React - .NET Core

App Setup and Basic Information

Contents

[Prerequisites 3](#_Toc70067508)

[Backend 3](#_Toc70067509)

[Setup 3](#_Toc70067510)

[Creating Components 3](#_Toc70067511)

[Referencing Components 3](#_Toc70067512)

[Install Entity Framework 4](#_Toc70067513)

[Install Required Packages via NuGet 4](#_Toc70067514)

[Git 4](#_Toc70067515)

[Architecture 5](#_Toc70067516)

[Components 5](#_Toc70067517)

[Initialization Information 5](#_Toc70067518)

[Components 7](#_Toc70067519)

[API 7](#_Toc70067520)

[Controllers 7](#_Toc70067521)

[Persistence 8](#_Toc70067522)

[Entity Framework 8](#_Toc70067523)

[Database connection 8](#_Toc70067524)

[ORM 8](#_Toc70067525)

[Commands 9](#_Toc70067526)

[Used Patterns 10](#_Toc70067527)

[Error Handling 10](#_Toc70067528)

[HTTP responses 10](#_Toc70067529)

[Approaches 10](#_Toc70067530)

[Frontend 12](#_Toc70067531)

[Setup 12](#_Toc70067532)

[Create React App 12](#_Toc70067533)

[Install Packages 12](#_Toc70067534)

[Clean Architecture 13](#_Toc70067535)

[How the flow of control will be implemented with the Mediator pattern 14](#_Toc70067536)

# Prerequisites

Latest versions of following tools:

* Rider IDE
* Node JS
* npm
* .NET SDK
* .NET Runtime

# Backend

## Setup

### Creating Solutions & Adding Projects to them

- cd “Project folder”

- dotnet new sln

- dotnet new webapi -n Api

- dotnet new classlib -n Application

- dotnet new classlib -n Domain

- dotnet new classlib -n Persistence

FaebuG@DESKTOP-GLOFFKS MINGW64 /e/dev/workspace/dotnet-react-demo/Reactivities

$ dotnet sln add Api/

Project `Api\Api.csproj` added to the solution.

FaebuG@DESKTOP-GLOFFKS MINGW64 /e/dev/workspace/dotnet-react-demo/Reactivities

$ dotnet sln add Application/

Project `Application\Application.csproj` added to the solution.

FaebuG@DESKTOP-GLOFFKS MINGW64 /e/dev/workspace/dotnet-react-demo/Reactivities

$ dotnet sln add Domain/

Project `Domain\Domain.csproj` added to the solution.

FaebuG@DESKTOP-GLOFFKS MINGW64 /e/dev/workspace/dotnet-react-demo/Reactivities

$ dotnet sln add Persistence/

Project `Persistence\Persistence.csproj` added to the solution.

FaebuG@DESKTOP-GLOFFKS MINGW64 /e/dev/workspace/dotnet-react-demo/Reactivities

$ dotnet sln list

Project(s)

----------

Api\Api.csproj

Application\Application.csproj

Domain\Domain.csproj

Persistence\Persistence.csproj

### Referencing Components

References are stored in the corresponding .csproj file.

FaebuG@DESKTOP-GLOFFKS MINGW64 /e/dev/workspace/dotnet-react-demo/Reactivities

$ cd Api/

FaebuG@DESKTOP-GLOFFKS MINGW64 /e/dev/workspace/dotnet-react-demo/Reactivities/Api

$ dotnet add reference ../Application/

Reference `..\Application\Application.csproj` added to the project.

FaebuG@DESKTOP-GLOFFKS MINGW64 /e/dev/workspace/dotnet-react-demo/Reactivities/Api

$ cd ..

FaebuG@DESKTOP-GLOFFKS MINGW64 /e/dev/workspace/dotnet-react-demo/Reactivities

$ cd Application/

FaebuG@DESKTOP-GLOFFKS MINGW64 /e/dev/workspace/dotnet-react-demo/Reactivities/Application

$ dotnet add reference ../Domain/

Reference `..\Domain\Domain.csproj` added to the project.

FaebuG@DESKTOP-GLOFFKS MINGW64 /e/dev/workspace/dotnet-react-demo/Reactivities/Application

$ dotnet add reference ../Persistence/

Reference `..\Persistence\Persistence.csproj` added to the project.

FaebuG@DESKTOP-GLOFFKS MINGW64 /e/dev/workspace/dotnet-react-demo/Reactivities/Application

$ cd ..

FaebuG@DESKTOP-GLOFFKS MINGW64 /e/dev/workspace/dotnet-react-demo/Reactivities

$ cd Persistence/

FaebuG@DESKTOP-GLOFFKS MINGW64 /e/dev/workspace/dotnet-react-demo/Reactivities/Persistence

$ dotnet add reference ../Domain/

Reference `..\Domain\Domain.csproj` added to the project.

### E:\dev\workspace\dotnet-react-demo\Reactivities>dotnet new classlib -n Infrastructure

### The template "Class library" was created successfully.

### Processing post-creation actions...

### Running 'dotnet restore' on Infrastructure\Infrastructure.csproj...

### Determining projects to restore...

### Restored E:\dev\workspace\dotnet-react-demo\Reactivities\Infrastructure\Infrastructure.csproj (in 53 ms).

### Restore succeeded.

### E:\dev\workspace\dotnet-react-demo\Reactivities>dotnet sln add Infrastructure

### Project `Infrastructure\Infrastructure.csproj` added to the solution.

### E:\dev\workspace\dotnet-react-demo\Reactivities>cd Infrastructure

### E:\dev\workspace\dotnet-react-demo\Reactivities\Infrastructure>dotnet add reference ../Application

### Reference `..\Application\Application.csproj` added to the project.

### E:\dev\workspace\dotnet-react-demo\Reactivities\Infrastructure>cd ..

### E:\dev\workspace\dotnet-react-demo\Reactivities\Domain>cd API

### E:\dev\workspace\dotnet-react-demo\Reactivities\Api>dotnet add reference ../Infrastructure

### Reference `..\Infrastructure\Infrastructure.csproj` added to the project.

### E:\dev\workspace\dotnet-react-demo\Reactivities\Api>dotnet restore

### Determining projects to restore...

### Restored E:\dev\workspace\dotnet-react-demo\Reactivities\Infrastructure\Infrastructure.csproj (in 231 ms).

### Restored E:\dev\workspace\dotnet-react-demo\Reactivities\Api\Api.csproj (in 231 ms).

### Restored E:\dev\workspace\dotnet-react-demo\Reactivities\Application\Application.csproj (in 231 ms).

### 2 of 5 projects are up-to-date for restore.

### Install Entity Framework

1. Go to <https://www.nuget.org/packages/dotnet-ef>
2. Run the installation command for the newest verison:

dotnet tool install --global dotnet-ef --version x.x.x

1. If already installed, update instead:

dotnet tool update --global dotnet-ef --version x.x.x

1. Should be listed now with the following command:

FaebuG@DESKTOP-GLOFFKS MINGW64 /e/dev/workspace/dotnet-react-demo/Reactivities

$ dotnet tool list --global

Package Id Version Commands

--------------------------------------

dotnet-ef 5.0.4 dotnet-ef

### Install Required Packages via NuGet

Open NuGet in Rider via Tools | NuGet.

Install following packages with version of your runtime for corresponding component:

|  |  |
| --- | --- |
| **Package** | **Project** |
| microsoft.entityframeworkcore.sqlite | Persistence |
| Microsoft.EntityFrameworkCore.Design | API |
| mediatr.extensions.microsoft.dependencyinjection | Application |
| AutoMapper.Extensions.Microsoft.DependencyInjection | Application |
| FluentValidation.AspNetCore | Application |
| Microsoft.AspNetCore.Identity.EntityFrameworkCore | Domain |
| Microsoft.IdentityModel.Tokens | API |
| System.IdentityModel.Tokens.Jwt | API |
| Microsoft.AspNetCore.Authentication.JwtBearer | API |
| CloudinaryDotNet | Infrastructure |
| Npgsql.EntityFrameworkCore.PostgreSQL | Persistence |

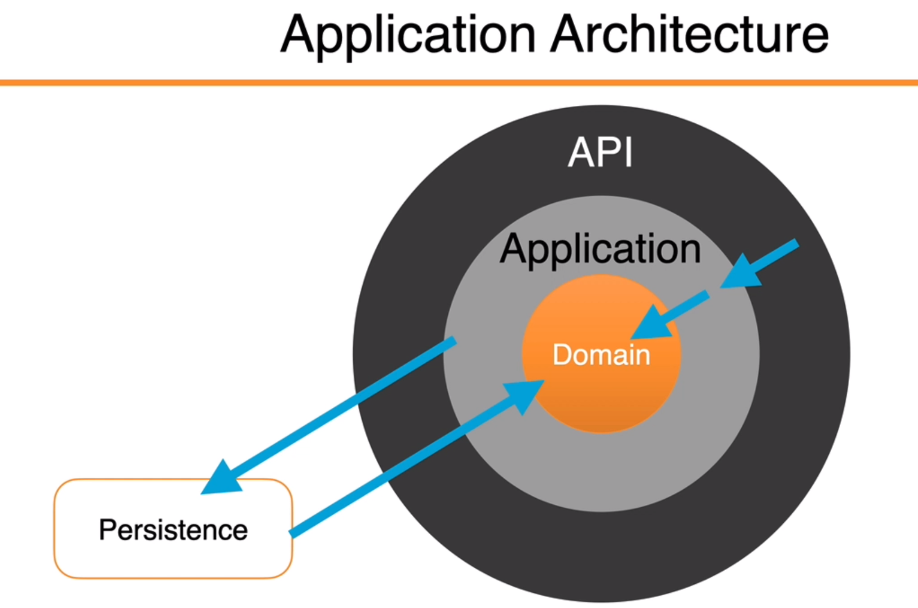
### 

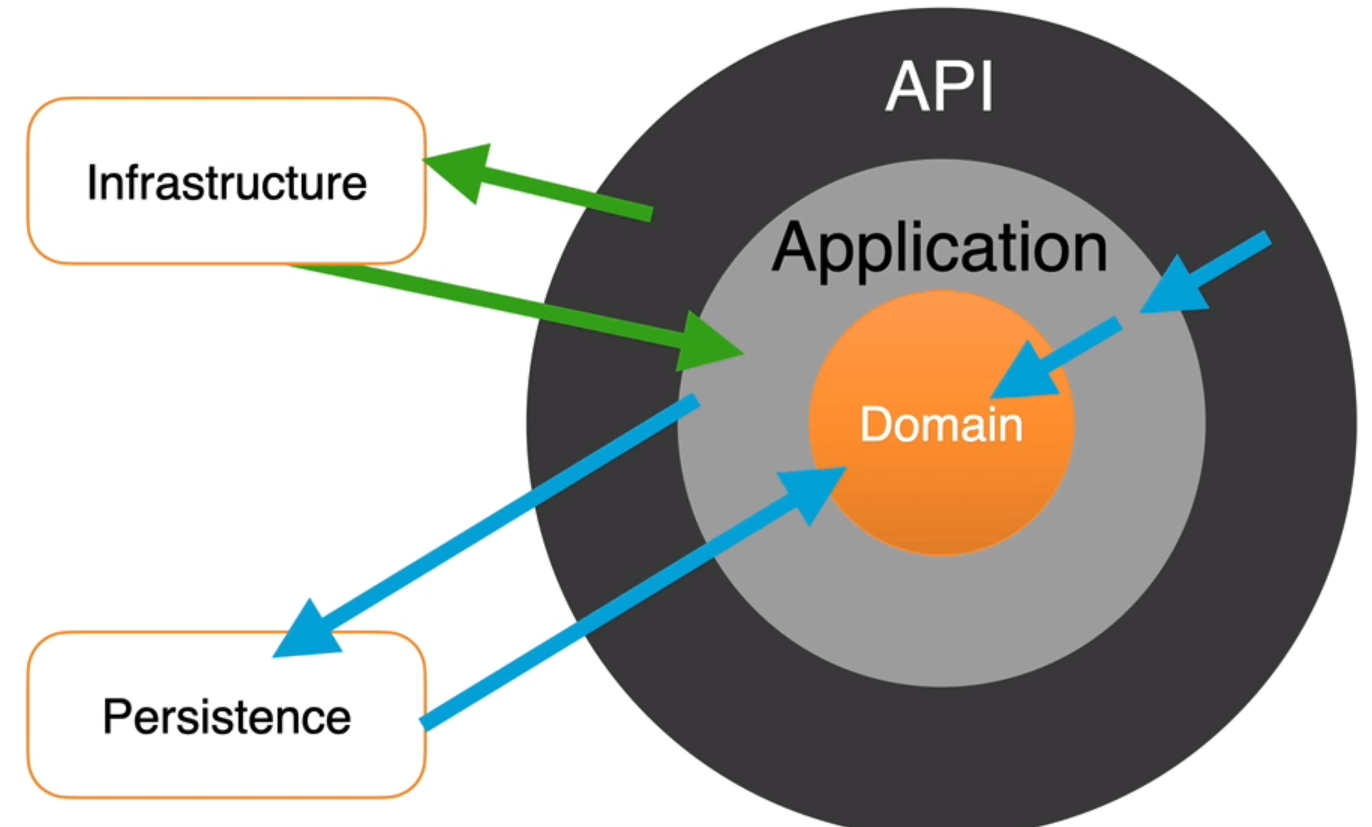
### Git

Run following commands:

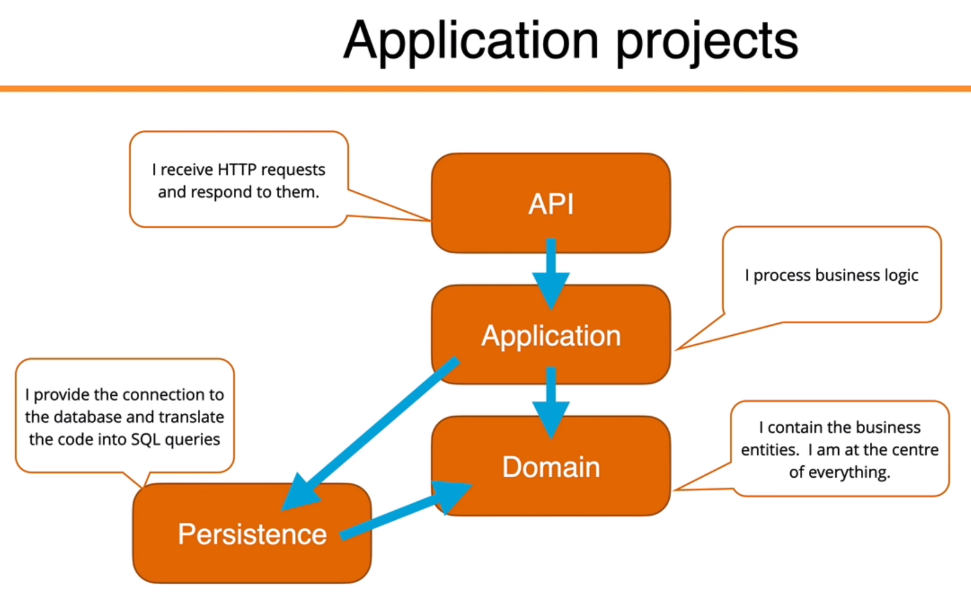
* git init
* dotnet new gitignore

## Architecture





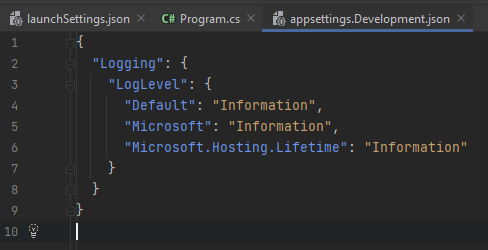
## Components



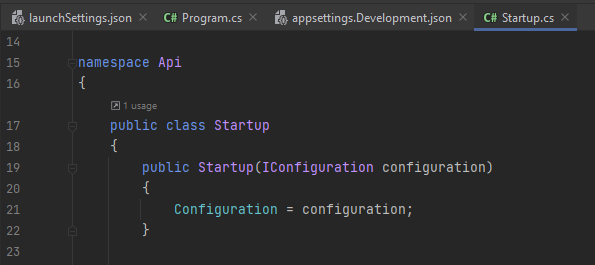
## Initialization Information

**Starting Component > Api**

* It will launch the Application over a profile in Properties/launchSettings.json. When launching in VSCode “IIS Express”, over command line “API” profile. In development, https is not needed in "applicationUrl” in profile, to keep it as simple as possible.
* Api.Program.Main(args) will be executed.
* Kestrel (default ASP.NET core webserver & host of our app) will be launched. (section 2 – 5.)
* appsettings.Development.json is used in development mode. appsettings.json is used regardless of the mode. We’ll change Microsoft LogLevel to “Information” so Entity Framework Query information will be shown in dev mode.



* This configuration data will be injected into the Startup class through its constructor.

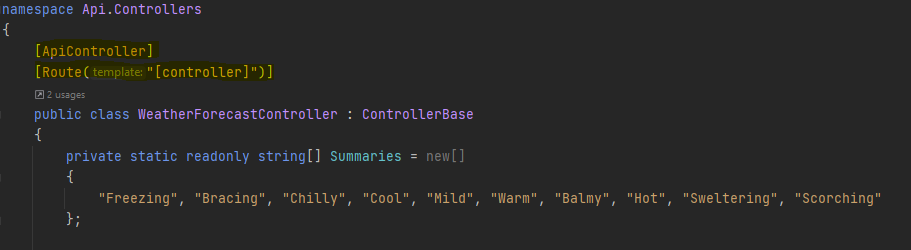


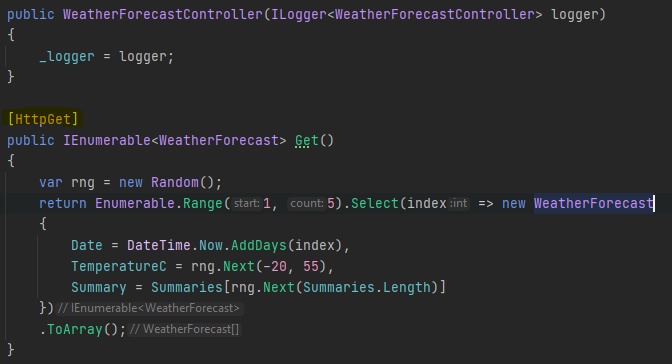
* Middleware can be added through the Configure method.
* HTTP Requests will call the controller endpoints from the Api component.

## Components

### API

#### Controllers

Attributes:

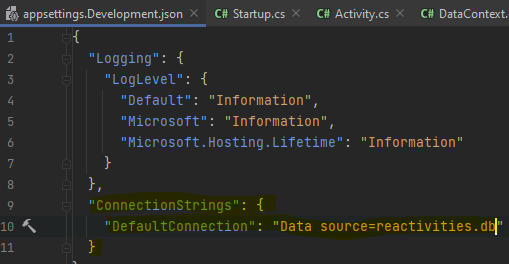
An endpoint:

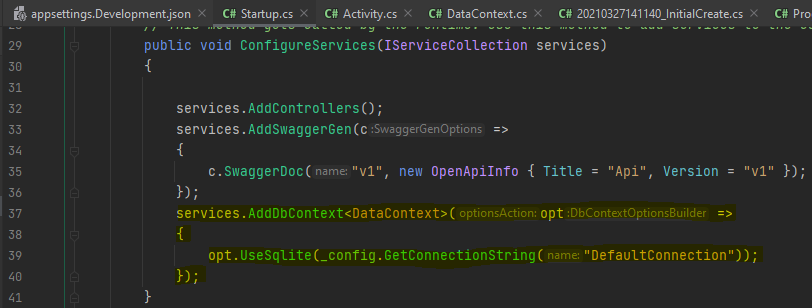
Get requests sent to this controller will invoke the “IEnumerable<WeatherForecast> Get()” function.

### Persistence

#### Entity Framework

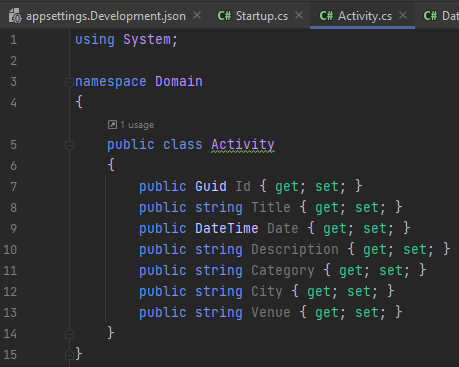
##### Database connection

Adding database connection to the config:

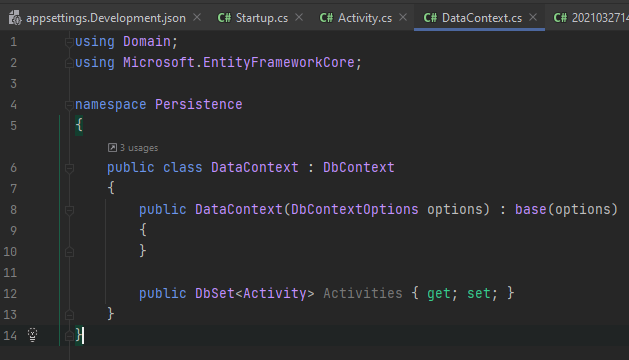
Added the db connection to our contained services.

##### ORM

Our domain entities specify the columns which will be generated in our migration and the name of the table:



Our data context will operate as a service and will query and save instances of our domain entity:



##### Commands

s flag: specifies the starter component

p flag: specifies where the data context is located

* Create migration: dotnet ef migrations add InitialCreate -p Persistence -s API
  + Add column: dotnet ef migrations add AddCanceledProperty -p Persistence -s Api
* Drop database: dotnet ef database drop -s API -p Persistence

##### Used Patterns

**CQRS**

“The Command and Query Responsibility Segregation (CQRS) pattern separates read and update operations for a data store. Implementing CQRS in your application can maximize its performance, scalability, and security.”

**Mediator Pattern**

“The Mediator pattern in C# enables objects to communicate, without knowing each other's identities. It also encapsulates a protocol that objects can follow. You can think of a Mediator object as a kind of a coordinator; that handles traffic between appropriate parties based on its own logic.”

“With clean architecture we are mainly concerned with keeping the different layers separate from each other.   The Application layer should not know anything about the outer layers, such as the API for example.    Mediator allows us to send an object from the API to the Application layer, and the Application layer handler sends an object back to the API.   The idea being that we could switch the API to (lets say) a console app without changing any of the logic in the Application layer.   ”

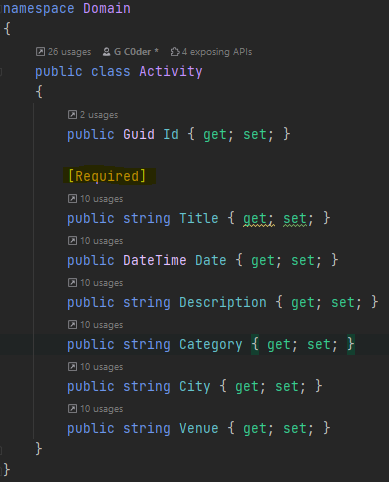
## Error Handling

### HTTP responses

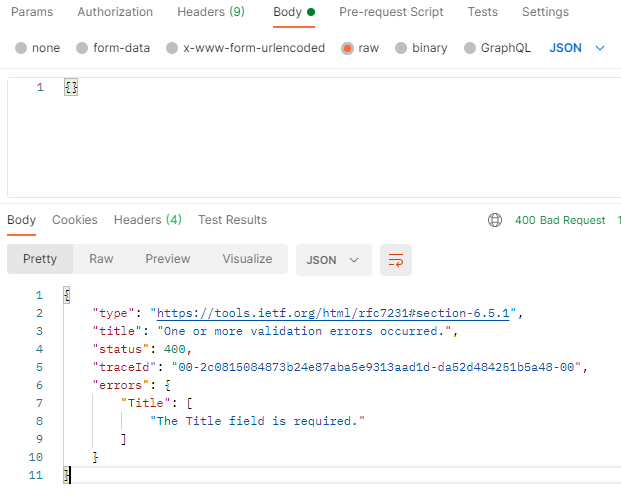


### Approaches

Setting fields as required in the business entities



In this case an empty request body results in the following:



Our “ApiController” Attribute automaticly sends a HTTP 400 response if a requirement is not met:



# Frontend

## Setup

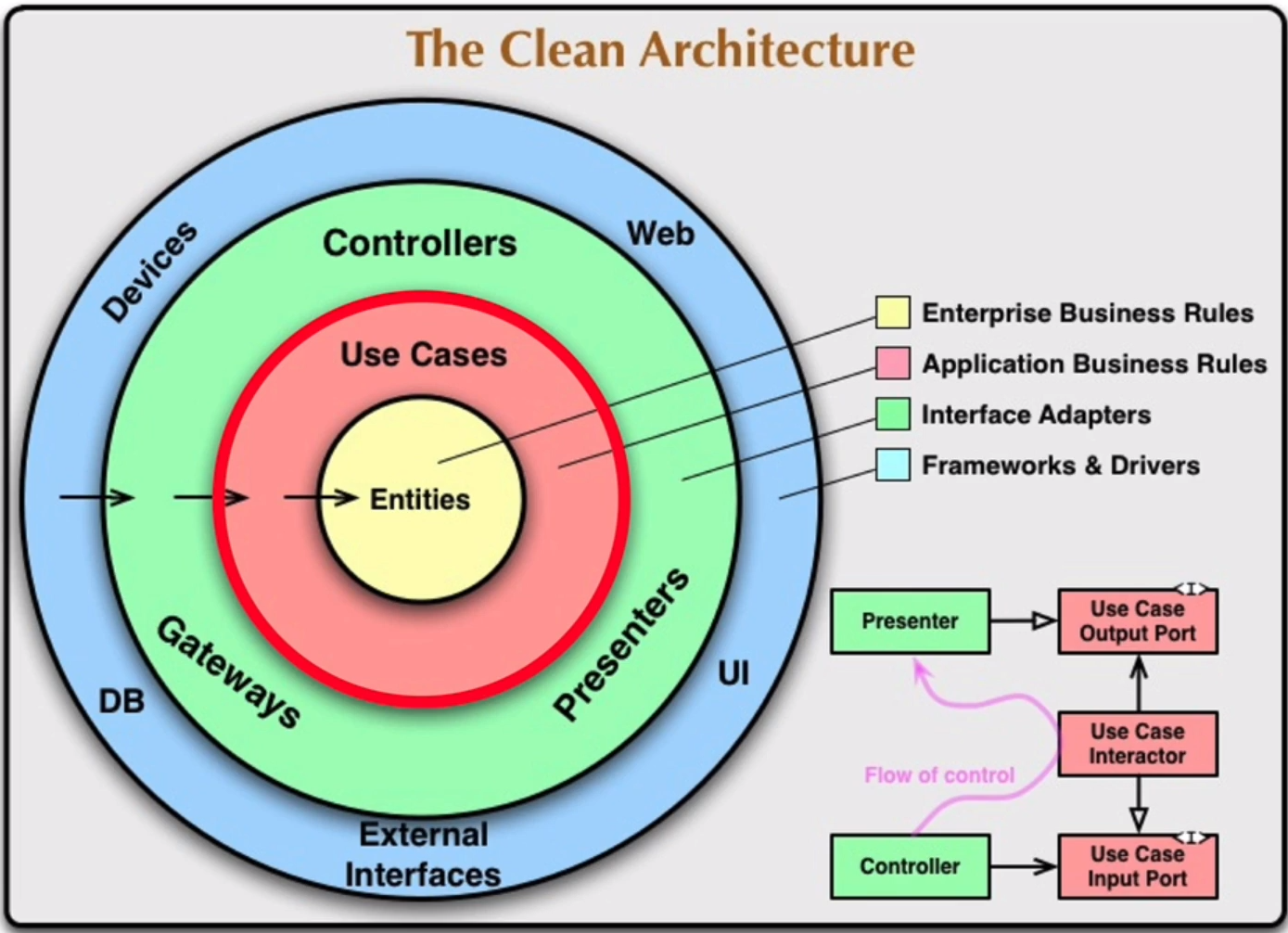
### Create React App

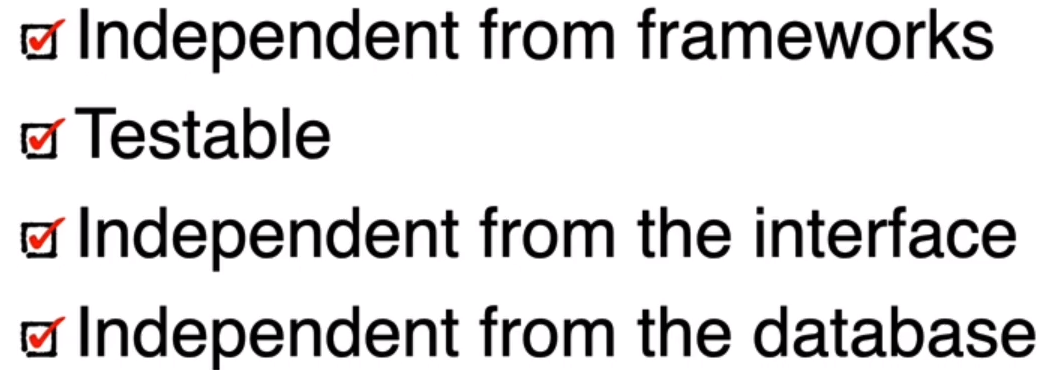
npx create-react-app client-app --use-npm --template typescript

### Install Packages

* npm install eslint --save-dev
  + npm install eslint-plugin-react --save-dev
* npm install semantic-ui-react semantic-ui-css
* npm install uuid
  + npm i --save-dev @types/uuid
* npm install mobx mobx-react-lite
* npm install react-router-dom
  + npm install @types/react-router-dom --save-dev
* npm install react-calendar
  + npm install @types/react-calendar
* npm install react-toastify
* npm install formik
* npm install yup
  + npm install @types/yup --save-dev
* npm install react-datepicker
  + npm install @types/react-datepicker -–save-dev
* npm install [date-fns@2.16.1](mailto:date-fns@2.16.1)
* npm install react-dropzone
* npm install react-cropper
* npm install react-text-truncate
  + npm install @types/react-text-truncate --save-dev
* npm install @microsoft/signalr
* npm install react-infinite-scroller
  + npm install @types/react-infinite-scroller

# Clean Architecture





# How the flow of control will be implemented with the Mediator pattern

