Session_3_MCV

MVC RECAP

🧠 MVC = Model – View – Controller

MVC is an **architectural pattern** used to organize web applications.

It separates the app into three main parts so each part has a clear responsibility:

🧩 1. **Model**

Purpose:

Represents the **data**, **rules**, and **logic** of the application.

Details:

- Models describe how data looks and behaves.

- They are connected to the **database** through an ORM (like Entity Framework).
- They can contain validation rules, relationships, and small pieces of logic related to the data itself.

Typical folder: `/Models`

Common files:

- `Member.cs`
- `Trainer.cs`
- `Plan.cs`
- `ApplicationDbContext.cs`

🪟 2. View

Purpose:

Represents the **user interface (UI)** — what the user actually sees on the screen.

Details:

- Views display data that the controller sends to them.
- They are written using **HTML**, **CSS**, and **Razor syntax** (`.cshtml` files).
- A View should never contain business logic; it only focuses on presentation.

Typical folder: `/Views/{ControllerName}/`

Common files:

- `Index.cshtml`
- `Details.cshtml`
- `Edit.cshtml`
- `Create.cshtml`

🧭 3. Controller

Purpose:

Acts as the **bridge** between the Model and the View.

Details:

- Controllers handle incoming **HTTP requests** (from the browser).
- They communicate with the Model to get or update data.
- Then they send that data to the View for display.
- Each method inside a controller is called an **Action**.

Typical folder: `/Controllers`

Common files:

- `HomeController.cs`
- `MemberController.cs`
- `TrainerController.cs`

⚙️ How They Work Together

1. The **user** sends a request (e.g., clicking a link or submitting a form).
2. The **Controller** receives that request.
3. The Controller talks to the **Model** to get or change data.
4. The Controller sends that data to the **View**.
5. The **View** renders the final HTML page back to the **user**.

View Models And Services

ليه بنستخدم (view model) ؟؟

A **ViewModel** is a **data transfer object** (DTO) that acts as a **bridge between the Model and the View**.

It is designed **only for displaying or collecting data** from the user interface (UI) (يعرض ويستقبل)

- It’s **not** stored in the database.
- It doesn’t represent a database table.
- It exists only to **transfer and shape data** that your view needs.

⚙️ نبدأ بفولدر Services داخل الـ BLL

🔷 اسمه "Services" لأنه فيه الكود اللي بيعمل المنطق (Logic)

الفولدر ده بيحتوي على:

- **Interfaces** → تعريف الخدمات (contract)
- **Classes** → تنفيذ الخدمات فعليًا

مثال:

```scss
BLL
 └── Services
      ├── IMemberService.cs      ← Interface (تعريف الخدمة)
      └── MemberService.cs       ← Class (تنفيذ الخدمة)
```

🧠 الفكرة:

- `IMemberService` يقول: "إيه اللي الخدمة دي بتعمله"
- `MemberService` يقول: "إزاي بتعمله فعليًا"

والـ Controller بياخد الخدمة دي عن طريق **Dependency Injection** علشان يستخدمها بدل ما يتعامل مع الـ database مباشرة.

👁 بعد كده فولدر ViewModels

الفولدر ده مخصص لحاجة واحدة بس:

> "تجهيز البيانات اللي الواجهة (View) محتاجاها علشان تعرضها".

لكن بدل ما نحط كل الـ ViewModels عشوائي في فولدر واحد، بنرتبهم حسب الجزء اللي يخصهم (زي Member أو Plan).

مثال:

Copy code markdown

```
BLL
 └── ViewModels
      └── MemberViewModels
           ├── MemberCreateVM.cs
           ├── MemberEditVM.cs
           └── MemberDetailsVM.cs
```

🧠 الفكرة:

- `MemberCreateVM` → البيانات اللي محتاجها الـ View لما المستخدم يعمل "Add Member"
- `MemberEditVM` → البيانات الخاصة بصفحة التعديل
- `MemberDetailsVM` → البيانات اللي بتتعرض في صف لتفاصيل

vip

🔁 العلاقة بين الاتنين (Services و ViewModels)

الـ **Service** بيستخدم الـ **Models** من الـ DAL ويحوّلها إلى **ViewModels** جاهزة للعرض.

💡 مثال بسيط:

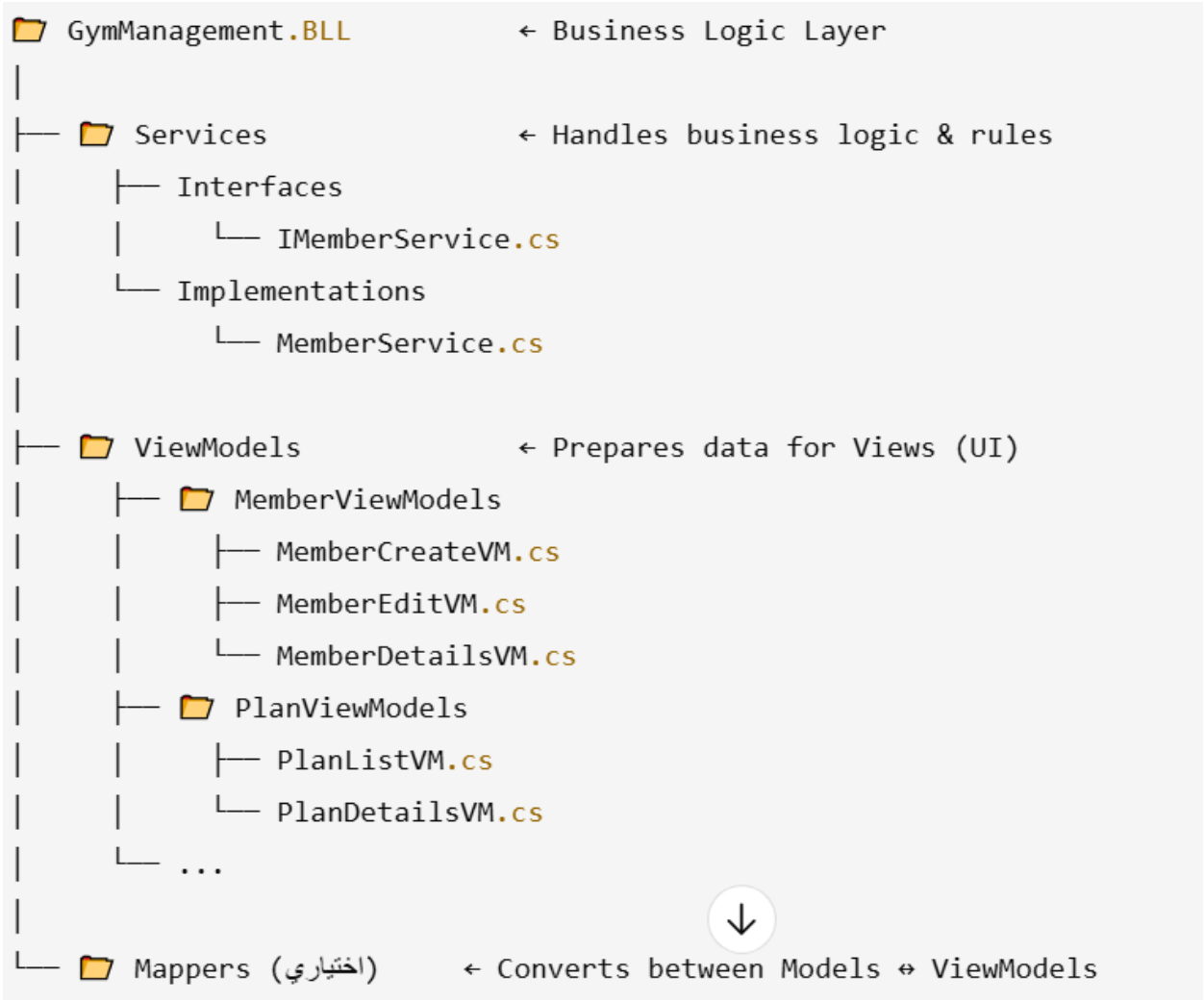
- الـ DAL فيه `Member` (بيمثل الجدول في قاعدة البيانات)
- الـ BLL فيه `MemberService` (بيجيب الداتا ويطبق القواعد)
- والـ BLL كمان فيه `MemberDetailsVM` (بيجهز شكل الداتا اللي هتروح للـ View)

السير (Chain):

Copy code pgsql

```
Controller → uses Service → gets Model data → maps to ViewModel → sends to View
```

```
GymManagementSolution
│
├── GymManagement.DAL          ← Data Access Layer
│   │
│   ├── Models                 ← Database entities (represent tables)
│   │     ├── Member.cs
│   │     ├── Plan.cs
│   │     └── Session.cs
│   │
│   ├── Repositories           ← Data access logic (CRUD)
│   │     ├── IGenericRepository.cs
│   │     └── GenericRepository.cs
│   │
│   └── ApplicationDbContext.cs  ← Manages connection to database
│
```


GymManagement.BLL ← Business Logic Layer
Services ← Handles business logic & rules
Interfaces
IMemberService.cs
Implementations
MemberService.cs
ViewModels ← Prepares data for Views (UI)
MemberViewModels
MemberCreateVM.cs
MemberEditVM.cs
MemberDetailsVM.cs
PlanViewModels
PlanListVM.cs
PlanDetailsVM.cs
...
Mappers (اختياري) ← Converts between Models ⇔ ViewModels

```
- 📂 GymManagement.PL (MVC)      ← Presentation Layer
  │
  ├── 📂 Controllers             ← Handles requests, uses Services
  │     ├── MemberController.cs
  │     ├── PlanController.cs
  │     └── HomeController.cs
  │
  ├── 📂 Views                   ← UI (.cshtml files)
  │     ├── Member
  │     │     ├── Create.cshtml
  │     │     ├── Edit.cshtml
  │     │     └── Details.cshtml
  │     └── Plan
  │
  └── 📂 wwwroot                 ← Static files (CSS, JS, images)
```

Copy code scss

```
GymManagementBLL
│
├── Services
│   ├── Classes
│   │   └── MemberService.cs       ✅ (Implementation)
│   ├── Interfaces
│   │   └── IMemberService.cs      ✅ (Contract)
│
└── ViewModels
    ├── CreateMemberViewModel.cs
    ├── HealthRecordViewModel.cs
    ├── MemberViewModel.cs
    └── Class1.cs (غالبًا مؤقّت)
```

VIP chat with GPT about 2 and 3 session of MVC

https://chatgpt.com/c/68e97374-45f0-8328-ae4e-bbd1664ca979

في حتة دي ضفنا الرقم بتاع المستخدم بالغرم انو مش ظاهر في الموقع السبب ان لازم نضيفو علشان بنحتاجو بعد كدا وبنعمل تراك للشخص بية وممكن نخفية عادي من الفورنت علشان ميظهرش في الصفحة

```
public IEnumerable<MemberViewModel> GetAllMembers()
{
    var   Members=_memberRepository.GetAll();
    if (Members == null)
        return Enumerable.Empty<MemberViewModel>();
    var MembersViewModels = Members.Select(x => new MemberViewModel
    {

        Id = x.Id,
        Name = x.Name,
        Email = x.Email,
        Phone = x.Phone,
        Photo = x.Photo,
        Gender=x.Gender.ToString(),
    });

    return MembersViewModels;
}
```