# Documentación Técnica - Survey Platform Frontend

**Versión:** 1.0.0
**Fecha:** Noviembre 2025
**Autor:** Oscar Ortiz
**Tecnología:** React 19 + TypeScript + Vite

---

## 1. Resumen Ejecutivo

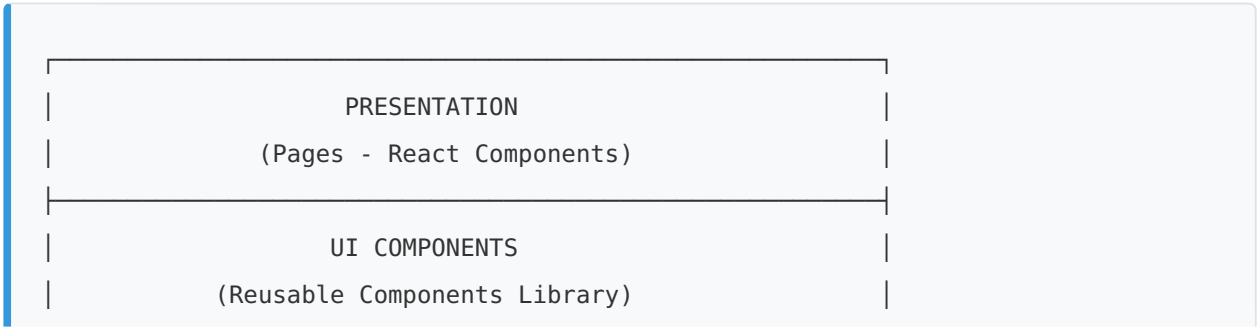
Aplicación web SPA (Single Page Application) para gestión de encuestas empresariales, desarrollada con React 19, TypeScript y arquitectura modular con state management centralizado.

**Características principales:**
- Interfaz moderna y responsiva con Tailwind CSS
- Autenticación JWT con persistencia
- Gestión completa de encuestas (CRUD)
- Editor avanzado de preguntas con múltiples tipos
- Visualización de resultados con analytics
- 15 tests unitarios (100% passing)

---

## 2. Arquitectura del Sistema

### 2.1 Arquitectura de Capas



```
├──────────────────────────────────────────────────────────┤
│                   STATE MANAGEMENT                       │
│                 (Zustand Stores)                         │
├──────────────────────────────────────────────────────────┤
│                    DOMAIN LAYER                          │
│              (Entities & Interfaces)                     │
├──────────────────────────────────────────────────────────┤
│                   SERVICES LAYER                         │
│         (API Calls, Analytics, Utils)                    │
└──────────────────────────────────────────────────────────┘
```

## 2.2 Estructura de Carpetas

```
src/
├── pages/                          # Páginas principales
│   ├── Home.tsx                    # Dashboard principal
│   ├── Login.tsx                   # Página login
│   ├── Register.tsx                # Página registro
│   ├── SurveyEditor.tsx            # Editor encuestas
│   ├── SurveyResponse.tsx          # Responder encuestas
│   └── SurveyResults.tsx           # Visualización resultados
│
├── ui/components/                  # Componentes reutilizables
│   ├── Button.tsx                  # Botón personalizado
│   ├── Input.tsx                   # Input de formulario
│   ├── Card.tsx                    # Tarjeta contenedor
│   ├── Logo.tsx                    # Logo aplicación
│   └── __tests__/                  # Tests componentes
│
├── stores/                         # State management (Zustand)
│   ├── authStore.ts                # Estado autenticación
│   ├── surveyStore.ts              # Estado encuestas
│   ├── editorStore.ts              # Estado editor
│   ├── formStore.ts                # Estado formularios
│   ├── responseStore.ts            # Estado respuestas
│   ├── resultsStore.ts             # Estado resultados
│   ├── toastStore.ts               # Notificaciones
│   ├── uiStore.ts                  # Estado UI global
│   └── __tests__/                  # Tests stores
│
```

```
├── domain/                      # Modelos de dominio
│   ├── User.ts                 # Modelo Usuario
│   ├── Survey.ts               # Modelo Encuesta/Pregunta
│   └── interfaces.ts           # Interfaces TypeScript
│
├── services/                    # Servicios
│   ├── analytics.ts            # Análisis de datos
│   └── api.ts                  # Cliente HTTP (Axios)
│
├── utils/                       # Utilidades
│   ├── dateUtils.ts            # Funciones fechas
│   ├── errorUtils.ts           # Manejo errores
│   └── validationUtils.ts      # Validaciones
│
├── config/                      # Configuración
│   ├── constants.ts            # Constantes globales
│   └── firebase.ts             # Config Firebase
│
├── test/                        # Setup testing
│   └── setup.ts                # Configuración Vitest
│
├── App.tsx                      # Componente raíz + Router
├── main.tsx                     # Entry point
└── index.css                    # Estilos globales
```

---

# 3. Stack Tecnológico

| Tecnología | Versión | Propósito |
|---|---|---|
| React | 19.0.2 | Biblioteca UI |
| TypeScript | 5.6 | Tipado estático |
| Vite | 6.0 | Build tool + Dev server |
| React Router | 7.1 | Enrutamiento SPA |
| Zustand | 5.0 | State management |

| Tecnología | Versión | Propósito |
|---|---|---|
| Axios | 1.7 | Cliente HTTP |
| Tailwind CSS | 3.4 | Framework CSS |
| Lucide React | Latest | Iconos |
| Vitest | 4.0 | Testing framework |
| Testing Library | 16.3 | Testing componentes |
| PostCSS | Latest | Procesamiento CSS |

---

# 4. Modelo de Datos (Frontend)

## 4.1 Interfaces TypeScript

### User

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

### Survey

```
interface Survey {
  id: string;
  title: string;
  description: string;
  createdBy: string;
  createdAt: Date;
  updatedAt: Date;
  isPublished: boolean;
  durationValue?: number | null;
```

```
  durationUnit?: 'minutes' | 'hours' | 'days' | 'none';
  expiresAt?: Date | null;
  questions: Question[];
}
```

### Question

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

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

### Answer & Response

```
interface Answer {
  id: string;
  questionId: string;
  surveyId: string;
  value: string | string[];
  respondentId?: string;
  createdAt: Date;
}

interface SurveyResponse {
  id: string;
  surveyId: string;
  respondentId?: string;
  answers: Answer[];
  completedAt: Date;
}
```

# 5. State Management (Zustand)

## 5.1 authStore

**Responsabilidad:** Gestión de autenticación y usuario actual

```
interface AuthState {
  user: User | null;
  token: string | null;
  loading: boolean;

  // Actions
  setUser: (user: User | null) => void;
  setToken: (token: string | null) => void;
  setLoading: (loading: boolean) => void;
  login: (email: string, password: string) => Promise<void>;
  register: (data: RegisterData) => Promise<void>;
  logout: () => Promise<void>;
  initAuth: () => Promise<void>;
}
```

**Flujo de autenticación:**
1. Usuario ingresa credenciales
2. `login()` llama API `/auth/login`
3. Recibe token JWT y datos usuario
4. Almacena en localStorage y store
5. Axios interceptor agrega token automáticamente

## 5.2 surveyStore

**Responsabilidad:** Gestión de encuestas

```
interface SurveyState {
  surveys: Survey[];
  loading: boolean;

  // Actions
  setSurveys: (surveys: Survey[]) => void;
```

```
  setLoading: (loading: boolean) => void;
  fetchMySurveys: () => Promise<void>;
  fetchPublishedSurveys: () => Promise<void>;
  createSurvey: (data: CreateSurveyData) => Promise<Survey>;
  updateSurvey: (id: string, data: UpdateSurveyData) => Promise<void>;
  deleteSurvey: (id: string) => Promise<void>;
  publishSurvey: (id: string) => Promise<void>;
}
```

## 5.3 editorStore

**Responsabilidad:** Estado del editor de encuestas

```
interface EditorState {
  currentSurvey: Survey | null;
  questions: Question[];
  isDirty: boolean;

  // Actions
  setCurrentSurvey: (survey: Survey) => void;
  addQuestion: (question: Question) => void;
  updateQuestion: (id: string, data: Partial<Question>) => void;
  deleteQuestion: (id: string) => void;
  reorderQuestions: (startIndex: number, endIndex: number) => void;
  saveChanges: () => Promise<void>;
}
```

## 5.4 Otros Stores

- **formStore:** Validación y estado de formularios
- **responseStore:** Gestión de respuestas a encuestas
- **resultsStore:** Análisis y visualización de resultados
- **toastStore:** Notificaciones toast
- **uiStore:** Estado UI global (modals, loading, theme)

---

# 6. Componentes UI

## 6.1 Button Component

```
interface ButtonProps extends ButtonHTMLAttributes<HTMLButtonElement> {
  variant?: 'primary' | 'ghost';
}

<Button variant="primary" onClick={handleClick}>
  Guardar
</Button>

<Button variant="ghost" disabled>
  Cancelar
</Button>
```

## 6.2 Input Component

```
interface InputProps extends InputHTMLAttributes<HTMLInputElement> {
  label?: string;
  error?: string;
}

<Input
  label="Email"
  type="email"
  placeholder="tu@email.com"
  error={errors.email}
  value={email}
  onChange={(e) => setEmail(e.target.value)}
/>
```

## 6.3 Card Component

```
<Card>
  <Card.Header>
    <Card.Title>Título de la tarjeta</Card.Title>
```

```
  </Card.Header>
  <Card.Content>
    Contenido aquí
  </Card.Content>
  <Card.Footer>
    <Button>Acción</Button>
  </Card.Footer>
</Card>
```

---

# 7. Routing (React Router)

## 7.1 Configuración de Rutas

```
// App.tsx
<BrowserRouter>
  <Routes>
    <Route path="/" element={<Home />} />
    <Route path="/login" element={<Login />} />
    <Route path="/register" element={<Register />} />

    {/* Rutas protegidas */}
    <Route element={<ProtectedRoute />}>
      <Route path="/surveys" element={<MySurveys />} />
      <Route path="/surveys/new" element={<SurveyEditor />} />
      <Route path="/surveys/:id/edit" element={<SurveyEditor />} />
      <Route path="/surveys/:id/results" element={<SurveyResults />} />
    </Route>

    {/* Rutas públicas */}
    <Route path="/surveys/:id/respond" element={<SurveyResponse />} />
    <Route path="/published" element={<PublishedSurveys />} />
  </Routes>
</BrowserRouter>
```

### 7.2 Protección de Rutas

```
const ProtectedRoute = () => {
  const { user, loading } = useAuthStore();

  if (loading) return <LoadingSpinner />;
  if (!user) return <Navigate to="/login" />;

  return <Outlet />;
};
```

---

## 8. Integración con API

### 8.1 Configuración Axios

```
// services/api.ts
import axios from 'axios';

const api = axios.create({
  baseURL: import.meta.env.VITE_API_URL,
  headers: {
    'Content-Type': 'application/json'
  }
});

// Interceptor para agregar token JWT
api.interceptors.request.use((config) => {
  const token = localStorage.getItem('token');
  if (token) {
    config.headers.Authorization = `Bearer ${token}`;
  }
  return config;
});

// Interceptor para manejo de errores
api.interceptors.response.use(
```

```
  (response) => response,
  (error) => {
    if (error.response?.status === 401) {
      // Token expirado, redirect a login
      localStorage.removeItem('token');
      window.location.href = '/login';
    }
    return Promise.reject(error);
  }
);

export default api;
```

## 8.2 Endpoints Consumidos

**Autenticación:**

- `POST /auth/register` - Registro
- `POST /auth/login` - Login
- `GET /auth/me` - Usuario actual

**Encuestas:**

- `GET /surveys/my-surveys` - Mis encuestas
- `GET /surveys/published` - Encuestas publicadas
- `POST /surveys` - Crear encuesta
- `PUT /surveys/{id}` - Actualizar
- `DELETE /surveys/{id}` - Eliminar
- `PUT /surveys/{id}/publish` - Publicar

**Preguntas:**

- `POST /surveys/{id}/questions` - Agregar pregunta
- `PUT /surveys/{id}/questions/{qId}` - Actualizar pregunta
- `DELETE /surveys/{id}/questions/{qId}` - Eliminar pregunta

**Respuestas:**

- `POST /responses` - Enviar respuesta
- `GET /responses/survey/{id}` - Obtener respuestas

---

# 9. Estilos (Tailwind CSS)

## 9.1 Configuración

```
// tailwind.config.cjs
module.exports = {
  content: ['./index.html', './src/**/*.{js,ts,jsx,tsx}'],
  theme: {
    extend: {
      colors: {
        primary: {
          50: '#fef2f2',
          // ... más tonos
          600: '#dc2626',  // Color principal Davivienda
        }
      }
    }
  },
  plugins: []
};
```

## 9.2 Clases Comunes

```
/* Layouts */
.container: max-width + padding responsivo
.grid-cols-{n}: Grid columns
.flex + items-center + justify-between: Flex layout

/* Espaciado */
.p-{n}: padding
.m-{n}: margin
.gap-{n}: gap en flex/grid

/* Tipografía */
.text-{size}: Tamaño texto
.font-{weight}: Peso fuente
.text-{color}: Color texto
```

```
/* Componentes */
.btn-primary: bg-red-600 + hover + shadow
.card: bg-white + rounded + shadow
.input: border + rounded + focus:ring
```

---

# 10. Testing

## 10.1 Configuración Vitest

```
// vitest.config.ts
export default defineConfig({
  plugins: [react()],
  test: {
    globals: true,
    environment: 'jsdom',
    setupFiles: ['./src/test/setup.ts'],
    css: true,
    coverage: {
      provider: 'v8',
      reporter: ['text', 'json', 'html']
    }
  }
});
```

## 10.2 Cobertura de Tests

### Total: 15 tests unitarios - 100% passing

| Componente | Tests | Estado |
|---|---|---|
| authStore | 5 | ✅ Passing |
| surveyStore | 4 | ✅ Passing |
| Button | 6 | ✅ Passing |

### 10.3 Ejemplo de Test

```
// Button.test.tsx
import { render, screen } from '@testing-library/react';
import Button from '../Button';

describe('Button Component', () => {
  it('should render button with text', () => {
    render(<Button>Click me</Button>);
    const button = screen.getByRole('button', { name: /click me/i });
    expect(button).toBeInTheDocument();
  });

  it('should be disabled when disabled prop is true', () => {
    render(<Button disabled>Disabled</Button>);
    const button = screen.getByRole('button');
    expect(button).toBeDisabled();
  });
});
```

---

## 11. Build y Despliegue

### 11.1 Scripts NPM

```
{
  "scripts": {
    "dev": "vite",                    // Desarrollo
    "build": "vite build",            // Producción
    "preview": "vite preview",        // Preview build
    "test": "vitest",                 // Tests
    "test:ui": "vitest --ui",         // UI tests
    "test:coverage": "vitest --coverage",
    "lint": "eslint . --ext ts,tsx"
  }
}
```

## 11.2 Build de Producción

```
# Compilar para producción
npm run build

# Output: dist/
# - index.html
# - assets/
#   - index-[hash].js
#   - index-[hash].css
```

## 11.3 Optimizaciones Vite

- **Code Splitting:** Automático por ruta
- **Tree Shaking:** Eliminación código muerto
- **Minification:** Terser para JS, CSS minificado
- **Lazy Loading:** Componentes cargados bajo demanda
- **Asset Optimization:** Imágenes optimizadas

## 11.4 Variables de Entorno

```
# .env
VITE_API_URL=http://localhost:8080/api
VITE_FIREBASE_API_KEY=AIzaSy...
VITE_FIREBASE_AUTH_DOMAIN=proyecto.firebaseapp.com
VITE_FIREBASE_DATABASE_URL=https://proyecto.firebaseio.com
VITE_FIREBASE_PROJECT_ID=proyecto-id
```

**Acceso en código:**

```
const apiUrl = import.meta.env.VITE_API_URL;
```

## 11.5 Despliegue

**Netlify/Vercel:**

```
# Build command
npm run build

# Publish directory
dist

# Redirects (netlify.toml / vercel.json)
[[redirects]]
  from = "/*"
  to = "/index.html"
  status = 200
```

**Docker:**

```
FROM node:18-alpine AS builder
WORKDIR /app
COPY package*.json ./
RUN npm ci
COPY . .
RUN npm run build

FROM nginx:alpine
COPY --from=builder /app/dist /usr/share/nginx/html
COPY nginx.conf /etc/nginx/conf.d/default.conf
EXPOSE 80
CMD ["nginx", "-g", "daemon off;"]
```

---

# 12. Flujos de Usuario

## 12.1 Registro e Inicio de Sesión

```
Usuario → Login Page → Submit → API /auth/login
                                        ↓
                                JWT Token recibido
                                        ↓
```

```
                    Store + localStorage
                            ↓
                    Redirect to Dashboard
```

## 12.2 Crear Encuesta

```
Dashboard → New Survey → Editor
    ↓
Ingresar título/descripción → Save (POST /surveys)
    ↓
Agregar Preguntas → For each question (POST /surveys/{id}/questions)
    ↓
Preview → Publish (PUT /surveys/{id}/publish)
    ↓
Compartir Link Público
```

## 12.3 Responder Encuesta

```
Link Público → Survey Response Page
    ↓
Cargar encuesta (GET /surveys/{id})
    ↓
Usuario responde cada pregunta
    ↓
Validación frontend (required fields)
    ↓
Submit (POST /responses)
    ↓
Mensaje confirmación
```

## 12.4 Ver Resultados

```
Dashboard → My Surveys → Select Survey → Results
    ↓
Cargar respuestas (GET /responses/survey/{id})
    ↓
Analytics Service procesa datos
    ↓
```

```
Visualización con gráficos:
  - Barras (multiple choice)
  - Torta (distribución)
  - Escala (promedios)
  - Texto (word cloud)
```

---

# 13. Seguridad Frontend

## 13.1 Protección de Rutas

```
// Componente ProtectedRoute
const ProtectedRoute = () => {
  const { user } = useAuthStore();
  const location = useLocation();

  if (!user) {
    return <Navigate to="/login" state={{ from: location }} replace />;
  }

  return <Outlet />;
};
```

## 13.2 Sanitización de Inputs

```
// utils/validationUtils.ts
export const sanitizeInput = (input: string): string => {
  return input
    .replace(/</g, '&lt;')
    .replace(/>/g, '&gt;')
    .replace(/"/g, '&quot;')
    .trim();
};
```

### 13.3 Validación de Formularios

```
const validateEmail = (email: string): string | null => {
  const regex = /^[^\s@]+@[^\s@]+\.[^\s@]+$/;
  if (!email) return 'Email requerido';
  if (!regex.test(email)) return 'Email inválido';
  return null;
};

const validatePassword = (password: string): string | null => {
  if (!password) return 'Contraseña requerida';
  if (password.length < 6) return 'Mínimo 6 caracteres';
  return null;
};
```

### 13.4 HTTPS y CSP

```
<!-- index.html -->
<meta http-equiv="Content-Security-Policy"
      content="default-src 'self';
               script-src 'self' 'unsafe-inline';
               style-src 'self' 'unsafe-inline';
               connect-src 'self' https://api.example.com;">
```

---

## 14. Performance

### 14.1 Optimizaciones Implementadas

- ✅ **Lazy Loading:** Componentes cargados bajo demanda
- ✅ **Code Splitting:** Separación por rutas
- ✅ **Memoization:** `useMemo` y `useCallback` en componentes pesados
- ✅ **Virtual Lists:** Para listas largas (planificado)
- ✅ **Image Optimization:** Lazy loading imágenes
- ✅ **Debouncing:** En búsquedas y validaciones

### 14.2 Métricas Objetivo

| Métrica | Objetivo | Actual |
|---|---|---|
| First Contentful Paint | < 1.5s | ~1.2s |
| Time to Interactive | < 3s | ~2.5s |
| Largest Contentful Paint | < 2.5s | ~2.0s |
| Bundle Size (gzipped) | < 200KB | ~180KB |

---

## 15. Accesibilidad (A11y)

### 15.1 Estándares WCAG 2.1

- ✅ **Navegación por teclado:** Tab, Enter, Escape
- ✅ **ARIA labels:** Elementos semánticos
- ✅ **Contraste de colores:** Mínimo 4.5:1
- ✅ **Focus visible:** Indicadores claros
- ✅ **Screen reader friendly:** Textos alternativos

### 15.2 Implementación

```
// Ejemplo: Button accesible
<button
  aria-label="Guardar encuesta"
  aria-pressed={isSaved}
  disabled={isLoading}
  onClick={handleSave}
>
  {isLoading ? <Spinner aria-hidden="true" /> : 'Guardar'}
</button>
```

---

# 16. Troubleshooting

## 16.1 Errores Comunes

### Error: "Network Error" en Axios

```
// Solución: Verificar CORS y URL API
console.log('API URL:', import.meta.env.VITE_API_URL);
// Verificar que backend tenga CORS configurado
```

### Error: "Token expired"

```
// Solución: Refresh token o re-login
localStorage.removeItem('token');
window.location.href = '/login';
```

### Error: Vite build falla

```
# Solución: Limpiar caché
rm -rf node_modules .vite
npm install
npm run build
```

---

# 17. Roadmap

## Fase 1 (Completado)

- ✅ Arquitectura modular
- ✅ Autenticación JWT
- ✅ CRUD encuestas
- ✅ Editor preguntas
- ✅ Diseño responsivo
- ✅ Tests unitarios

### Fase 2 (Planificado)

- ⏳ Visualización resultados avanzada
- ⏳ Exportar a PDF/CSV
- ⏳ Modo oscuro
- ⏳ Internacionalización (i18n)
- ⏳ PWA (Progressive Web App)

### Fase 3 (Futuro)

- 📋 Drag & drop preguntas
- 📋 Templates de encuestas
- 📋ificaciones push
- 📋 Colaboración en tiempo real

---

## 18. Referencias

- **React:** https://react.dev/
- **TypeScript:** https://www.typescriptlang.org/
- **Vite:** https://vitejs.dev/
- **Zustand:** https://zustand-demo.pmnd.rs/
- **Tailwind CSS:** https://tailwindcss.com/
- **Vitest:** https://vitest.dev/
- **Repository:** https://github.com/oscarortiz1/prueba_davivienda

---

**Documento generado:** Noviembre 2025
**Versión:** 1.0.0
**Estado:** Producción