### **FRONT END**

### **1. Imagen corporativa bien definida**

Definir una identidad visual sólida ayudará a mantener coherencia en el diseño de la aplicación.

**Tareas:**

* **Logotipo:**
  + Crear un logotipo representativo de la app. Ejemplo: utilizar elementos relacionados con familias, unión, o eventos (podría incluir una casa, manos unidas, o un calendario estilizado).
  + Usar herramientas como Canva, Illustrator, o herramientas gratuitas como FreeLogoDesign.
* **Gama de colores:**
  + Elegir colores que transmitan confianza, inclusión y accesibilidad (ejemplo: tonos azules para confianza, verdes para tranquilidad, y acentos cálidos como naranja para accesibilidad).
  + Asegurarse de que la paleta sea accesible según las normas WCAG (Web Content Accessibility Guidelines).
* **Tipografías:**
  + Seleccionar una tipografía principal para títulos (ejemplo: Roboto Slab, Montserrat).
  + Usar una tipografía secundaria para textos (ejemplo: Open Sans, Lato).
  + Asegurarse de que sean legibles en dispositivos móviles.

### **2. Flujos de usuario**

Definir los pasos que seguirá el usuario en las principales funcionalidades de la app.

**Ejemplo de flujos básicos:**

1. **Registro de familia:**
   * Acceso a la pantalla de registro.
   * Completar datos (ubicación, intereses, etc.).
   * Confirmación del registro y acceso al sistema.
2. **Búsqueda de eventos:**
   * Seleccionar filtros (ubicación, intereses).
   * Navegar por una lista de eventos compatibles.
   * Ver detalles del evento y familias participantes.
   * Inscribirse al evento.
3. **Gestión de cuenta:**
   * Inicio de sesión.
   * Modificación de perfil (datos, intereses, disponibilidad).
   * Visualización de eventos a los que se ha inscrito.
4. **Creación de eventos (que lo pueda hacer una persona):**

### **3. Pantallas de la aplicación**

Diseñar y definir todas las pantallas necesarias para la iteración 1.

**Ejemplo de pantallas:**

1. **Públicas:**
   * Página de inicio/ landing page: Información general de la app, beneficios, registro.
   * Pantalla de registro/inicio de sesión.
   * Página de eventos públicos (sin detalles específicos).
2. **Privadas (solo para usuarios registrados):**
   * Panel de usuario (gestión de perfil, historial de eventos).
   * Listado de eventos filtrados.
   * Detalles del evento (familias compatibles, ubicación, intereses).

**Herramienta sugerida:**

* Usar **Figma** para crear wireframes de cada pantalla.
* Definir tamaños de letras, botones y otros componentes visuales.

### **4. Comportamiento según el estado de sesión**

Definir cómo la app se comporta según si el usuario ha iniciado sesión o no.

**Comportamiento público (sin sesión iniciada):**

* Acceso limitado a:
  + Información general de la app.
  + Registro de familias.
  + Visualización básica de eventos.

**Comportamiento privado (con sesión iniciada):**

* Acceso completo a:
  + Gestión del perfil del usuario.
  + Inscripción en eventos.
  + Historial de actividades
  + Recompensas (?)

**Componentes específicos:**

* Barra de navegación:
  + Sin sesión: Sobre nosotros, contacto organización y Botones de "Inicio de sesión" y "Registro".
  + Con sesión: Menú con acceso al perfil, eventos y logout.
* Mensajes personalizados: Mostrar el nombre de la familia una vez autenticada.

### **5. Acciones para esta etapa inicial**

1. **Asignar tareas del diseño visual:**
   * Una persona encargada de definir la identidad visual y crear wireframes iniciales.
2. **Crear los flujos de usuario:**
   * Esquematizar los pasos principales para cada funcionalidad en diagramas (puedes usar herramientas como Miro o Lucidchart).
3. **Wireframes y pantallas:**
   * Diseñar las pantallas públicas y privadas con Figma o Canva.
   * Revisar en equipo para asegurar coherencia y claridad.
4. **Definir reglas de acceso:**
   * Especificar qué componentes son visibles para usuarios con/sin sesión.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

### **Endpoints del Backend**

#### **1. Endpoint de Registro e Inicio de Sesión**

**Objetivo:** Permitir que las familias se registren y accedan a la aplicación.

**URL:** /api/auth/register  
**Método:** POST  
**Request Body (JSON):**  
{

"name": "Family Name",

"email": "example@example.com",

"password": "securepassword123",

"location": "Madrid",

"interests": ["outdoor", "music", "sports"]

}

**Response (JSON):**  
{

"message": "Registration successful",

"userId": 12345

}

**URL:** /api/auth/login  
**Método:** POST  
**Request Body (JSON):**json  
{

"email": "example@example.com",

"password": "securepassword123"

}

**Response (JSON):**  
{

"message": "Login successful",

"token": "eyJhbGciOiJIUzI1..."

}

#### **2. Endpoint para Gestión del Perfil**

**Objetivo:** Gestionar y actualizar información del perfil de las familias.

**URL:** /api/profile/{userId}  
**Método:** GET  
**Headers:**  
{

"Authorization": "Bearer <token>"

}

**Response (JSON):**  
{

"userId": 12345,

"name": "Family Name",

"email": "example@example.com",

"location": "Madrid",

"interests": ["outdoor", "music", "sports"]

}

**URL:** /api/profile/{userId}  
**Método:** PUT  
**Request Body (JSON):**  
{

"location": "Barcelona",

"interests": ["art", "cooking"]

}

**Response (JSON):**  
{

"message": "Profile updated successfully"

}

#### **3. Endpoint para Gestión de Eventos**

**Objetivo:** Listar, crear y ver detalles de eventos disponibles.

**URL:** /api/events  
**Método:** GET  
**Query Params:**  
?location=Madrid&interests=music

**Response (JSON):**  
[

{

"eventId": 1,

"title": "Music Meetup",

"date": "2025-01-10T18:00:00Z",

"location": "Madrid",

"familiesCount": 5

},

{

"eventId": 2,

"title": "Outdoor Adventure",

"date": "2025-01-12T09:00:00Z",

"location": "Madrid",

"familiesCount": 8

}

]

**URL:** /api/events  
**Método:** POST  
**Request Body (JSON):**json  
{

"title": "Cooking Class",

"date": "2025-01-15T10:00:00Z",

"location": "Madrid",

"description": "Learn to cook family recipes",

"creatorId": 12345

}

**Response (JSON):**  
{

"message": "Event created successfully",

"eventId": 3

}

**URL:** /api/events/{eventId}  
**Método:** GET  
**Response (JSON):**  
{

"eventId": 1,

"title": "Music Meetup",

"date": "2025-01-10T18:00:00Z",

"location": "Madrid",

"description": "A meetup for families who love music",

"families": [

{

"familyId": 101,

"name": "The Smiths",

"interests": ["music", "outdoor"]

},

{

"familyId": 102,

"name": "The Garcias",

"interests": ["music", "art"]

}

]

}

#### **4. Endpoint para Participación en Eventos**

**Objetivo:** Inscribirse o cancelar la participación en un evento.

**URL:** /api/events/{eventId}/join  
**Método:** POST  
**Headers:**json  
{

"Authorization": "Bearer <token>"

}

**Response (JSON):**  
{

"message": "Successfully joined the event"

}

**URL:** /api/events/{eventId}/leave  
**Método:** DELETE  
**Headers:**  
{

"Authorization": "Bearer <token>"

}

**Response (JSON):**  
{

"message": "Successfully left the event"

* }

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

### **Resumen técnico para la implementación de la iteración 1 de ParentLink**

#### **Descripción de la funcionalidad principal**

ParentLink comenzará como una aplicación de eventos donde las familias pueden registrarse con sus criterios e intereses. La plataforma utilizará esta información para recomendarles eventos locales que hayan creado dichos eventos, mostrando el número de familias con intereses similares para fomentar la interacción y la confianza. Una familia puede crear un evento.

### **Arquitectura técnica**

1. **Frontend: React.js**
   * Interfaz de usuario moderna e interactiva para:
     + Registro de familias con un formulario detallado (criterios, intereses, ubicación, etc.).
     + Listado de eventos disponibles basado en la ubicación y coincidencias de intereses.
     + Vista de detalle de cada evento, incluyendo el número de familias similares inscritas.
     + Creación de evento con un formulario (ubicación, temática, dirigido a niños de edad x-x)
   * Uso de bibliotecas como **React Router** para navegación y **Axios** para comunicación con el backend.
2. **Backend: Java Spring Boot**
   * Gestión de la lógica de negocio y conexión con la base de datos:
     + API REST para manejar solicitudes del frontend (registro, listado de eventos, creación de eventos, filtros).
     + Validación de datos al recibir información de las familias y eventos.
     + Algoritmo para encontrar coincidencias entre familias según intereses y ubicación.
   * Gestión de autenticación y autorización con **Spring Security** (para el registro seguro de usuarios).
3. **Base de datos: MySQL/PostgreSQL**
   * Tablas principales:
     + families: Para almacenar información de las familias (nombre, ubicación, intereses, etc.).
     + events: Para guardar eventos creados, su ubicación, detalles y familias inscritas.
     + matches: Relación entre familias y eventos basada en criterios comunes.
   * Consultas optimizadas para buscar eventos por ubicación y coincidencias de intereses.
4. **API REST**
   * Endpoints básicos:
     + POST /api/register-family: Registrar nueva familia con criterios e intereses.
     + GET /api/events?location=xyz&interests=abc: Listar eventos compatibles por ubicación e intereses.
     + POST /api/events/{eventId}/register: Inscribir una familia en un evento.
   * Uso de **Swagger** para documentar la API.
5. **Sistema de recomendación**
   * Implementar un sistema básico de coincidencia basado en:
     + Localización (geolocalización por ciudad o código postal).
     + Intereses específicos seleccionados por las familias.
     + Algoritmo de puntuación para clasificar eventos según relevancia para cada familia.
6. **Interfaz gráfica atractiva**
   * Diseño responsive con **CSS frameworks** como Tailwind o Material-UI.
   * Integración de mapas interactivos con **Leaflet.js** o **Google Maps API** para mostrar eventos cercanos.
7. **Seguridad y privacidad**
   * Protección de datos personales con cifrado en tránsito (HTTPS) y en reposo (en la base de datos).
   * Restricción del acceso a los datos sensibles mediante roles de usuario y permisos.

GitHub para asignar tareas y Slack para comunicación?

Ver cómo documentar?