**Project Summary: Edx-Briselle Enterprise Application**

**1. Project Overview**

You are developing a dynamic, enterprise-grade platform similar to Salesforce. The goal is to enable users to configure tables, fields, UI, and workflows dynamically. The stack includes:

* **Backend:** ASP.NET Core (with possible use of Node.js where needed)
* **Frontend:** React (using Tailwind CSS and Material-UI)
* **Database:** PostgreSQL
* **Authentication:** Keycloak with Google login integration
* **Real-time Sync:** Required for UI updates

**2. Key Goals & Features**

* **User Management:**
  + First user (admin/customer) is created manually.
  + Subsequent users self-register and manage their profiles.
  + User schema is dynamic, allowing fields to be added later.
* **Modular & Plugin-Based System:**
  + Components should be modularized to allow configuration via the database.
  + The UI should be configurable from the database.
* **Entity-Based Dynamic Data Model:**
  + Database includes tables like Entity, DObj, OField, DData, and Logs.
  + Users can define and update entities dynamically.
* **Authentication Flow:**
  + Using Keycloak for login (Google authentication enabled).
  + Users should be redirected to the application after login.
  + Roles & settings managed within the application but synced with Keycloak.

**3. Current Progress & Implementations**

**3.1 Backend Setup**

***Backend Configuration:***

using edx\_briselle.Server.Data;

using Microsoft.EntityFrameworkCore;

var builder = WebApplication.CreateBuilder(args);

// Add Database Context

var connectionString = builder.Configuration.GetConnectionString("DefaultConnection");

builder.Services.AddDbContext<ApplicationDbContext>(options =>

options.UseNpgsql(connectionString));

// Add services to the container.

builder.Services.AddControllers();

builder.Services.AddEndpointsApiExplorer();

builder.Services.AddSwaggerGen();

var app = builder.Build();

if (app.Environment.IsDevelopment())

{

app.UseSwagger();

app.UseSwaggerUI();

}

app.UseHttpsRedirection();

app.UseAuthorization();

app.MapControllers();

app.Run();

***CORS Issue Fix:***

* Implemented CORS policy in the backend to allow API requests from the frontend:

var MyAllowSpecificOrigins = "AllowAll";

builder.Services.AddCors(options =>

{

options.AddPolicy(name: MyAllowSpecificOrigins,

policy =>

{

policy.AllowAnyOrigin()

.AllowAnyMethod()

.AllowAnyHeader();

});

});

app.UseCors(MyAllowSpecificOrigins);

**3.2 Frontend Setup**

* Integrated React Admin Dashboard template: C:\BriselleServer\edx-briselle\template\React-tailwind-admin-dashboard-main
* Successfully fetched entity data from the backend API (http://localhost:5113/api/entity/get-entity-data).
* Fixed CORS issue blocking API requests.

**3.3 Keycloak Authentication**

* Configured First Broker Login flow instead of creating a custom one.
* After logging in with Google, Keycloak created the user, but login returned to the same page.
* Plan to ensure users are redirected to the application post-login.

**4. Next Steps**

1️⃣ **Fine-tune authentication flow in Keycloak** to ensure seamless login and redirection. 2️⃣ **Enhance database schema** to support dynamic user-defined entities. 3️⃣ **Modularize frontend components** for better configurability. 4️⃣ **Improve real-time updates** using WebSockets or another suitable technology. 5️⃣ **Optimize backend performance** and implement logging. 6️⃣ **Start work on UI-based entity management** to allow dynamic field creation.

**5. Backup & Storage Plan**

* **Code Backup:** Git repository (version-controlled updates recommended).
* **Database Backup:** PostgreSQL dumps every 24 hours.
* **Configuration Backup:** Store API keys and settings in .env or a secure config file.

This document should provide a complete snapshot of your project's current status, helping you resume development smoothly in the future. 🚀