setItem('token', token);
};

const getToken = async (): Promise<string | null> => {
  return await AsyncStorage.getItem('token');
};
```

---

4. Navegación

4.1 Estructura de Navegación

```
App
├── AuthNavigator (Stack)
│   ├── Login
│   └── Register
│
└── DrawerNavigator (Drawer)
    └── MainNavigator (Stack)
```

```
        ├── MySurveys
        ├── SurveyEditor
        ├── SurveyResults
        └── SurveyResponse
```

4.2 Navegadores Implementados

4.2.1 AuthNavigator

Tipo: Stack Navigator
Ubicación: `src/presentation/navigation/AuthNavigator.tsx`

```
<Stack.Navigator screenOptions={{ headerShown: false, animation: 'fade' }}>
  <Stack.Screen name="Login" component={LoginScreen} />
  <Stack.Screen name="Register" component={RegisterScreen} />
</Stack.Navigator>
```

4.2.2 MainNavigator

Tipo: Stack Navigator
Ubicación: `src/presentation/navigation/MainNavigator.tsx`

Navegación principal de la aplicación con animación slide_from_right.

4.2.3 DrawerNavigator

Tipo: Drawer Navigator
Ubicación: `src/presentation/navigation/DrawerNavigator.tsx`

Menú lateral con opciones de navegación y logout.

4.3 Type Safety en Navegación

```
export type AuthStackParamList = {
  Login: undefined;
  Register: undefined;
};

export type MainStackParamList = {
  MySurveys: undefined;
  SurveyEditor: { surveyId: string };
```

```
  SurveyResults: { surveyId: string };
  SurveyResponse: { surveyId: string };
};
```

---

5. Comunicación con API

5.1 Cliente HTTP

Ubicación: `src/shared/utils/http.ts`

```
export const httpClient = axios.create({
  baseURL: 'http://localhost:3000/api',
  timeout: 10000,
  headers: {
    'Content-Type': 'application/json',
  },
});
```

5.2 Interceptores

Request Interceptor

Añade el token de autenticación a cada petición:

```
httpClient.interceptors.request.use(
  async (config) => {
    const token = await AsyncStorage.getItem('token');
    if (token) {
      config.headers.Authorization = `Bearer ${token}`;
    }
    return config;
  },
  (error) => Promise.reject(error)
);
```

Response Interceptor

Maneja errores globalmente:

```
httpClient.interceptors.response.use(
  (response) => response,
  async (error) => {
    if (error.response?.status === 401) {
      // Limpiar sesión y redirigir a login
      await useAuthStore.getState().logout();
    }
    return Promise.reject(error);
  }
);
```

5.3 Endpoints y Contratos

Autenticación

POST /api/auth/register

```
Request:
{
  name: string;
  email: string;
  password: string;
}

Response:
{
  user: {
    id: string;
    name: string;
    email: string;
  };
  token: string;
}
```

POST /api/auth/login

```
Request:
{
  email: string;
  password: string;
}

Response:
{
  user: {
    id: string;
    name: string;
    email: string;
  };
  token: string;
}
```

Encuestas

GET /api/surveys

```
Response: Survey[]
```

POST /api/surveys

```
Request:
{
  title: string;
  description: string;
  expiresAt?: Date;
}

Response: Survey
```

PUT /api/surveys/:id

```
Request:
{
  title?: string;
  description?: string;
```

```
  expiresAt?: Date;
}

Response: Survey
```

POST /api/surveys/:id/publish

```
Response: Survey
```

Preguntas

POST /api/surveys/:surveyId/questions

```
Request:
{
  title: string;
  type: 'TEXT' | 'MULTIPLE_CHOICE' | 'CHECKBOX' | 'DROPDOWN' | 'SCALE';
  required: boolean;
  order: number;
  options?: string[];
}

Response: Survey
```

PUT /api/surveys/:surveyId/questions/:questionId

```
Request: Partial<Question>
Response: Survey
```

Respuestas

POST /api/surveys/:id/responses

```
Request:
{
  respondentEmail: string;
  answers: Array<{
    questionId: string;
    value: string[];
```

```
  }>;
}

Response: Response
```

GET /api/surveys/:id/responses

```
Response: ResponseDetailResponse[]
```

---

6. Autenticación y Seguridad

6.1 Flujo de Autenticación

```
1. Usuario ingresa credenciales
   ↓
2. App envía POST /api/auth/login
   ↓
3. Backend valida y retorna JWT
   ↓
4. App guarda token en AsyncStorage
   ↓
5. App actualiza estado global (Zustand)
   ↓
6. Navegación redirige a pantalla principal
```

6.2 Gestión de Tokens

Almacenamiento:
- Token almacenado en AsyncStorage (persistente)
- Token en memoria en Zustand (runtime)

Ciclo de vida:

```
// Login
await AsyncStorage.setItem('token', token);
set({ token, user, isLoading: false });
```

```
// Logout
await AsyncStorage.removeItem('token');
set({ token: null, user: null });

// Refresh
const token = await AsyncStorage.getItem('token');
if (token) {
  // Validar token y restaurar sesión
}
```

6.3 Seguridad

Medidas implementadas:
1. **Token en headers**: Authorization: Bearer {token}
2. **Timeout en peticiones**: 10 segundos
3. **Validación de inputs**: Validaciones en cliente
4. **HTTPS**: En producción
5. **No almacenar contraseñas**: Solo tokens

Validaciones del lado del cliente:

```
export const validateEmail = (email: string): boolean => {
  const emailRegex = /^[^\s@]+@[^\s@]+\.[^\s@]+$/;
  return emailRegex.test(email);
};

export const validatePassword = (password: string): boolean => {
  return password.length >= 6;
};

export const validateName = (name: string): boolean => {
  return name.trim().length >= 2;
};
```

---

7. Modelos de Datos

7.1 Entidades de Dominio

Survey

```
export interface Survey {
  id: string;
  title: string;
  description: string;
  questions: Question[];
  createdBy: string;
  isPublished: boolean;
  expiresAt: Date | null;
  createdAt: Date;
  updatedAt: Date;
}
```

Question

```
export interface Question {
  id: string;
  title: string;
  type: QuestionType;
  required: boolean;
  order: number;
  options?: string[];
  imageUrl?: string;
}

export type QuestionType =
  | 'text'
  | 'textarea'
  | 'multiple-choice'
  | 'checkbox'
  | 'dropdown'
  | 'scale';
```

User

```
export interface User {
  id: string;
  name: string;
  email: string;
}
```

7.2 DTOs (Data Transfer Objects)

CreateSurveyDto

```
export interface CreateSurveyDto {
  title: string;
  description: string;
  expiresAt?: Date;
}
```

CreateQuestionDto

```
export interface CreateQuestionDto {
  title: string;
  type: string;
  required: boolean;
  order: number;
  options?: string[];
}
```

SubmitResponseDto

```
export interface SubmitResponseDto {
  respondentEmail: string;
  answers: Array<{
    questionId: string;
    value: string[];
  }>;
}
```

7.3 Transformación de Datos

Backend → Frontend:

```
const mapTypeFromBackend = (type: string): QuestionType => {
  const mapping: Record<string, QuestionType> = {
    'TEXT': 'text',
    'MULTIPLE_CHOICE': 'multiple-choice',
    'CHECKBOX': 'checkbox',
    'DROPDOWN': 'dropdown',
    'SCALE': 'scale',
  };
  return mapping[type] || 'text';
};
```

Frontend → Backend:

```
const mapTypeToBackend = (type: QuestionType): string => {
  const mapping: Record<QuestionType, string> = {
    'text': 'TEXT',
    'textarea': 'TEXT',
    'multiple-choice': 'MULTIPLE_CHOICE',
    'checkbox': 'CHECKBOX',
    'dropdown': 'DROPDOWN',
    'scale': 'SCALE',
  };
  return mapping[type] || 'TEXT';
};
```

---

8. Flujos de Usuario

8.1 Flujo de Creación de Encuesta

```
1. Usuario navega a SurveyEditor con surveyId='new'
   ↓
2. Usuario completa información básica (título, descripción)
```

```
   ↓
3. Usuario presiona "Guardar"
   ↓
4. App crea encuesta en estado borrador
   ↓
5. Usuario agrega preguntas una por una
   ↓
6. Para cada pregunta:
   - Define tipo
   - Configura opciones (si aplica)
   - Marca como obligatoria (opcional)
   ↓
7. Usuario presiona "Publicar"
   ↓
8. App valida:
   - Al menos 1 pregunta
   - Preguntas de opción múltiple tienen ≥2 opciones
   ↓
9. App publica encuesta
   ↓
10. Usuario puede compartir enlace
```

8.2 Flujo de Respuesta a Encuesta

```
1. Usuario accede a SurveyResponse con surveyId
   ↓
2. App carga encuesta publicada
   ↓
3. App valida:
   - Encuesta está publicada
   - No ha expirado
   ↓
4. Usuario completa respuestas
   ↓
5. Usuario presiona "Enviar"
   ↓
6. App valida campos obligatorios
   ↓
7. App formatea respuestas según tipo de pregunta
   ↓
```

```
8. App envía respuestas al backend
   ↓
9. App muestra confirmación
   ↓
10. Usuario es redirigido
```

8.3 Flujo de Visualización de Resultados

```
1. Usuario navega a SurveyResults con surveyId
   ↓
2. App carga encuesta y respuestas
   ↓
3. Para cada pregunta:
   - Si es tipo texto: muestra lista de respuestas
   - Si tiene opciones: genera gráfico
   ↓
4. App decide tipo de gráfico:
   - 2 opciones: Gráfico de barras
   - >2 opciones: Gráfico circular
   ↓
5. App actualiza resultados cada 5 segundos
   ↓
6. Usuario puede ver estadísticas detalladas
```

---

9. Optimizaciones

9.1 Renderizado

useMemo para datos procesados:

```
const chartData = useMemo(() =>
  getChartDataForQuestion(question),
  [question, responses]
);
```

useCallback para funciones:

```
const handleSubmit = useCallback(async () => {
  // lógica
}, [dependencies]);
```

9.2 Sincronización Silenciosa

```
const loadDataSilently = async () => {
  try {
    const data = await surveyDataSource.getSurveyResponses(surveyId);
    setResponses(data);
  } catch (error) {
    // No mostrar error al usuario
    console.log('Error en actualización automática:', error);
  }
};

useEffect(() => {
  const interval = setInterval(loadDataSilently, 5000);
  return () => clearInterval(interval);
}, []);
```

9.3 Lazy Loading

Componentes se cargan solo cuando son necesarios:

```
const SurveyEditor = lazy(() => import('./screens/survey-editor/SurveyEditorScreen'));
```

9.4 Caché de Imágenes

React Native cachea automáticamente las imágenes:

```
<Image
  source={{ uri: imageUrl }}
  style={styles.image}
  resizeMode="contain"
/>
```

---

10. Manejo de Errores

10.1 Estrategia de Manejo de Errores

Capas de manejo:
1. **Try-Catch local**: En funciones asíncronas
2. **Interceptores de Axios**: Errores HTTP globales
3. **Error Boundaries**: Errores de renderizado (React)
4. **Toast notifications**: Feedback al usuario

10.2 Tipos de Errores

Errores de Red

```
try {
  await httpClient.get('/surveys');
} catch (error: any) {
  if (error.code === 'ECONNABORTED') {
    showToast('Tiempo de espera agotado', 'error');
  } else if (!error.response) {
    showToast('Error de conexión', 'error');
  }
}
```

Errores de Validación

```
if (!validate()) {
  showToast('Por favor completa todos los campos', 'error');
  return;
}
```

Errores de Autenticación

```
if (error.response?.status === 401) {
  showToast('Sesión expirada', 'error');
  await logout();
}
```

10.3 Logging

Desarrollo:

```
console.log('[DEBUG] Survey loaded:', survey);
console.error('[ERROR] Failed to load:', error);
```

Producción:
- Integración con servicio de logging (ej: Sentry)
- No exponer información sensible

---

11. Guías de Desarrollo

11.1 Crear Nueva Pantalla

1. **Crear carpeta en screens/**

```
mkdir src/presentation/screens/nueva-funcionalidad
```

1. **Crear archivos de componente y estilos**

```
touch src/presentation/screens/nueva-funcionalidad/NuevaScreen.tsx
touch src/presentation/screens/nueva-funcionalidad/NuevaScreen.styles.ts
```

1. **Implementar componente**

```
import React from 'react';
import { View, Text } from 'react-native';
import { styles } from './NuevaScreen.styles';

const NuevaScreen: React.FC = () => {
  return (
    <View style={styles.container}>
      <Text>Nueva Pantalla</Text>
    </View>
  );
```

```
};

export default NuevaScreen;
```

1. **Crear estilos**

```
import { StyleSheet } from 'react-native';

export const styles = StyleSheet.create({
  container: {
    flex: 1,
    backgroundColor: '#F9FAFB',
  },
});
```

1. **Agregar a navegación**

```
// En MainNavigator.tsx
import NuevaScreen from '../screens/nueva-funcionalidad/NuevaScreen';

// Agregar tipo en types.ts
export type MainStackParamList = {
  Nueva: { param?: string };
};

// Agregar screen
<Stack.Screen name="Nueva" component={NuevaScreen} />
```

11.2 Agregar Nuevo Store

1. **Crear archivo de store**

```
// src/presentation/stores/nuevoStore.ts
import { create } from 'zustand';

interface NuevoStore {
  data: any[];
  setData: (data: any[]) => void;
}
```

```
export const useNuevoStore = create<NuevoStore>((set) => ({
  data: [],
  setData: (data) => set({ data }),
}));
```

1. **Usar en componentes**

```
const data = useNuevoStore((state) => state.data);
const setData = useNuevoStore((state) => state.setData);
```

11.3 Crear Componente Reutilizable

```
// src/presentation/components/MiComponente.tsx
import React from 'react';
import { View, Text, StyleSheet } from 'react-native';

interface MiComponenteProps {
  titulo: string;
  onPress?: () => void;
}

export const MiComponente: React.FC<MiComponenteProps> = ({
  titulo,
  onPress
}) => {
  return (
    <View style={styles.container}>
      <Text>{titulo}</Text>
    </View>
  );
};

const styles = StyleSheet.create({
  container: {
    padding: 16,
  },
});
```

11.4 Convenciones de Código

Nombres de archivos:
- Componentes: `PascalCase.tsx`
- Estilos: `PascalCase.styles.ts`
- Utilidades: `camelCase.ts`
- Constantes: `UPPER_SNAKE_CASE.ts`

Imports organizados:

```
// 1. React y librerías externas
import React, { useState } from 'react';
import { View } from 'react-native';

// 2. Componentes locales
import { CustomButton } from '../components/CustomButton';

// 3. Hooks
import { useAuth } from '../hooks/useAuth';

// 4. Stores
import { useAuthStore } from '../stores/authStore';

// 5. Tipos
import { User } from '../../core/domain/entities/User';

// 6. Estilos
import { styles } from './Screen.styles';
```

---

12. Testing

12.1 Estructura de Tests

```
__tests__/
├── components/
│   ├── CustomButton.test.tsx
│   └── CustomInput.test.tsx
```

```
├── screens/
│   └── LoginScreen.test.tsx
├── stores/
│   ├── authStore.test.ts
│   └── surveyStore.test.ts
└── utils/
    └── validations.test.ts
```

12.2 Testing de Componentes

```
import React from 'react';
import { render, fireEvent } from '@testing-library/react-native';
import { CustomButton } from '../CustomButton';

describe('CustomButton', () => {
  it('renders correctly', () => {
    const { getByText } = render(
      <CustomButton title="Click me" onPress={() => {}} />
    );
    expect(getByText('Click me')).toBeTruthy();
  });

  it('calls onPress when pressed', () => {
    const mockOnPress = jest.fn();
    const { getByText } = render(
      <CustomButton title="Click me" onPress={mockOnPress} />
    );

    fireEvent.press(getByText('Click me'));
    expect(mockOnPress).toHaveBeenCalled();
  });
});
```

12.3 Testing de Stores

```
import { renderHook, act } from '@testing-library/react-hooks';
import { useAuthStore } from '../authStore';

describe('authStore', () => {
```

```
  it('should set user on login', async () => {
    const { result } = renderHook(() => useAuthStore());

    await act(async () => {
      await result.current.login('test@test.com', 'password');
    });

    expect(result.current.user).toBeDefined();
    expect(result.current.token).toBeTruthy();
  });
});
```

12.4 Testing de Utilidades

```
import { validateEmail } from '../validations';

describe('validateEmail', () => {
  it('returns true for valid email', () => {
    expect(validateEmail('test@test.com')).toBe(true);
  });

  it('returns false for invalid email', () => {
    expect(validateEmail('invalid-email')).toBe(false);
  });
});
```

---

13. Despliegue

13.1 Build para Producción

Android

```
cd android
./gradlew assembleRelease
```

APK ubicado en:
`android/app/build/outputs/apk/release/app-release.apk`

iOS

```
cd ios
pod install
```

Abrir Xcode y crear archive para App Store.

13.2 Variables de Entorno

Desarrollo:

```
API_BASE_URL=http://localhost:3000/api
```

Producción:

```
API_BASE_URL=https://api.produccion.com/api
```

13.3 Versionado

Seguir Semantic Versioning:
- MAJOR: Cambios incompatibles
- MINOR: Nueva funcionalidad compatible
- PATCH: Correcciones de bugs

---

14. Decisiones Técnicas

14.1 ¿Por qué Zustand en lugar de Redux?

- **Menos boilerplate**: No requiere actions, reducers, etc.
- **TypeScript nativo**: Mejor inferencia de tipos
- **Performance**: Re-renders más eficientes
- **Simplicidad**: API más intuitiva
- **Tamaño**: Bundle más pequeño

14.2 ¿Por qué Clean Architecture?

- **Separación de responsabilidades**: Cada capa tiene un propósito
- **Testabilidad**: Fácil de testear cada capa independientemente
- **Mantenibilidad**: Código organizado y predecible
- **Escalabilidad**: Fácil agregar nuevas features
- **Independencia del framework**: Lógica de negocio independiente

14.3 ¿Por qué React Native Chart Kit?

- **Nativo**: Renderizado nativo, mejor performance
- **Variedad**: Múltiples tipos de gráficos
- **Customizable**: Fácil de personalizar
- **TypeScript**: Soporte completo
- **Activamente mantenido**: Comunidad activa

---

15. Troubleshooting

15.1 Problemas Comunes

Error: Unable to resolve module

```
# Limpiar caché
npm start -- --reset-cache
```

Error: Android build failed

```
cd android
./gradlew clean
cd ..
npm run android
```

Error: iOS pod install failed

```
cd ios
pod deintegrate
pod install
cd ..
```

Error: Network request failed
- Verificar que el backend esté corriendo
- Verificar URL en `http.ts`
- En Android emulator, usar `http://10.0.2.2:3000` en lugar de `localhost`

15.2 Logs y Debugging

Ver logs de Android:

```
adb logcat *:S ReactNative:V ReactNativeJS:V
```

Ver logs de iOS:

```
npx react-native log-ios
```

---

16. Recursos Adicionales

Documentación Oficial

- React Native
- React Navigation
- Zustand
- Axios

Herramientas Recomendadas

- **VS Code**: Editor de código
- **React Native Debugger**: Debugging
- **Flipper**: Debugging avanzado

- **Postman**: Testing de API

Última actualización: 2025-01-09

Mantenedores: Equipo de desarrollo

Versión del documento: 1.0.0