Here’s a complete, clean **summary document** that walks through the **end-to-end integration of a .NET API with Supabase (PostgreSQL), using Angular 20** as the frontend. It includes **only essential steps** to set up the **database**, **API**, **repository pattern**, and **UI consumption**.

**✅ Fullstack Integration Guide: Supabase + .NET + Angular 20**

**1. 🛠️ Database Setup with Supabase (PostgreSQL)**

**Step 1.1 – Create a Supabase Project**

* Go to [https://supabase.com](https://supabase.com/)
* Create a project → choose free plan
* Set database name (e.g., PMS-DB)
* Save credentials

**Step 1.2 – Get Direct Connection String**

* Use the following format in .NET:

"ConnectionStrings": {

"DefaultConnection": "Host=db.<host>.supabase.co;Database=postgres;Username=postgres;Password=YOUR\_PASSWORD;Port=5432;SSL Mode=Require;Trust Server Certificate=true"

}

**2. ⚙️ .NET Backend Setup (Web API)**

**Step 2.1 – Create .NET API Project**

dotnet new webapi -n PMS.WebAPI

cd PMS.WebAPI

**Step 2.2 – Install Required Packages**

dotnet add package Npgsql.EntityFrameworkCore.PostgreSQL

dotnet add package Microsoft.EntityFrameworkCore.Design

**Step 2.3 – Create DbContext**

📄 Data/AppDBContext.cs

public class AppDBContext : DbContext

{

public AppDBContext(DbContextOptions<AppDBContext> options) : base(options) { }

public DbSet<Todo> Todos => Set<Todo>();

}

**Step 2.4 – Create Model**

📄 Models/Todo.cs

public class Todo

{

public int Id { get; set; }

public string Title { get; set; } = "";

public bool IsComplete { get; set; }

}

**3. 🧱 Use Repository Pattern (Best Practice)**

**Step 3.1 – Interface**

📄 Interfaces/ITodoService.cs

public interface ITodoService

{

Task<IEnumerable<Todo>> GetAllAsync();

Task<Todo?> GetByIdAsync(int id);

Task<Todo> AddAsync(Todo todo);

Task UpdateAsync(Todo todo);

Task DeleteAsync(int id);

}

**Step 3.2 – Repository**

📄 Repositories/TodoService.cs

public class TodoService : ITodoService

{

private readonly AppDBContext \_context;

public TodoService(AppDBContext context) => \_context = context;

public async Task<IEnumerable<Todo>> GetAllAsync() => await \_context.Todos.ToListAsync();

public async Task<Todo?> GetByIdAsync(int id) => await \_context.Todos.FindAsync(id);

public async Task<Todo> AddAsync(Todo todo) { \_context.Todos.Add(todo); await \_context.SaveChangesAsync(); return todo; }

public async Task UpdateAsync(Todo todo) { \_context.Entry(todo).State = EntityState.Modified; await \_context.SaveChangesAsync(); }

public async Task DeleteAsync(int id) { var todo = await \_context.Todos.FindAsync(id); if (todo != null) { \_context.Todos.Remove(todo); await \_context.SaveChangesAsync(); } }

}

**Step 3.3 – Controller**

📄 Controllers/TodoController.cs

[ApiController]

[Route("api/[controller]")]

public class TodoController : ControllerBase

{

private readonly ITodoService \_repo;

public TodoController(ITodoService repo) => \_repo = repo;

[HttpGet] public async Task<IActionResult> Get() => Ok(await \_repo.GetAllAsync());

[HttpPost] public async Task<IActionResult> Post([FromBody] Todo t) => Ok(await \_repo.AddAsync(t));

[HttpPut("{id}")] public async Task<IActionResult> Put(int id, [FromBody] Todo t) { if (id != t.Id) return BadRequest(); await \_repo.UpdateAsync(t); return NoContent(); }

[HttpDelete("{id}")] public async Task<IActionResult> Delete(int id) { await \_repo.DeleteAsync(id); return NoContent(); }

}

**Step 3.4 – Register Services**

📄 Program.cs

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

options.UseNpgsql(builder.Configuration.GetConnectionString("DefaultConnection")));

builder.Services.AddScoped<ITodoService, TodoService>();

**4. 📦 EF Core Migration**

dotnet ef migrations add InitialCreate

dotnet ef database update

**5. 🌐 Angular 20 Frontend Integration**

**Step 5.1 – Generate Angular Project (Standalone)**

ng new PMS\_UI --standalone --routing --style=scss

cd PMS\_UI

**Step 5.2 – Create Service**

📄 src/app/services/todo.service.ts

export class TodoService {

constructor(private http: HttpClient) {}

getTodos(): Observable<Todo[]> {

return this.http.get<Todo[]>(`${environment.apiUrl}/todo`);

}

addTodo(todo: Omit<Todo, 'id'>): Observable<Todo> {

return this.http.post<Todo>(`${environment.apiUrl}/todo`, todo);

}

updateTodo(todo: Todo): Observable<Todo> {

return this.http.put<Todo>(`${environment.apiUrl}/todo/${todo.id}`, todo);

}

deleteTodo(id: number): Observable<void> {

return this.http.delete<void>(`${environment.apiUrl}/todo/${id}`);

}

}

**Step 5.3 – Create Component**

ng generate component components/todo

📄 todo.component.ts

@Component({

selector: 'app-todo',

standalone: true,

imports: [CommonModule, ReactiveFormsModule],

templateUrl: './todo.component.html'

})

export class TodoComponent implements OnInit {

todos: Todo[] = [];

form = this.fb.group({

id: [0],

title: ['', Validators.required],

isComplete: [false]

});

constructor(private todoService: TodoService, private fb: FormBuilder) {}

ngOnInit(): void {

this.loadTodos();

}

loadTodos() {

this.todoService.getAll().subscribe(data => this.todos = data);

}

save() {

const todo = this.form.value as Todo;

if (todo.id) {

this.todoService.update(todo).subscribe(() => this.loadTodos());

} else {

this.todoService.add(todo).subscribe(() => this.loadTodos());

}

this.form.reset({ id: 0, title: '', isComplete: false });

}

edit(todo: Todo) {

this.form.setValue(todo);

}

delete(id: number) {

this.todoService.delete(id).subscribe(() => this.loadTodos());

}

}

📄 todo.component.html

<form [formGroup]="form" (ngSubmit)="save()">

<input formControlName="title" placeholder="Todo Title" />

<label>

<input type="checkbox" formControlName="isComplete" />

Completed

</label>

<button type="submit">Save</button>

</form>

<ul>

<li \*ngFor="let todo of todos">

{{ todo.title }} - {{ todo.isComplete ? '✅' : '❌' }}

<button (click)="edit(todo)">Edit</button>

<button (click)="delete(todo.id)">Delete</button>

</li>

</ul>

**Step 5.4 – Setup Routing**

📄 app.routes.ts

export const routes: Routes = [

{ path: '', component: TodoComponent },

];

📄 main.ts

bootstrapApplication(AppComponent, {

providers: [provideRouter(routes), importProvidersFrom(HttpClientModule, ReactiveFormsModule)]

});

**✅ Done!**

You now have:

* 🔗 Supabase PostgreSQL database
* 🧠 Clean .NET Web API using repository pattern
* 🌐 Angular 20 frontend consuming the API
* 📦 Fully working CRUD for Todo items