NOVEMBER 7, 2020

COMMANDER

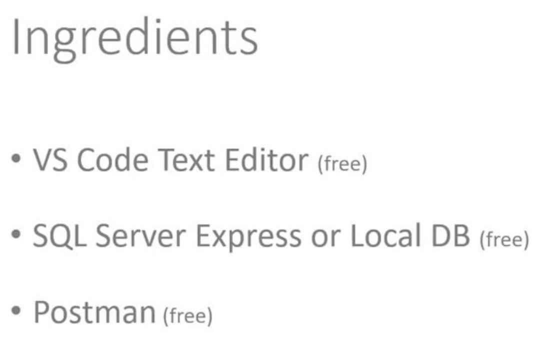
.NET CORE, MVC, REST API

CALIN DORAN
SOFTWARE DEVELOPER
Personal Project

Table of Contents

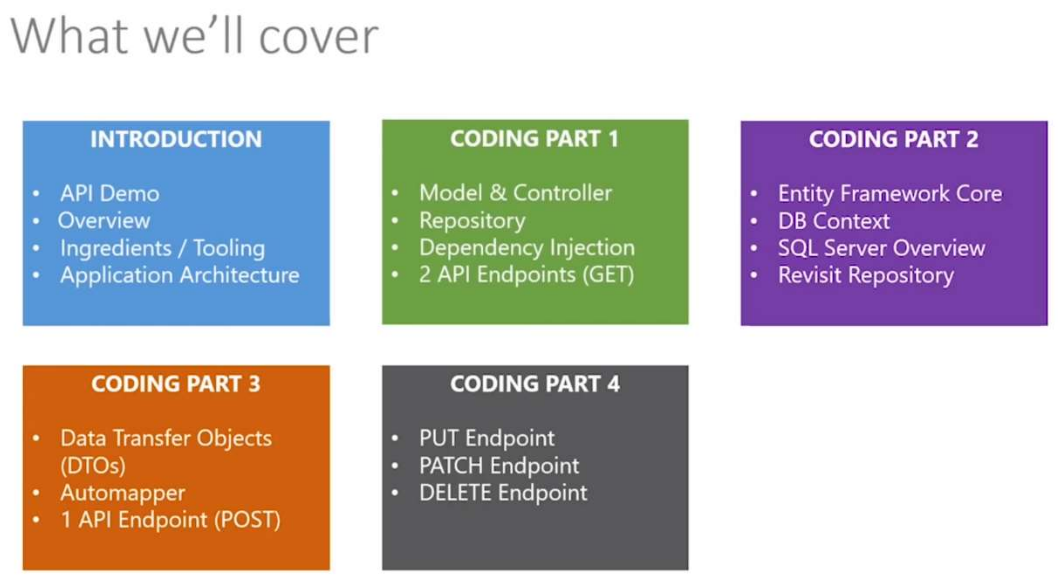
Introduction

In this project the developer aims to create an application that will provide commands for different platforms such as EF Core, .NET Core and so on. It will keep up commands for said platforms and allow the user to easily choose from which platform they need commands for. It can be difficult as a developer to try and remember every different command used on a daily basis, this helps to solve this problem.



Ingredients

- VS Code Text Editor (free)
- SQL Server Express or Local DB (free)
- Postman (free)



What we'll cover

INTRODUCTION

- API Demo
- Overview
- Ingredients / Tooling
- Application Architecture

CODING PART 1

- Model & Controller
- Repository
- Dependency Injection
- 2 API Endpoints (GET)

CODING PART 2

- Entity Framework Core
- DB Context
- SQL Server Overview
- Revisit Repository

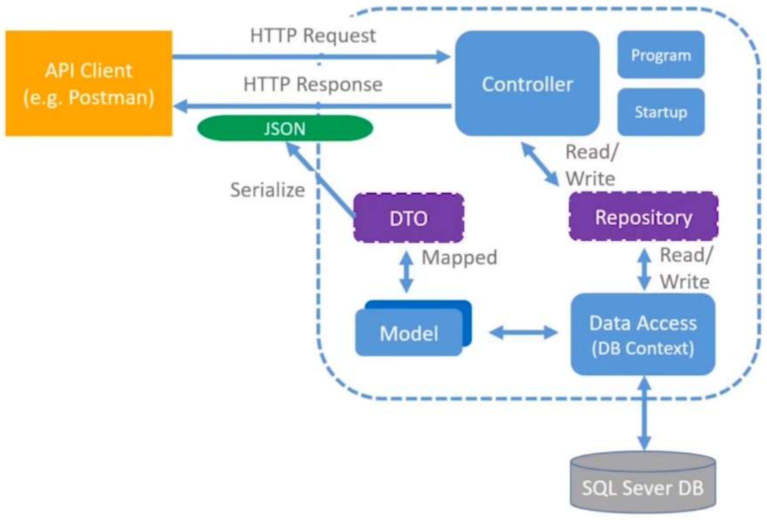
CODING PART 3

- Data Transfer Objects (DTOs)
- Automapper
- 1 API Endpoint (POST)

CODING PART 4

- PUT Endpoint
- PATCH Endpoint
- DELETE Endpoint

Application Architecture

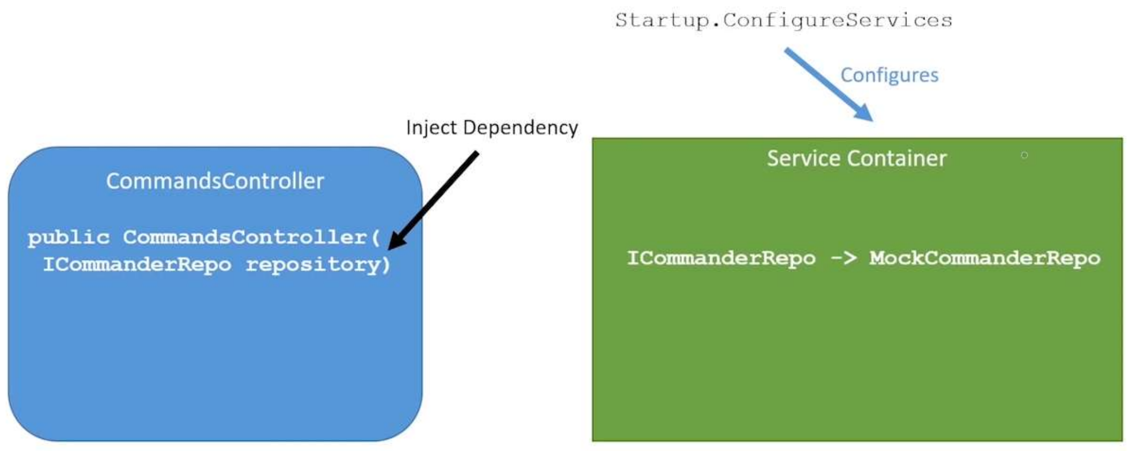


APIs & Interfaces

Our API End Points

URI	Verb	Operation	Description	Success	Failure
/api/commands	GET	READ	Read all resources	200 OK	400 Bad Request 404 Not Found
/api/commands/{id}	GET	READ	Read a single resource	200 OK	400 Bad Request 404 Not Found
/api/commands	POST	CREATE	Create a new resource	201 Created	400 Bad Request 405 Not Allowed
/api/commands/{id}	PUT	UPDATE	Update an entire resource	204 No Content	..
/api/commands/{id}	PATCH	UPDATE	Update partial resource	204 No Content	..
/api/commands/{id}	DELETE	DELETE	Deletes a single resource	200 OK 204 No Content	..
~~/api/commands~~	~~DELETE~~	~~DELETE~~	Delete all resources	200 OK 204 No Content	..

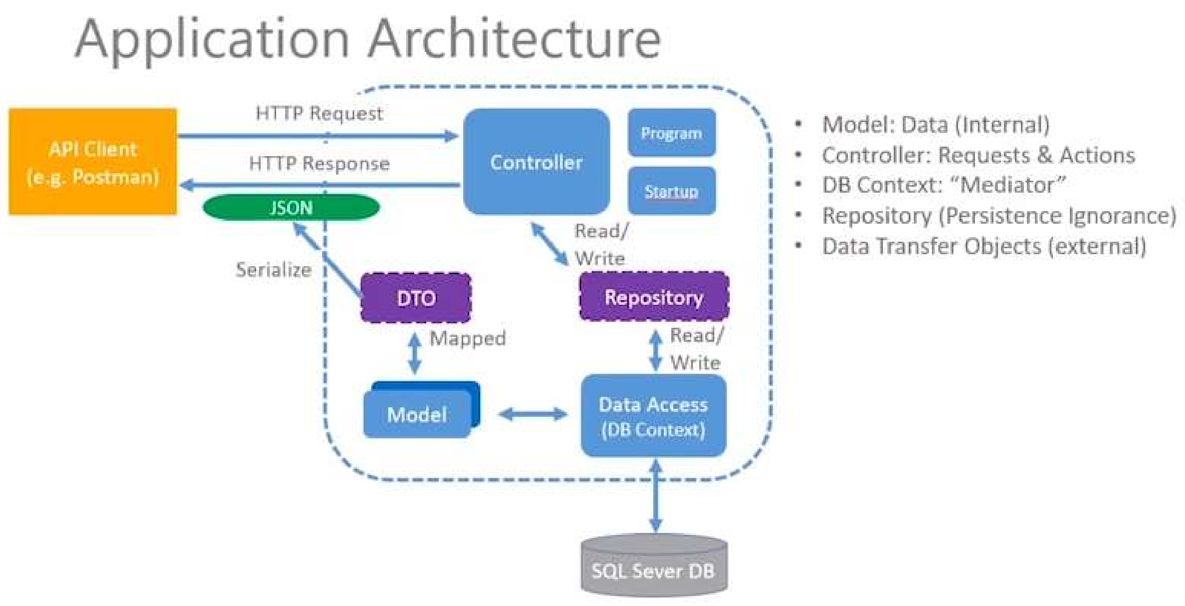
Dependency Injection


Startup.ConfigureServices
Configures
Inject Dependency
CommandsController
public CommandsController(
ICommanderRepo repository)
Service Container
ICommanderRepo -> MockCommanderRepo

Service Lifetimes

- AddSingleton
  - Same for every request
- AddScoped
  - Created once per client request
- Transient
  - New instance created every time

Database set-up and Migrations



Nuget.org

Install via command line with dotnet add package.

Packages needed:

1. Microsoft.EntityFrameworkCore
2. Microsoft.EntityFrameworkCore.Design
3. Microsoft.EntityFrameworkCore.SqlServer

No need for the –version as it will install the latest without that part of the command.

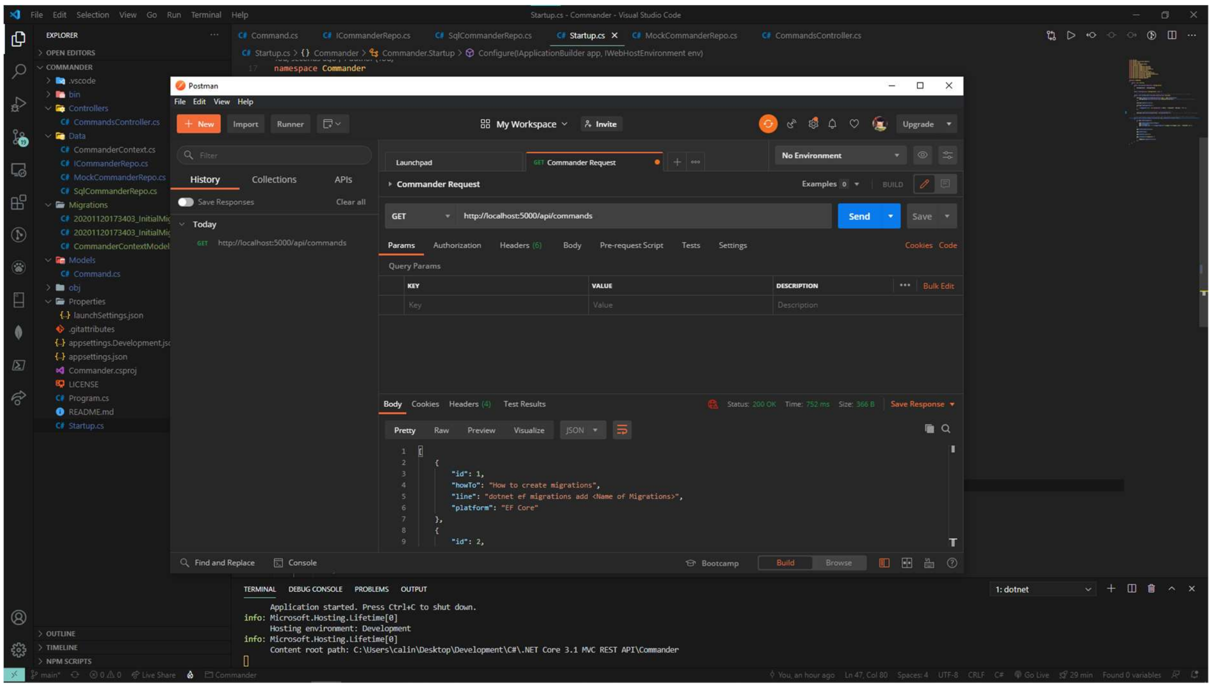
If there is an error with dotnet ef then we will need this command:

"dotnet tool install --global dotnet-ef"

"dotnet ef migrations add InitialMigration" to create a migration file.

Modified the constraints of our model so that its not null and maxlength is 250. Once this is done, we can run the command "dotnet ef database update" and it should create the database CommanderDB.

From here we can look at the database and edit/update fields with data for our commands table. And we are done with the database, model and migration setup.



Postman
GET
http://localhost:5000/api/commands
Send
Status: 200 OK Time: 752 ms Size: 366 B

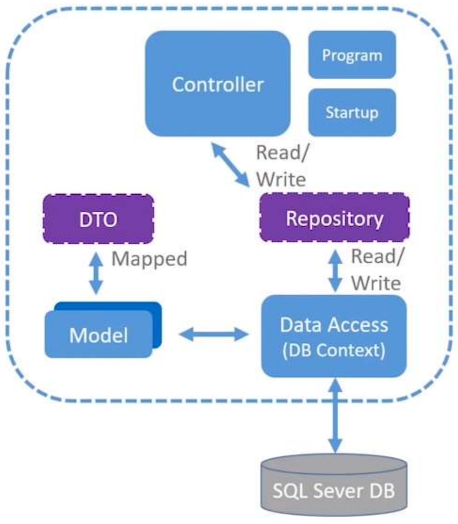
```
[
    {
        "id": 1,
        "howTo": "How to create migrations",
        "line": "dotnet ef migrations add <Name of Migrations>",
        "platform": "EF Core"
    },
    {
        "id": 2,
```

```
      Application started. Press Ctrl+C to shut down.
info: Microsoft.Hosting.Lifetime[0]
      Hosting environment: Development
info: Microsoft.Hosting.Lifetime[0]
      Content root path: C:\Users\calin\Desktop\Development\C#\.NET Core 3.1 MVC REST API\Commander
```

We are getting actual information from our sqldb CommanderDB using the right context, this is great.

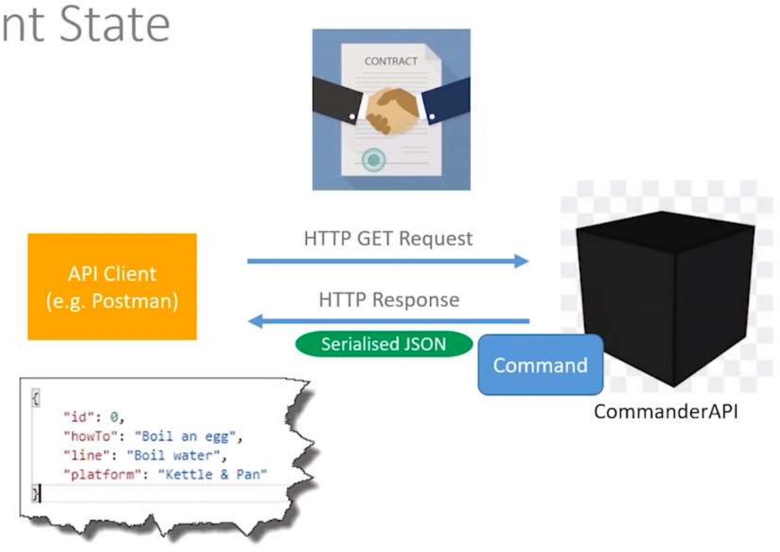
Data Transfer & Client Requests

Application Architecture



- Model: Data (Internal)
- Controller: Requests & Actions
- DB Context: “Mediator”
- Repository (Persistence Ignorance)
- Data Transfer Objects (external)

Current State



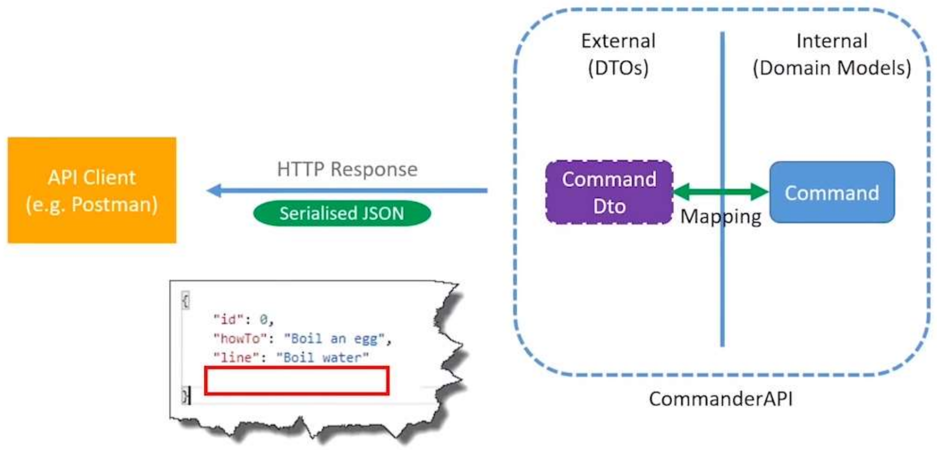
```
{
    "id": 0,
    "howTo": "Boil an egg",
    "line": "Boil water",
    "platform": "Kettle & Pan"
}
```

What’s wrong with that?

- We’re exposing internal domain implementation detail…
    - Potentially irrelevant, wrong format, insecure etc…
- We’re coupling our internals to our external contact, so:
    - Changes to our internals will be difficult in order to maintain our contract, or…
    - We’ll break our contract altogether

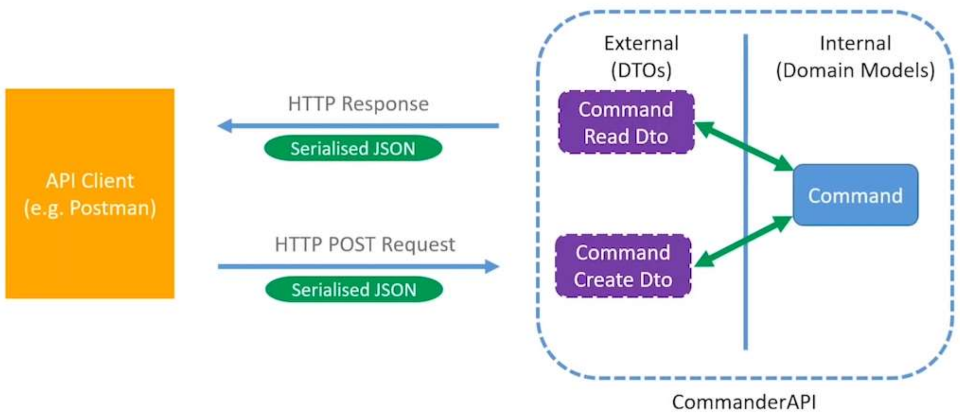
- The answer…?
    - We need to *decouple* our internal domain from what we send & receive

Data Transfer Objects



```
{
    "id": 0,
    "howTo": "Boil an egg",
    "line": "Boil water"
}
```

Data Transfer Objects

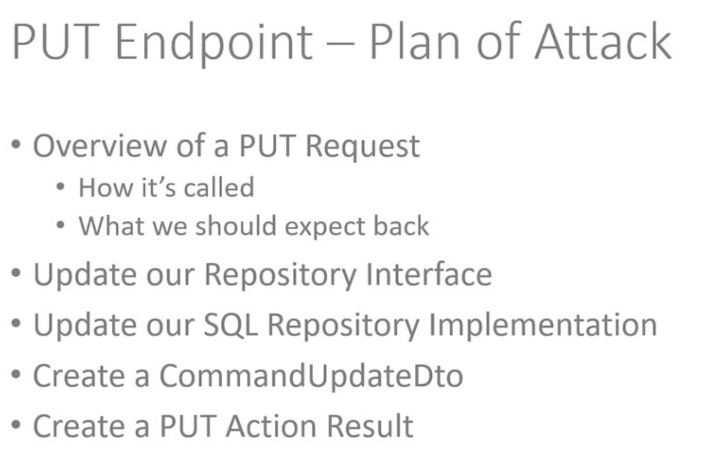


Adding additional packages from nuget.org,

Packages:

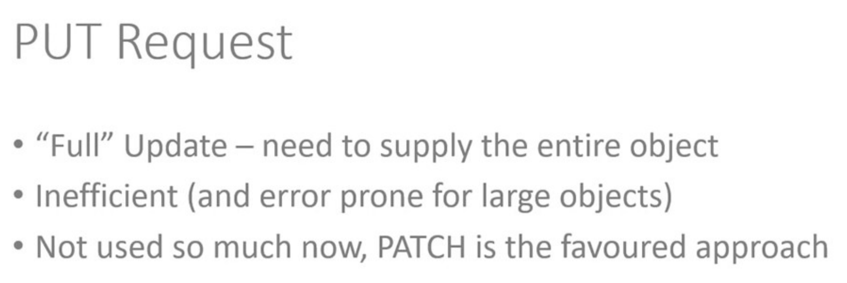
1. dotnet add package AutoMapper.Extensions.Microsoft.DependencyInjection

Endpoint Update Patch and Delete



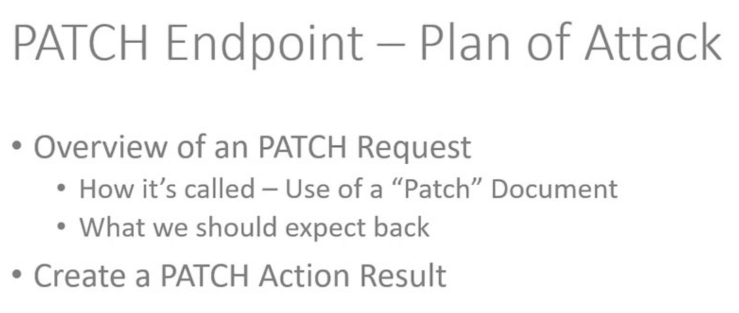
PUT Endpoint – Plan of Attack

- Overview of a PUT Request
  - How it's called
  - What we should expect back
- Update our Repository Interface
- Update our SQL Repository Implementation
- Create a CommandUpdateDto
- Create a PUT Action Result



PUT Request

- "Full" Update – need to supply the entire object
- Inefficient (and error prone for large objects)
- Not used so much now, PATCH is the favoured approach



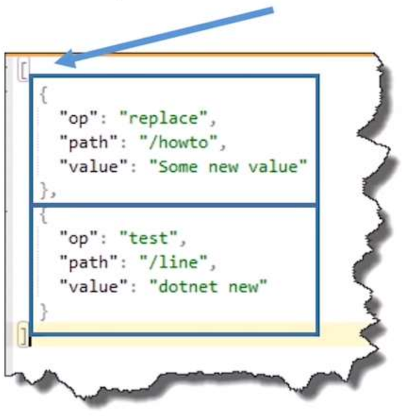
PATCH Endpoint – Plan of Attack

- Overview of an PATCH Request
  - How it's called – Use of a "Patch" Document
  - What we should expect back
- Create a PATCH Action Result

JSON Patch Standard

- JSON Patch standard specified in RFC 6902
- 6 Operations
  - Add
  - Remove
  - Replace
  - Copy
  - Move
  - Test

Example



```
[
  {
    "op": "replace",
    "path": "/howto",
    "value": "Some new value"
  },
  {
    "op": "test",
    "path": "/line",
    "value": "dotnet new"
  }
]
```

- All operations need to complete successfully

Adding additional packages from nuget.org,

Packages:

1. Microsoft.AspNetCore.JsonPatch
2. Microsoft.AspNetCore.Mvc.NewtonsoftJson

Front-end with Angular

In this section, installation of angular will be outlined as well as some options that need to be selected.

Install Node.js, this is done by going to the Node.js site and installing the latest LTS. Once done we use npm to install Angular, use npm install -g @angular/cli in the node.js cmd window. Next, created a new project, in this case CommandUI – ng new CommandUI – this is the command used.

After the angular project folder is set up, one of two things need to be done. First, find the tsconfig.app.json file and append the snippet of code below.

```
"angularCompilerOptions":
{   "enableIvy": false
  }
```

Once this is done the ng serve - -o command can be used to build and deploy it locally on the 4200 port and can be viewed in the browser.