# **Plan de Casos de Prueba - Social Service Platform**

## **1. Información General del Sistema**

### **Sistema Bajo Prueba**

* **Nombre**: Social Service Platform
* **Tecnología**: Next.js con Supabase
* **Tipos de Usuario**: Estudiantes, Socios Formadores, Administradores

### **Alcance de las Pruebas**

* Autenticación y autorización por roles
* Gestión de proyectos solidarios
* Sistema de postulaciones
* Carga masiva de datos
* Interfaces específicas por rol

## **2. Estrategia de Pruebas**

### **2.1 Clasificación de Casos de Prueba**

#### **Casos Automatizados (60%)**

* Pruebas de API y endpoints
* Validaciones de datos
* Autenticación y autorización
* Procesamiento de archivos
* Integración con Supabase

#### **Casos Manuales (40%)**

* Pruebas de interfaz de usuario
* Flujos de navegación
* Experiencia de usuario
* Pruebas exploratorias

## **3. Casos de Prueba Detallados**

### **3.1 MÓDULO DE AUTENTICACIÓN**

#### **CP-AUTH-001: Login de Estudiante con Google OAuth**

* **Tipo**: Manual
* **Prioridad**: Alta
* **Precondiciones**: Usuario debe tener cuenta con dominio @tec.mx
* **Pasos**:
  1. Acceder a la página de login
  2. Seleccionar "Iniciar sesión con Google"
  3. Ingresar credenciales @tec.mx
  4. Verificar redirección a dashboard de alumno
* **Resultado Esperado**: Usuario autenticado correctamente y matrícula extraída automáticamente
* **Datos de Prueba**: correo@tec.mx con matrícula válida

#### **CP-AUTH-002: Validación de Dominio de Correo**

* **Tipo**: Automatizado
* **Prioridad**: Alta
* **Descripción**: Verificar que solo correos @tec.mx pueden acceder como estudiantes
* **Casos**:
  + Correo válido: A01234567@tec.mx
  + Correo inválido: usuario@gmail.com
  + Formato incorrecto: correo@tecmilenio.mx

#### **CP-AUTH-003: Extracción Automática de Matrícula**

* **Tipo**: Automatizado
* **Prioridad**: Media
* **Descripción**: Validar extracción correcta de matrícula del correo
* **Casos**:
  + A01234567@tec.mx → Matrícula: A01234567

#### **CP-AUTH-004: Autenticación de Socio Formador**

* **Tipo**: Manual
* **Prioridad**: Alta
* **Precondiciones**: Cuenta de socio previamente creada por admin
* **Pasos**:
  1. Acceder a ruta /socio
  2. Iniciar sesión con credenciales de socio
  3. Verificar acceso a proyectos asignados
* **Resultado Esperado**: Acceso solo a proyectos correspondientes al socio

### **3.2 MÓDULO DE GESTIÓN DE PROYECTOS**

#### **CP-PROJ-001: Visualización de Proyectos para Estudiantes**

* **Tipo**: Manual
* **Prioridad**: Alta
* **Precondiciones**: Usuario estudiante autenticado
* **Pasos**:
  1. Navegar a /alumno/explorar
  2. Verificar listado de proyectos disponibles
  3. Comprobar información mostrada en CardItem
* **Resultado Esperado**: Lista de proyectos con datos completos y navegación funcional

#### **CP-PROJ-002: Filtrado y Búsqueda de Proyectos**

* **Tipo**: Manual
* **Prioridad**: Media
* **Pasos**:
  1. Aplicar filtros disponibles
  2. Realizar búsqueda por palabras clave
  3. Verificar resultados coherentes
* **Resultado Esperado**: Filtros funcionan correctamente

#### **CP-PROJ-003: Detalle de Proyecto**

* **Tipo**: Manual
* **Prioridad**: Alta
* **Pasos**:
  1. Seleccionar un proyecto del listado
  2. Verificar información detallada
  3. Comprobar disponibilidad de formulario de postulación
* **Resultado Esperado**: Información completa y botón de postulación activo

#### **CP-PROJ-004: Carga Masiva de Proyectos (CSV)**

* **Tipo**: Automatizado
* **Prioridad**: Alta
* **Precondiciones**: Usuario admin autenticado
* **Datos de Prueba**:

nombre,descripcion,socio\_formador,ubicacion,cupos

Proyecto A,Descripción del proyecto A,socio1@empresa.com,Monterrey,5

Proyecto B,Descripción del proyecto B,socio2@empresa.com,CDMX,3

* **Validaciones**:
  + Campos requeridos presentes
  + Formato de datos correcto
  + Creación exitosa en BD

#### **CP-PROJ-005: Carga Masiva de Proyectos (XLSX)**

* **Tipo**: Automatizado
* **Prioridad**: Alta
* **Descripción**: Validar procesamiento de archivos Excel
* **Validaciones**:
  + Lectura correcta de columnas
  + Manejo de celdas vacías
  + Conversión de tipos de datos

#### **CP-PROJ-006: Validación de Campos Requeridos en Carga**

* **Tipo**: Automatizado
* **Prioridad**: Alta
* **Casos**:
  + Archivo sin campo 'nombre'
  + Archivo sin campo 'socio\_formador'
  + Archivo con todos los campos requeridos

### **3.3 MÓDULO DE POSTULACIONES**

#### **CP-POST-001: Crear Nueva Postulación**

* **Tipo**: Manual
* **Prioridad**: Alta
* **Precondiciones**: Estudiante autenticado, proyecto disponible
* **Pasos**:
  1. Acceder a formulario de postulación /alumno/explorar/proyecto/formulario/[id]
  2. Completar datos requeridos
  3. Enviar postulación
* **Resultado Esperado**: Postulación creada con status 'pendiente'

#### **CP-POST-002: Validación de Formulario de Postulación**

* **Tipo**: Automatizado
* **Prioridad**: Media
* **Validaciones**:
  + Campos obligatorios completos
  + Formato de datos correcto
  + Longitud de texto adecuada

#### **CP-POST-003: Gestión de Postulaciones por Socio**

* **Tipo**: Manual
* **Prioridad**: Alta
* **Precondiciones**: Socio autenticado con postulaciones pendientes
* **Pasos**:
  1. Acceder a dashboard de socio
  2. Visualizar postulaciones en ListaSocio
  3. Cambiar status de postulación
* **Resultado Esperado**: Status actualizado correctamente

#### **CP-POST-004: Estados de Postulación**

* **Tipo**: Automatizado
* **Prioridad**: Media
* **Estados Válidos**:
  + pendiente → aprobada
  + pendiente → rechazada
  + aprobada → finalizada
  + rechazada → pendiente (no permitido)

#### **CP-POST-005: Notificaciones por Email**

* **Tipo**: Automatizado
* **Prioridad**: Baja
* **Descripción**: Verificar envío de emails via Resend
* **Casos**:
  + Notificación de postulación recibida
  + Notificación de cambio de estado

### **3.4 MÓDULO DE ADMINISTRACIÓN**

#### **CP-ADMIN-001: Creación Masiva de Usuarios**

* **Tipo**: Automatizado
* **Prioridad**: Alta
* **Endpoint**: /api/create-users
* **Precondiciones**: Lista de socios formadores en archivo
* **Validaciones**:
  + Usuarios creados en Supabase Auth
  + Relación correcta socio-proyecto
  + Credenciales válidas generadas

#### **CP-ADMIN-002: Dashboard de Administración**

* **Tipo**: Manual
* **Prioridad**: Media
* **Pasos**:
  1. Acceder como administrador
  2. Verificar métricas del sistema
  3. Acceder a herramientas de gestión
* **Resultado Esperado**: Dashboard funcional con datos actualizados

#### **CP-ADMIN-003: Gestión de UploaderButton**

* **Tipo**: Manual
* **Prioridad**: Alta
* **Pasos**:
  1. Seleccionar archivo CSV/XLSX
  2. Procesar carga
  3. Verificar feedback visual
* **Resultado Esperado**: Proceso de carga con indicadores de progreso

### **3.5 MÓDULO DE INTEGRACIÓN**

#### **CP-INT-001: Conectividad con Supabase**

* **Tipo**: Automatizado
* **Prioridad**: Crítica
* **Verificaciones**:
  + Conexión a base de datos
  + Autenticación funcional
  + Operaciones CRUD

#### **CP-INT-002: Gestión de Sesiones**

* **Tipo**: Automatizado
* **Prioridad**: Alta
* **Validaciones**:
  + supabase.auth.getSession() funcional
  + Persistencia de sesión
  + Logout correcto

#### **CP-INT-003: Real-time Features**

* **Tipo**: Manual
* **Prioridad**: Baja
* **Descripción**: Verificar actualizaciones en tiempo real de Supabase

## **4. Métricas de Cobertura**

### **4.1 Cobertura por Módulos**

| **Módulo** | **Casos Totales** | **Automatizados** | **Manuales** | **Cobertura** |
| --- | --- | --- | --- | --- |
| Autenticación | 5 | 3 (60%) | 2 (40%) | 95% |
| Gestión Proyectos | 6 | 4 (67%) | 2 (33%) | 90% |
| Postulaciones | 5 | 2 (40%) | 3 (60%) | 85% |
| Administración | 3 | 1 (33%) | 2 (67%) | 80% |
| Integración | 3 | 2 (67%) | 1 (33%) | 90% |
| **TOTAL** | **22** | **12 (55%)** | **10 (45%)** | **88%** |

### **4.2 Cobertura por Prioridad**

| **Prioridad** | **Cantidad** | **Porcentaje** |
| --- | --- | --- |
| Crítica | 1 | 4.50% |
| Alta | 13 | 59.10% |
| Media | 6 | 27.30% |
| Baja | 2 | 9.10% |

### **4.3 Cobertura Funcional**

* **Flujos Principales**: 100%
* **Casos de Error**: 75%
* **Validaciones**: 90%
* **Integración**: 85%

## **5. Herramientas y Tecnologías de Prueba**

### **5.1 Pruebas Automatizadas**

* **Framework**: Jest + React Testing Library
* **API Testing**: Supertest
* **E2E**: Playwright o Cypress
* **Base de Datos**: Supabase Testing Environment

### **5.2 Pruebas Manuales**

* **Navegadores**: Chrome, Firefox, Safari
* **Dispositivos**: Desktop, Tablet, Mobile
* **Herramientas**: DevTools, Postman

## **6. Criterios de Aceptación**

### **6.1 Criterios de Entrada**

* Ambiente de testing configurado
* Base de datos de prueba poblada
* Credenciales de testing disponibles

### **6.2 Criterios de Salida**

* 100% de casos críticos y alta prioridad ejecutados
* 0 defectos críticos pendientes
* Cobertura mínima del 85% alcanzada
* Documentación de resultados completa

## **9. Datos de Prueba**

### **9.1 Usuarios de Prueba**

****Estudiante: A01234567@tec.mx

Socio: socio.test@empresa.com

Admin: admin.test@sistema.com

### **9.2 Proyectos de Prueba**

* Proyecto con cupos disponibles
* Proyecto sin cupos
* Proyecto con postulaciones pendientes

### **9.3 Archivos de Carga**

* CSV válido con 10 registros
* XLSX válido con 5 registros
* Archivos con errores para validación

Yes, I can generate a class diagram for your Social Service Platform system. Based on the codebase structure, here's a UML class diagram that represents the main components and their relationships:

```mermaid

classDiagram

%% Core Data Models

class Solicitud {

+string estatus

+string matricula

+string email

+string carrera

+string numero

+string respuesta\_1

+string respuesta\_2

+string respuesta\_3

+number id\_proyecto

+string proyecto

+boolean modificado

}

class Project {

+any id\_proyecto

+any proyecto

+any cupos

+any objetivo\_ps

+any horas

+any modalidad

}

class Postulacion {

+number id\_proyecto

+string email

}

%% React Components - Student Interface

class CardItem {

+string name

+string description

+string state

+number id\_project

+string format

+number hours

+string color

+render()

}

class StudentExplorarPage {

+Project[] projects

+Postulacion[] postulaciones

+string search

+string[] filterCarrera

+string[] filterModalidad

+fetchProjects()

+handleSearch()

+handleFilter()

}

%% React Components - Socio Interface

class SocioPage {

+Solicitud[] solicitudes

+Solicitud[] solicitudesOriginal

+string search

+string[] filterCarrera

+string[] filterEstado

+string socioCorreo

+number[] proyectosSocio

+string nombreProyecto

+boolean mensajeVisible

+fetchProyectosSocio()

+fetchProyectos()

+handleEnviar()

}

class ListaSocio {

+Solicitud[] data

+setData()

+render()

}

class Download {

+string format

+string[] columns

+handleDownload()

+onClose()

}

%% React Components - Admin Interface

class UploaderButton {

+File file

+boolean isImporting

+string successMessage

+string errorMessage

+string[] requiredFields

+handleFileUpload()

+validateFields()

+filterRequiredFields()

+handleImport()

}

class AdminProyectosPage {

+Explorar[] proyectos

+string search

+string[] filterEstatus

+fetchProyectos()

}

%% Shared Components

class HeaderBar {

+string titulo

+LucideIcon Icono

+string username

+onClick()

+getUser()

}

class SearchBar {

+string search

+setSearch()

+onSearchApply()

+onSearchClear()

}

class FilterButton {

+string label

+string[] options

+string[] selectedValues

+onChange()

}

class DetailButton {

+string texto

+string size

+string color

+number id

+onClick()

}

%% API Routes

class CreateUsersAPI {

+Project[] projects

+createUser()

+sendEmail()

+insertSocioformador()

+POST()

}

%% Authentication

class LoginSocioPage {

+string correo

+string clave

+string error

+boolean isLoading

+checkSession()

+handleLogin()

}

%% Database Tables (represented as classes)

class ProyectosSolidarios {

+number id\_proyecto

+string proyecto

+number cupos

+string perfil\_aceptacion

+string objetivo\_ps

+string modalidad

+string carreras

+string correo

}

class PostulacionDB {

+string matricula

+number id\_proyecto

+string estatus

+string nombre

+string carrera

+string email

+string numero

}

class SocioformadorDB {

+number id

+string correo

+number id\_proyecto

+string nombre\_organizacion

}

%% External Services

class SupabaseClient {

+auth

+from()

+select()

+insert()

+update()

+getSession()

}

class ResendService {

+sendEmail()

}

%% Relationships

StudentExplorarPage --> CardItem : renders

StudentExplorarPage --> Project : uses

StudentExplorarPage --> HeaderBar : uses

StudentExplorarPage --> SearchBar : uses

StudentExplorarPage --> FilterButton : uses

StudentExplorarPage --> SupabaseClient : queries

SocioPage --> Solicitud : manages

SocioPage --> ListaSocio : renders

SocioPage --> Download : opens modal

SocioPage --> HeaderBar : uses

SocioPage --> SearchBar : uses

SocioPage --> FilterButton : uses

SocioPage --> DetailButton : uses

SocioPage --> SupabaseClient : queries

AdminProyectosPage --> UploaderButton : uses

UploaderButton --> CreateUsersAPI : calls

CreateUsersAPI --> ResendService : uses

CreateUsersAPI --> SupabaseClient : uses

LoginSocioPage --> SupabaseClient : authenticates

%% Data Model Relationships

ProyectosSolidarios ||--o{ PostulacionDB : "has applications"

ProyectosSolidarios ||--o{ SocioformadorDB : "managed by"

Solicitud --> PostulacionDB : "maps to"

Project --> ProyectosSolidarios : "maps to"

```

This class diagram represents your Social Service Platform's architecture, showing:

## Core Components

\*\*Data Models\*\*: [1](#1-0) and [2](#1-1)

\*\*React Pages\*\*: The main page components like `SocioPage`, `StudentExplorarPage`, and `AdminProyectosPage` that serve as controllers

\*\*Shared Components\*\*: Reusable UI elements like `HeaderBar`, `SearchBar`, `FilterButton` used across interfaces

\*\*API Routes\*\*: Backend endpoints like `CreateUsersAPI` [3](#1-2)

\*\*Database Tables\*\*: The core entities (`ProyectosSolidarios`, `PostulacionDB`, `SocioformadorDB`) that store your application data

\*\*External Services\*\*: `SupabaseClient` for database operations and `ResendService` for email notifications

## Key Relationships

The diagram shows how components interact - for example, `SocioPage` manages `Solicitud` objects, renders `ListaSocio` components, and uses shared UI components for search and filtering functionality.

## Notes

This diagram focuses on the main structural components rather than every utility function or minor component. The relationships show both composition (components using other components) and data flow (how data models map to database tables).

Wiki pages you might want to explore:

- [Overview (alexkm174x8/SocialServicePlatform)](/wiki/alexkm174x8/SocialServicePlatform#1)