⭐ 1. Class (General Purpose Class)

A **Class** is the basic building block in C#.
It is simply a blueprint that defines **properties**, **methods**, and **behaviour**.

✓ When do we use a Class?

- To group related data and behaviour
- For any general-purpose programming logic
- Utility classes, helper classes, service classes, etc.

✓ Example

```
public class EmailService
{
    public void SendEmail(string to, string message)
    {
        // logic
    }
}
```

👉 *A class is just a structure; Model/Entity/DTO are specific types of classes.**

⭐ 2. Model

A **Model** represents **data used by the application**—especially in **MVC / Razor Pages / API**.
It usually maps to **what your application needs**, not necessarily the database.

✓ When do we use Models?

- To transfer data between **Controller ⇆ View**
- To represent business objects inside the application
- To validate input using **Data Annotations**

✓ Example (ViewModel / API Model)

```
public class RegisterModel
```

```
{
    public string Username { get; set; }
    public string Email { get; set; }

    [MinLength(6)]
    public string Password { get; set; }
}
```

👉 **Models serve the UI or business logic, not the database.**

⭐ **3. Entity (Database Entity)**

An **Entity** represents a **table in the database**.
Used mostly in **Entity Framework Core**.

✓ **When do we use Entities?**

- For **ORM mapping** to database tables
- Represents rows/columns
- Contains keys, relationships, constraints

✓ **Example (EF Core Entity)**

```
public class UserEntity
{
    public int Id { get; set; }  // Primary key
    public string Username { get; set; }
    public string Email { get; set; }
}
```

👉 **Entities = Database objects.**
👉 **Models = Application objects.**

These two may look similar but serve different layers.

⭐ 4. DTO (Data Transfer Object)

A **DTO** is a simple object used to **transfer data between layers**
—usually **Controller ↔ Service**, **API ↔ Client**, or **Microservices**.

✓ When do we use DTOs?

- To send **only necessary fields** to clients
- To hide internal structure (security)
- To avoid over-posting attacks
- To shape API response or request format
- To improve performance—no unnecessary/large data

Example (DTO)

```
public class UserDto
{
    public string Username { get; set; }
    public string Email { get; set; }
}
```

👉 A DTO usually does **not** contain:

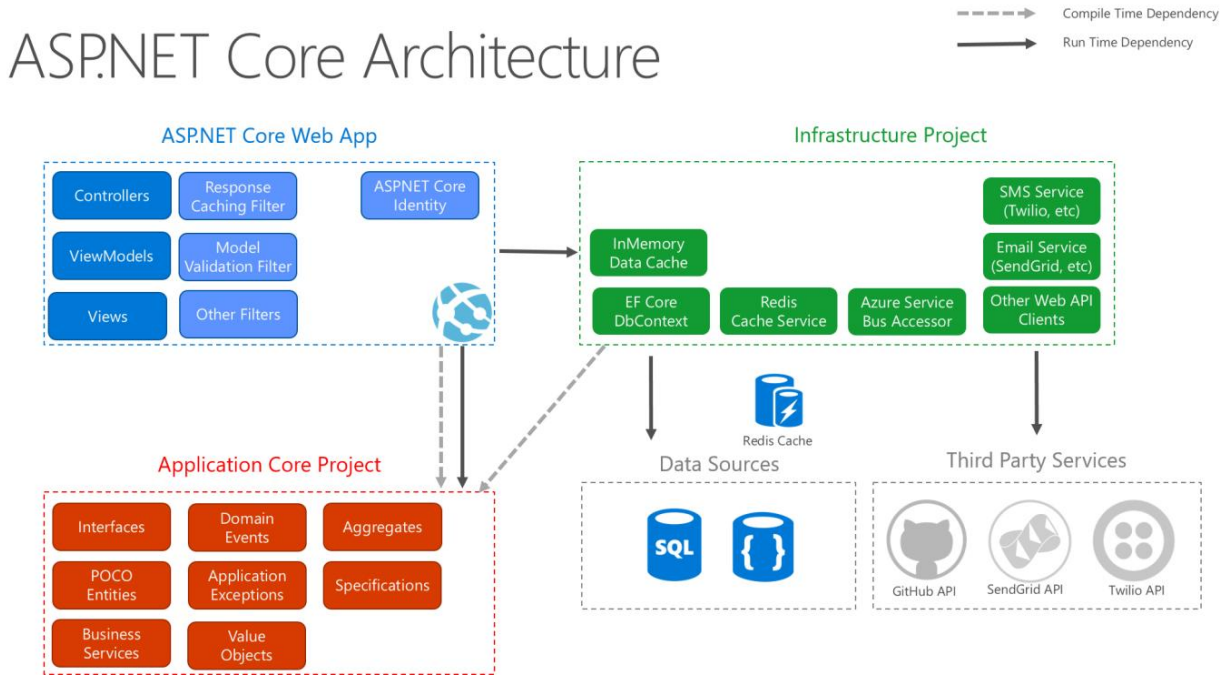
- Business logic
- Methods (except minimal validation)
- Navigation properties

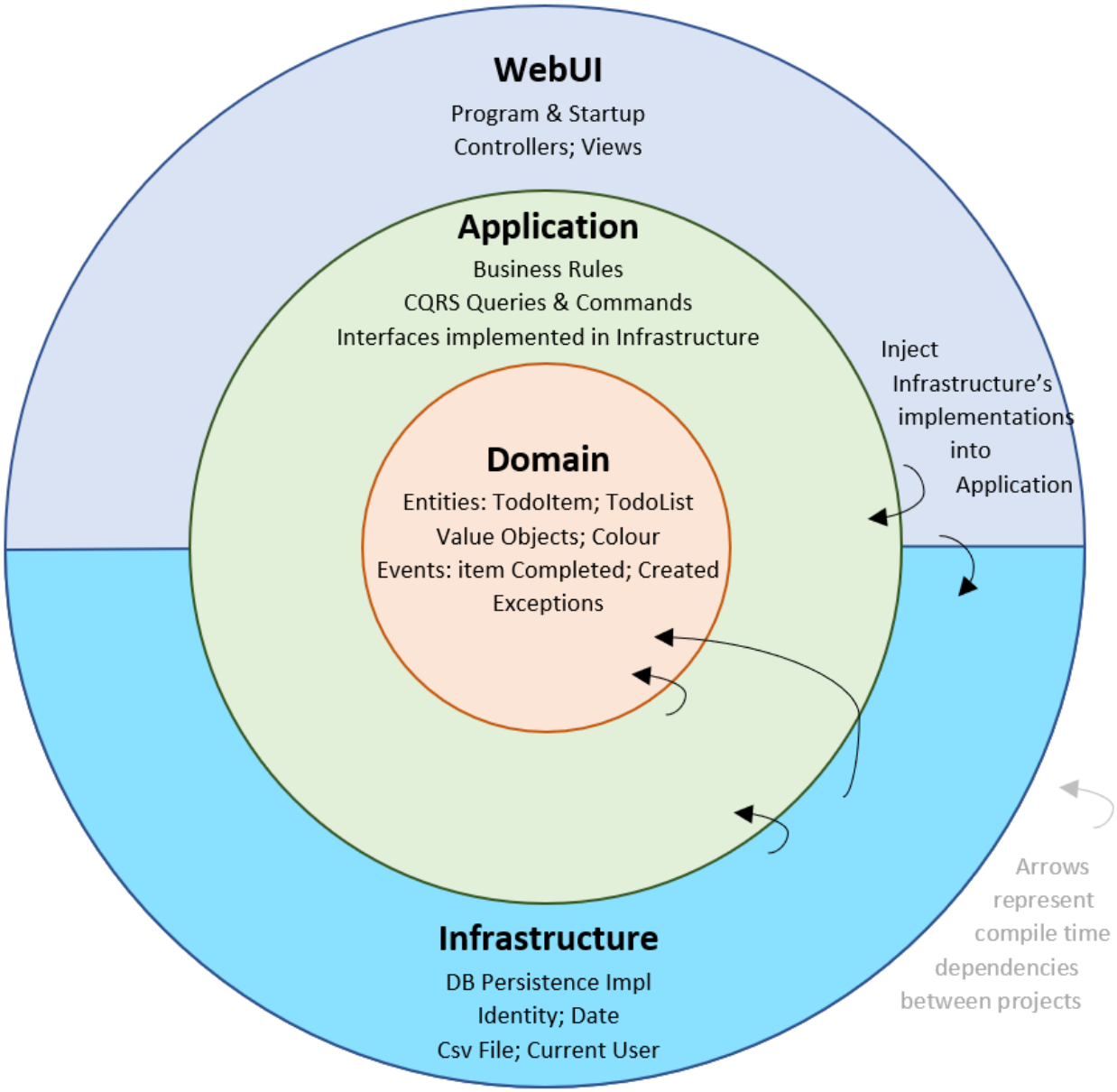
🎯 Quick Comparison Table

Concept	Layer	Purpose	Maps to DB?	Contains Logic?
Class	General	Code structure	❌	✓ (optional)
Model	UI/Business	Data for UI/API validations	❌	Minimal

Concept Layer	Purpose	Maps to DB?	Contains Logic?
Entity	Data Access Maps to DB table (ORM)	✓	❌ (usually)
DTO	API/Service Transfer data between layers	❌	❌

🎓 Real ASP.NET Core Example (Flow)





User Registration Flow

1. **DTO** (Client Request → API)
2. {
3. "username": "santhosh",
4. "email": "s@gmail.com",
5. "password": "123456"
6. }

7. **Model** (For validation/business logic inside API)
8. public class RegisterModel { ... }
9. **Entity** (To save in DB)
10. public class UserEntity { ... }
11. **Class/Service**
12. public class UserService
13. {
14. public void Register(RegisterModel model) { ... }
15. }

👉 Each layer uses an object for its own purpose.