ChatGPT

Implementación del Front-End en el entorno local

Este documento detalla cómo comenzar a implementar la interfaz de usuario de **SchoolSystem** usando las tecnologías seleccionadas (Blazor WebAssembly/PWA, Flutter/Dart y .NET MAUI) sin necesidad de salir del entorno de desarrollo actual. La idea es crear la estructura básica, servicios y componentes que puedan ser completados y probados posteriormente cuando se disponga de los entornos móviles y de escritorio.

Principios generales

- **Consistencia**: Mantenemos una experiencia coherente entre plataformas; las secciones de navegación (inicio, alumnos, maestros, padres, etc.) seguirán una estructura similar en todas las apps.
- **Separación de responsabilidades**: Cada cliente se comunica con la API a través de servicios; la lógica de negocio y la persistencia están en el backend.
- **Seguridad**: Todas las llamadas HTTP se realizarán por HTTPS y llevarán el token JWT en la cabecera `Authorization`.
- **Tiempo real y sincronización**: Incluimos un servicio para conectarse al hub SignalR, permitiendo notificaciones en tiempo real; y servicios de almacenamiento local para funcionar sin conexión.

Blazor WebAssembly / PWA

1. **Estructura del proyecto**
2. Crear un directorio `SchoolSystem.Web` dentro de la solución y, en él, un proyecto Blazor WebAssembly (*más adelante se ejecutará* `dotnet new blazorwasm`).

3. Dentro del proyecto, organizar las carpetas:

    - `Pages/` para componentes de páginas (e.g., `Login.razor`, `Alumnos.razor`, `AlumnosDetalle.razor`).
    - `Shared/` para componentes compartidos (e.g., `MainLayout.razor`, `NavMenu.razor`).
    - `Services/` para clases que consumen la API y el hub SignalR.
    - `Models/` para las clases DTO (puedes copiar las definiciones de los DTO del backend o generar sus equivalentes en C#).

4. **Servicios HTTP**

5. Crear en `Services` una clase `ApiService` que utilice `HttpClient` para realizar solicitudes a la API. Debe aceptar la URL base en el constructor y exponer métodos genéricos `GetAsync<T>`, `PostAsync<TRequest, TResponse>`, `PutAsync<TRequest>` y `DeleteAsync` que devuelvan instancias de `ApiResponse<T>` o `PagedResult<T>`.
6. Implementar un método para añadir el token JWT a la cabecera `Authorization`. El token se recuperará de `ProtectedLocalStorage`.

7. Crear servicios específicos (`AlumnoService`, `MaestroService`, etc.) que llamen a `ApiService` y devuelvan modelos de vista aptos para las páginas.

8. **Autenticación**

9. Crear `AuthenticationService` con métodos `LoginAsync` y `LogoutAsync`. `LoginAsync` enviará un `LoginDto` a `/api/auth/login` y almacenará el token y el rol en el almacenamiento protegido.

10. Implementar un `CustomAuthenticationStateProvider` que lea el token e informe a los componentes sobre el estado de autenticación.

11. **Componentes de páginas**

12. `Login.razor`: formulario con campos de usuario y contraseña, enlace al servicio de autenticación y mensajes de error basados en la respuesta de `ApiResponse`.
13. `Alumnos.razor`: lista paginada de alumnos. Utiliza `AlumnoService.GetPagedAsync(pageNumber, pageSize)` y recorre `PagedResult.Items` y `PagedResult.TotalPages` para mostrar controles de paginación.
14. `AlumnoDetalle.razor`: muestra información detallada de un alumno y permite editarla mediante formularios y validaciones en cliente.

15. **SignalR**

16. Crear en `Services` una clase `NotificationHubService` que se conecte al hub `/hubs/notifications` de la API. Implementar eventos para escuchar notificaciones (e.g., nuevas inscripciones, cambios de estado) y emitirlas a los componentes interesados.

17. **PWA y almacenamiento local**

18. Al generar el proyecto, Blazor creará un `service-worker.js`. Configurar el almacenamiento en `IndexedDB` para guardar datos de formularios y listados cuando no haya conexión.

Flutter/Dart (App para padres)

1. **Organización inicial**
2. Estructurar el proyecto en carpetas: `lib/screens/` para pantallas (home, login, alumnos), `lib/models/` para DTOs, `lib/services/` para consumo de API y `lib/providers/` o `lib/controllers/` para gestión de estado.
3. Definir modelos con `json_serializable` para mapear las respuestas de `ApiResponse<T>` y `PagedResult<T>`.

4. **Servicio API**

5. Crear una clase `ApiService` que use `http` o `dio` para llamadas GET/POST/PUT/DELETE. Añadir interceptores para colocar el token en la cabecera.
6. Crear clases de servicio específicas (`AlumnoService`, `InscripcionService`, etc.) que utilicen `ApiService`.

7. **Gestión de autenticación y roles**

8. Implementar un `LoginScreen` que envíe las credenciales y guarde el token en `flutter_secure_storage`. En base al rol (`Padre`), mostrar la pantalla de inicio con el listado de hijos.

9. Gestionar el estado global con Provider o Riverpod, almacenando token y datos de usuario.

10. **Tiempo real y sincronización**

11. Usar el paquete `signalr_core` para conectarse al hub de notificaciones.
12. Guardar datos de alumnos y asistencias en SQLite usando `sqflite` o en Hive para acceso offline y sincronizarlos cuando la conexión vuelva.

.NET MAUI (Aplicación de escritorio para escuelas)

1. **Estructura y patrón**
2. Crear un proyecto MAUI en `SchoolSystem.Desktop`. Seguir el patrón MVVM: carpetas `Views`, `ViewModels`, `Services` y `Models`.
3. Utilizar `HttpClient` compartido para llamar a la API, replicando la clase `ApiService` de Blazor.

4. **Persistencia y sincronización offline**

5. Configurar SQLite con `Microsoft.Data.Sqlite` o `SQLite-net` para guardar alumnos, maestros, grupos, etc. Implementar un servicio de sincronización que recupere los registros modificados y los envíe a la API cuando vuelva la conexión.

6. **Interfaz adaptada**

7. Usar controles nativos de MAUI (`DataGrid`, `CollectionView`) para mostrar listados y formularios. Implementar barras de herramientas y menús laterales.

Conclusiones

Con esta guía, puedes iniciar la implementación de los front-ends de SchoolSystem dentro del entorno actual sin necesidad de configurar de inmediato los entornos de ejecución de cada plataforma. Al crear la estructura de servicios, modelos y componentes, tendrás una base sólida que luego se integrará con las herramientas de compilación y despliegue cuando se disponga de las herramientas locales (.NET, Flutter, MAUI).
