It simplifies **data access** by allowing developers to interact with databases using **.NET objects** (C# classes) instead of writing raw SQL queries.  
**Faster Development**: EF generates SQL behind the scenes

**Easier Maintenance**: Models are easier to refactor than raw SQL scripts.

**Code First Approach**

1. You define entities (e.g., User.cs, Document.cs) as simple C# classes

**2. Create DbContext**

**3.** EF Core uses a class derived from DbContext to **track and manage** your entities.

This DbContext is your **bridge between C# code and the SQL database**.

4. Register DbContext in DI Container (in Program.cs)

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

options.UseSqlServer(

builder.Configuration.GetConnectionString("DefaultConnection"),

b => b.MigrationsAssembly("DigitalDocumentLockCommon")

));

Now EF Core knows how to connect to the database using your connection string from appsettings.json.

**Commands**

dotnet ef migrations add AddUserActivityLogTable --project ../DigitalDocumentLockCommon --startup-project ../DigitalDocumentLockAPI

dotnet ef database update --project ../DigitalDocumentLockCommon --startup-project ../DigitalDocumentLockAPI

 Reads your model classes and DbContext

 Generates SQL to create the tables

 Applies those scripts to the actual database

5. EF Core **automatically translates LINQ queries** into SQL and fetches data.

Project

│

├── Models/ ← All your POCO entities

│ └── User.cs, Document.cs

│

├── Db/

│ └── AppDbContext.cs ← Maps models to database

│

├── Migrations/

│ └── Migration files ← History of schema changes

│

├── Repository/

│ └── UserRepository.cs ← Data access logic

│

├── appsettings.json ← Connection string to DB

│

├── Program.cs ← Registers EF Core services

**SignIn Functionality:**

1. When the user clicks **"Sign Up"** in your frontend (probably Angular/React/etc.), it makes an **HTTP POST request** like:  
   **POST http://localhost:5138/api/Signup**

**Content-Type: application/json**

**{**

**"firstName": "Meghana",**

**"lastName": "R",**

**"email": "meghana@email.com",**

**"password": "Strong@123"**

**}**

1. Your controller method in SignupController.cs catches the request:

 FromBody tells ASP.NET to **deserialize** the incoming JSON to your User model.

 The User object now contains all the properties filled from the frontend form.

It then calls:

var result = await \_repo.SignupAsync(user);

This goes to the repository layer.

1. Your SignupRepository.cs performs several tasks:

Validates Input

Validates Email Format

**Validates Password Strength**

Checks for:

* 1 capital letter
* 1 number
* 1 special character
* Minimum 8 characters

If validation fails, returns 400.

Hashes the Password

#### Saves to DB using EF Core

await \_ctx.Users.AddAsync(user);

await \_ctx.SaveChangesAsync();

This saves the User to the **Users** table in SQL Server.

#### Returns Response DTO

Returns a ResultDto like:

### **Back to Controller**

Back in your controller:

Appropriate response is returned to the frontend.

1. **Database Table Mapping**

In User.cs you used:

[Table("Users")]

[Column("user\_id")]

This ensures the class maps directly to SQL table/column names.

And your AppDbContext.cs declares:

public DbSet<User> Users { get; set; }

Entity Framework uses this to talk to the Users table.

Frontend Form ⟶ HTTP POST /api/Signup

↓

Controller (SignupController)

↓

Repository (SignupRepository)

↓

Validation → Hash password → Save via EF Core

↓

Database (Users table, SQL Server)

↓

Returns HTTP 200/400/409 with response message

Why brcrypt:

| **BCrypt Does** | **Why It’s Good** |
| --- | --- |

|  |  |
| --- | --- |
| One-way hash | Cannot be reversed, even if DB is stolen |

|  |  |
| --- | --- |
| Adds salt | Prevents attackers from guessing common passwords |

|  |  |
| --- | --- |
| Slow to compute | Prevents brute-force attacks |

|  |  |
| --- | --- |
| Verifiable | You can check password match during login easily |

**Login Functionality:**

**Frontend Triggers API**

* The frontend sends a **POST** request to:

POST http://localhost:<port>/api/Login/userLogin

with this JSON body: