Contents

[A - ASP.NET Core Fundamentals 2](#_Toc100080698)

[Page rout Razor 2](#_Toc100080699)

[Action result 3](#_Toc100080700)

[Select list item 3](#_Toc100080701)

[Validation 3](#_Toc100080702)

[Redirect to page with route value 5](#_Toc100080703)

[Entity framework 5](#_Toc100080704)

[Interface 9](#_Toc100080705)

[\_ViewStar 10](#_Toc100080706)

[\_viewImport 10](#_Toc100080707)

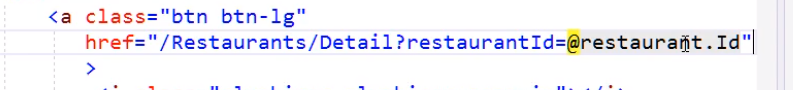
[View component 11](#_Toc100080708)

[ASI ASP.NET CORE 12](#_Toc100080709)

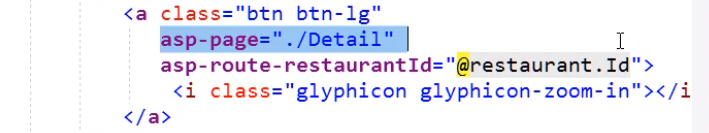
# A - ASP.NET Core Fundamentals

## Page rout Razor

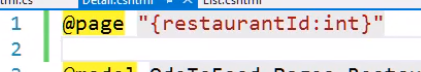
* Query string



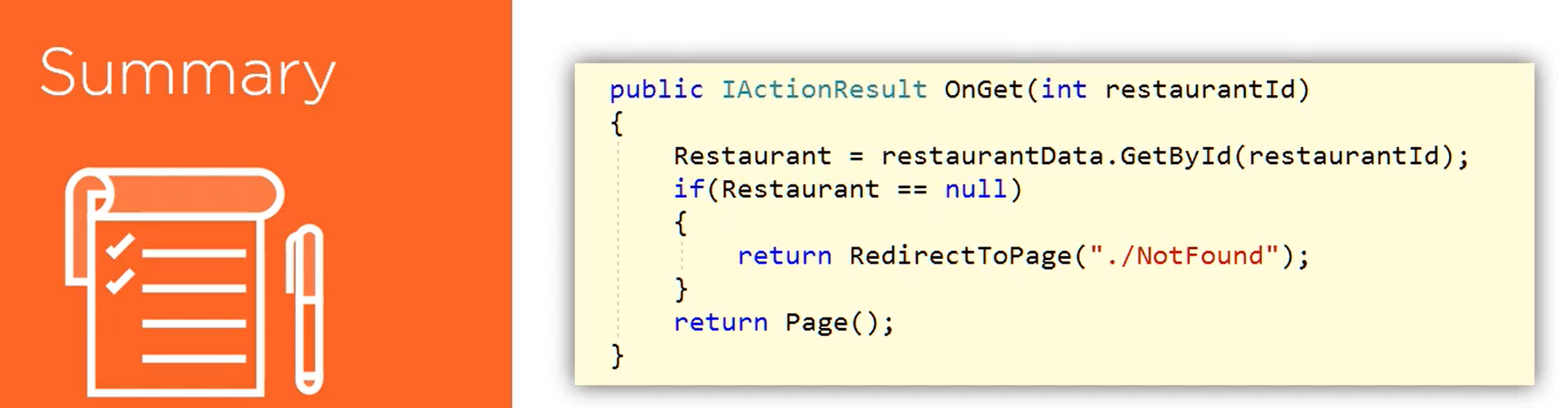
* Asp tag helper



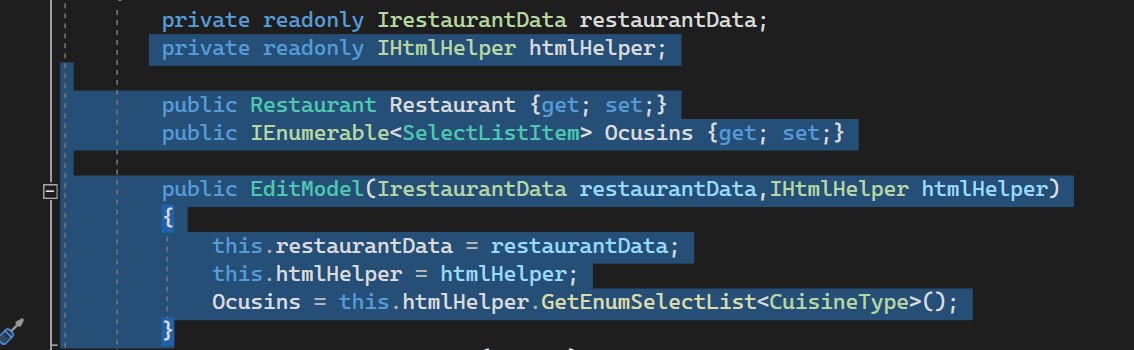
* Page Route



## Action result

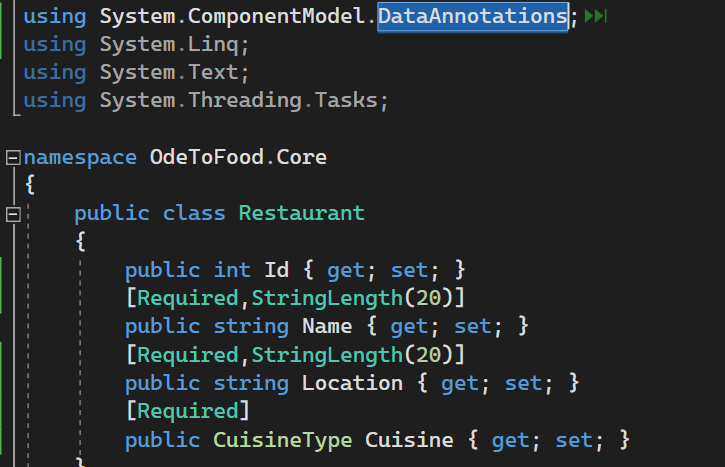


## Select list item

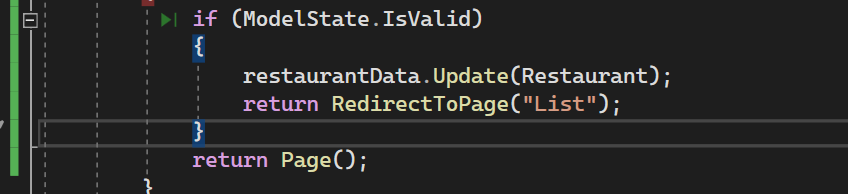


## Validation

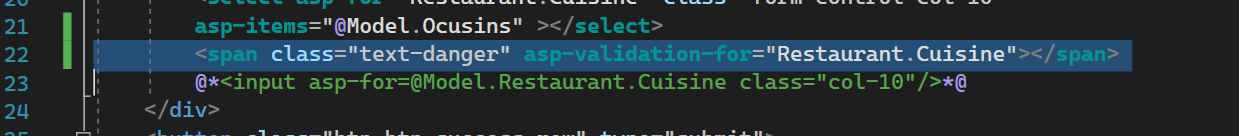
Data annotation



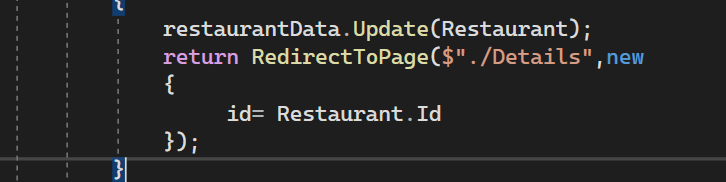
Modelstate



Asp Validation for



## Redirect to page with route value



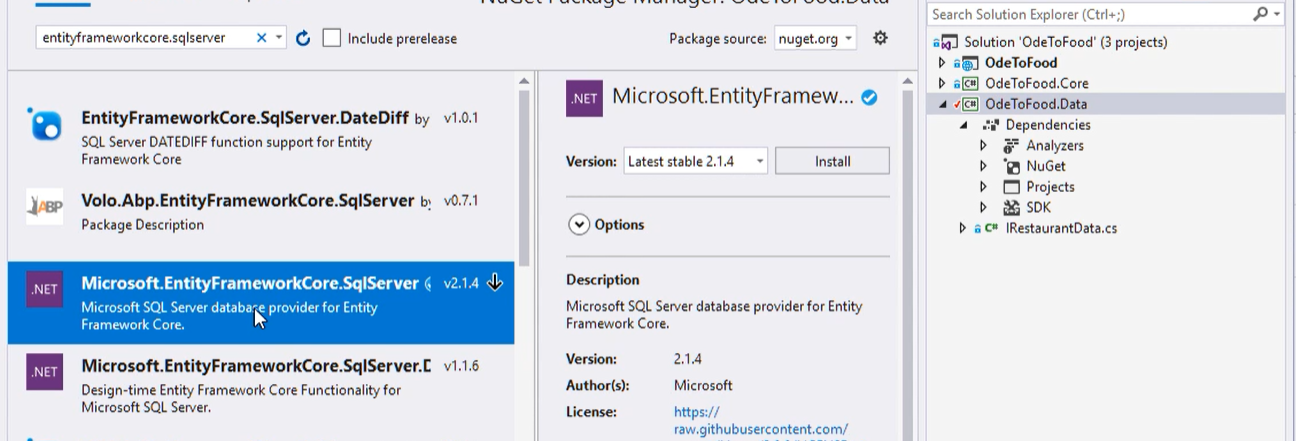
## Entity framework

Install package on project .data

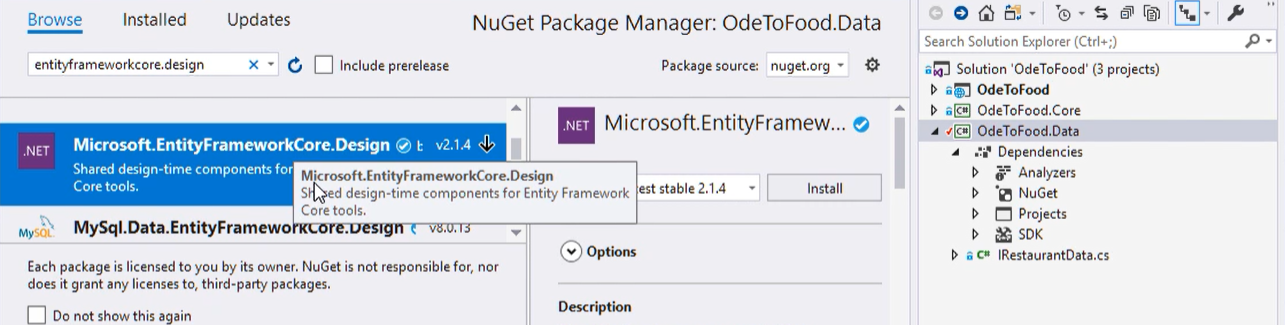
1. Microsoft.EntityFrameworkCore



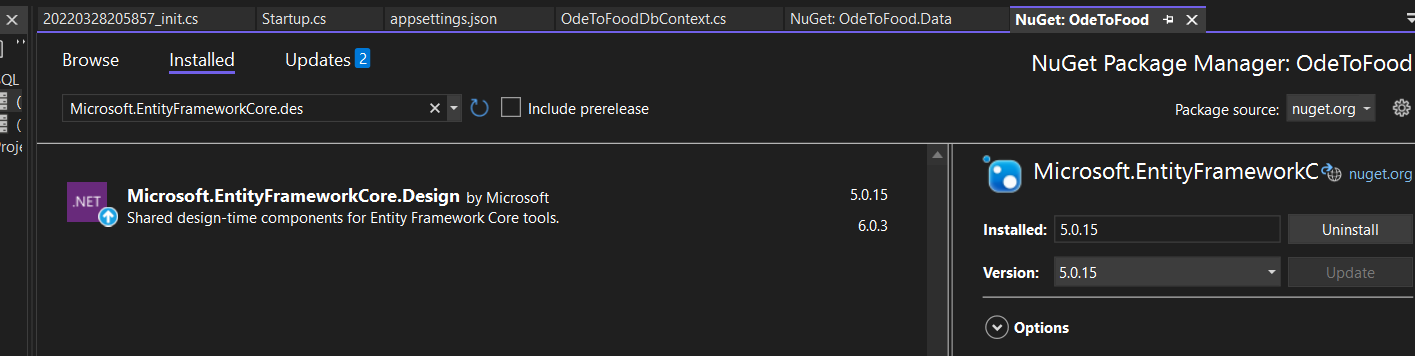
1. Microsoft.EntityFrameworkCore.Sqlserver



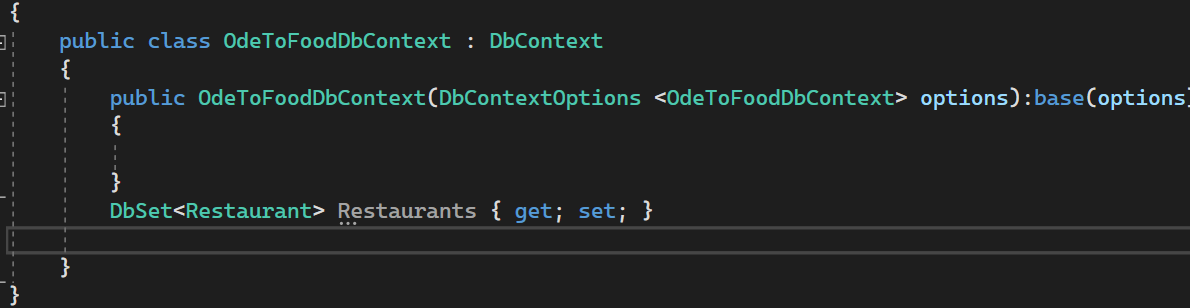
1. Microsoft.EntityFrameworkCore.Design



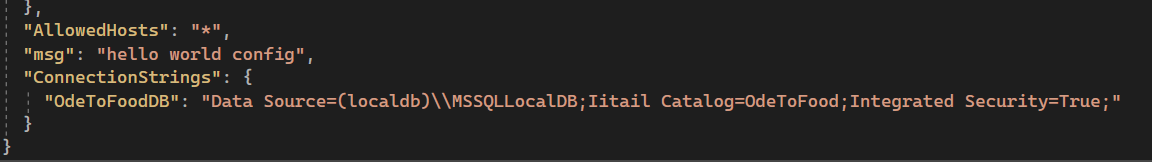
Add package design to startup project



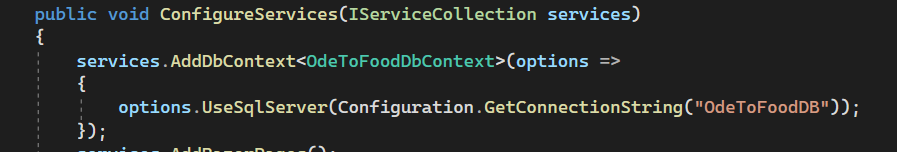
Create dbcontext class



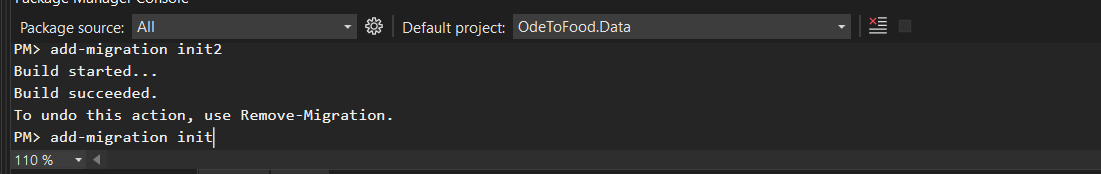
Add connection string on appsetting.json



Add configure services



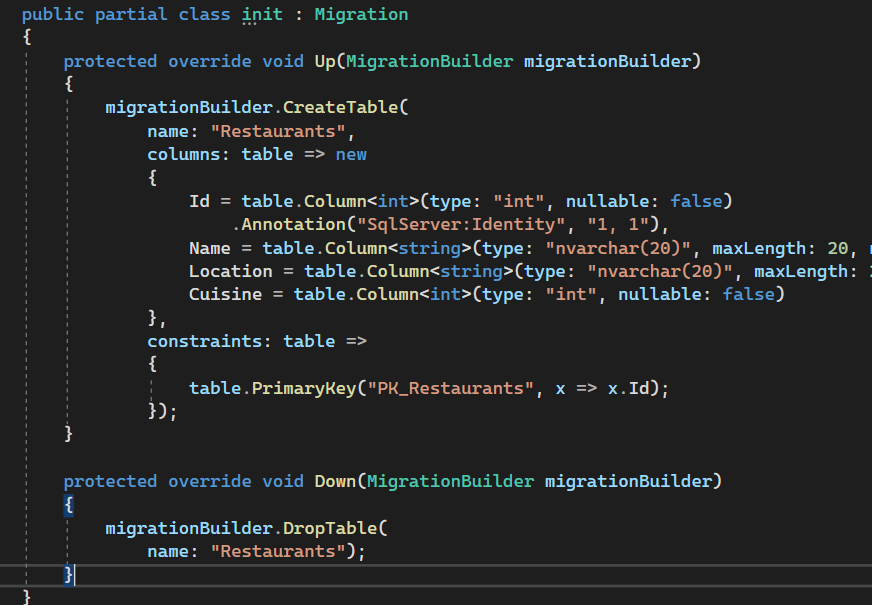
Open package manger console on odetofood.data

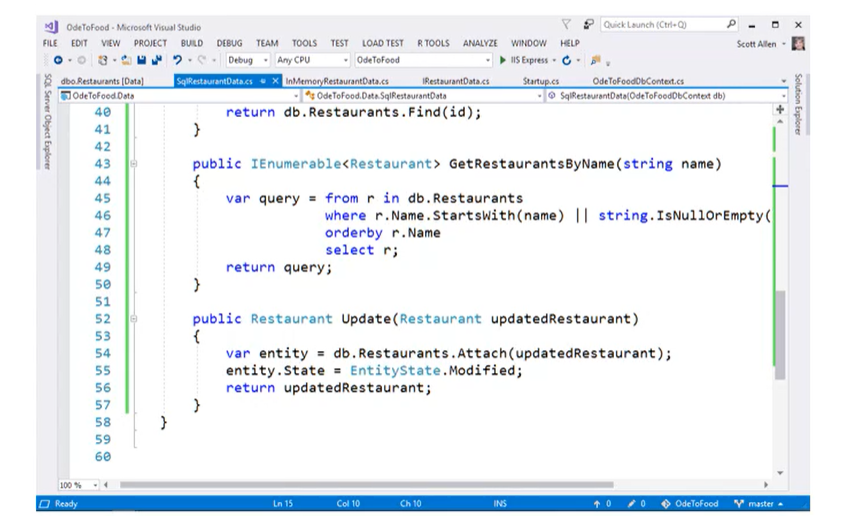


add-migration init

update-database

the migration



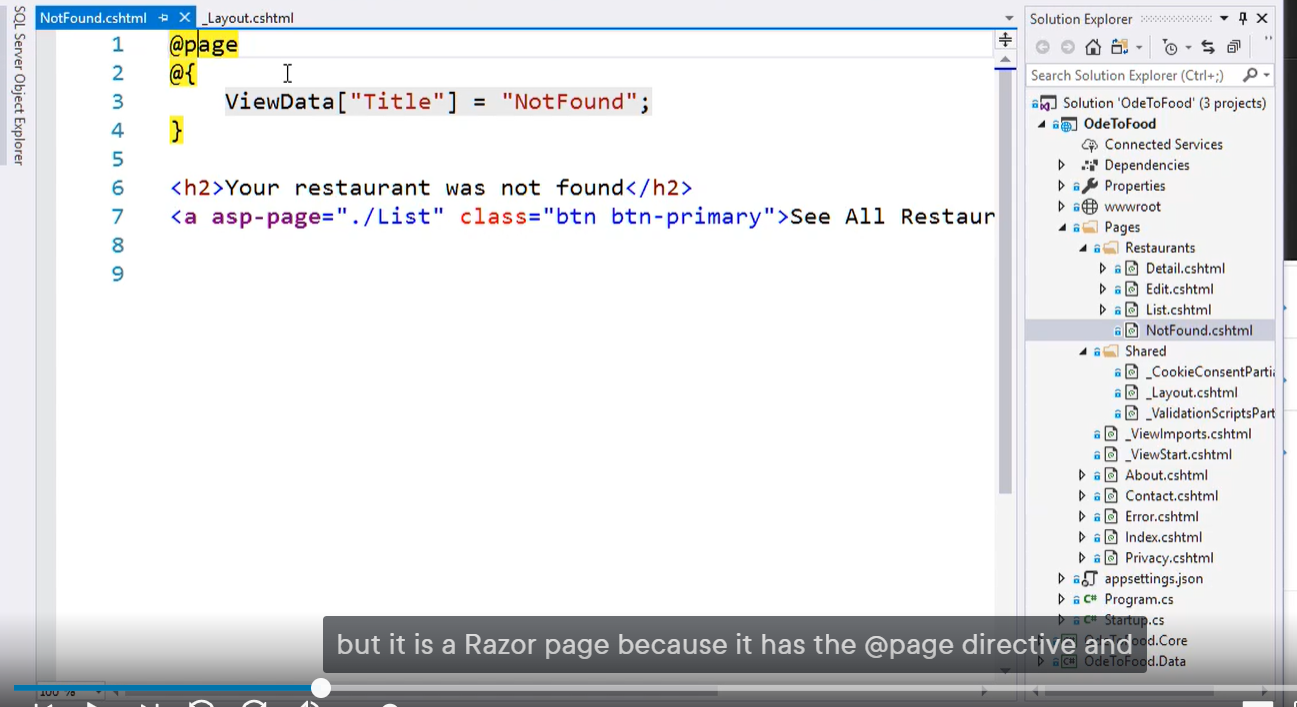


## Interface

@page directive

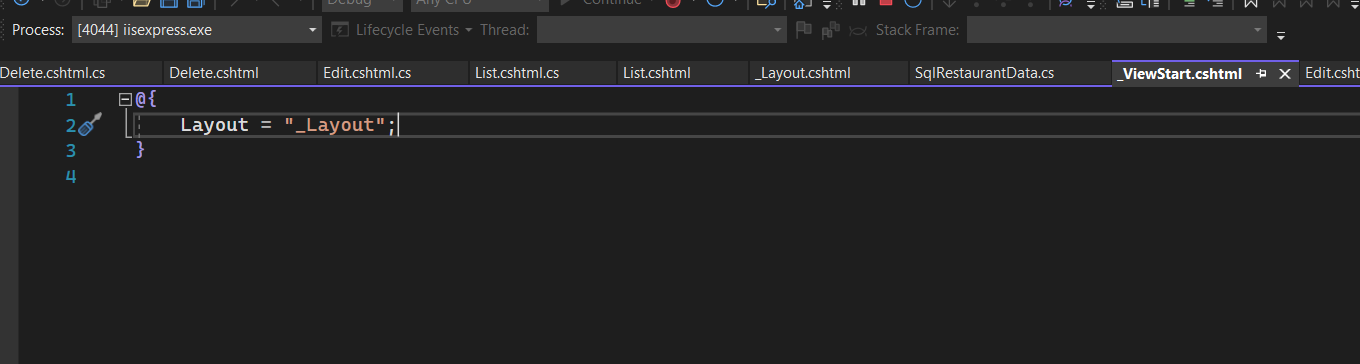
To make page a razor page

It makes the page response to url



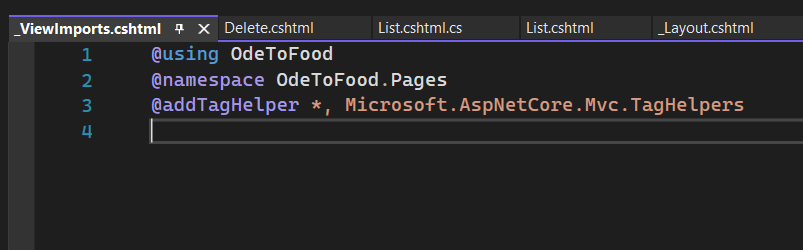
## \_ViewStar

The page that execute before any razor page

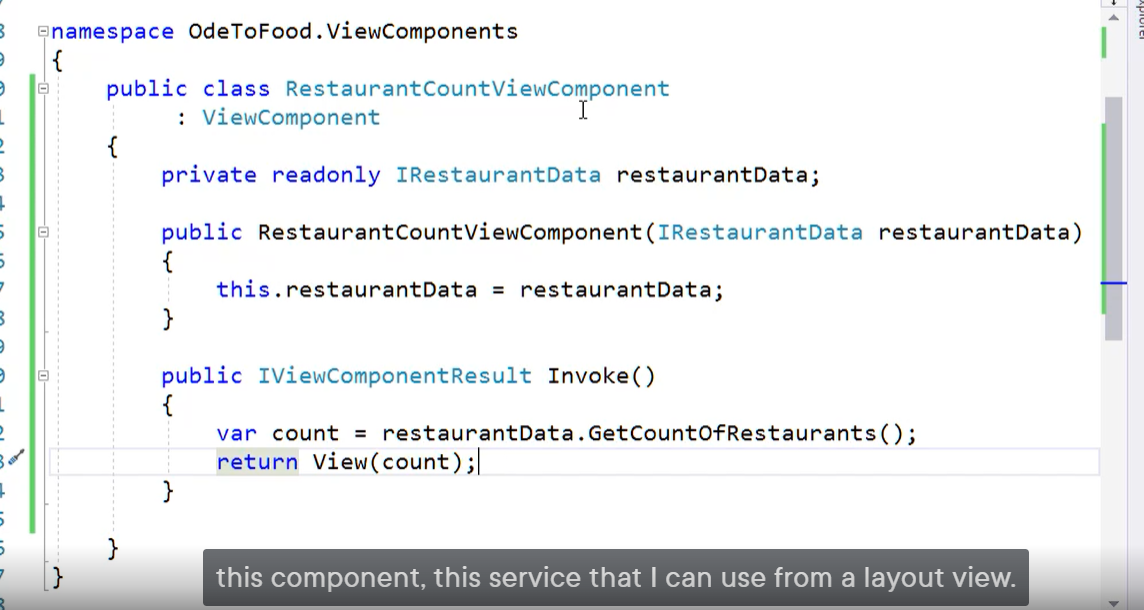


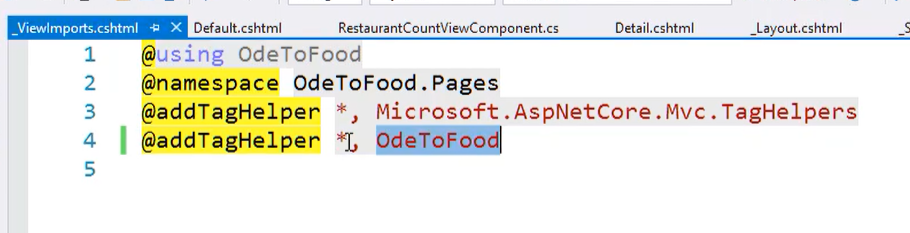
## \_viewImport

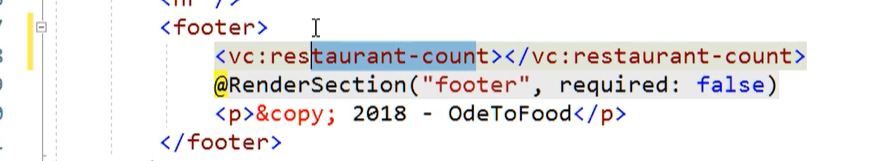
To using namespaces



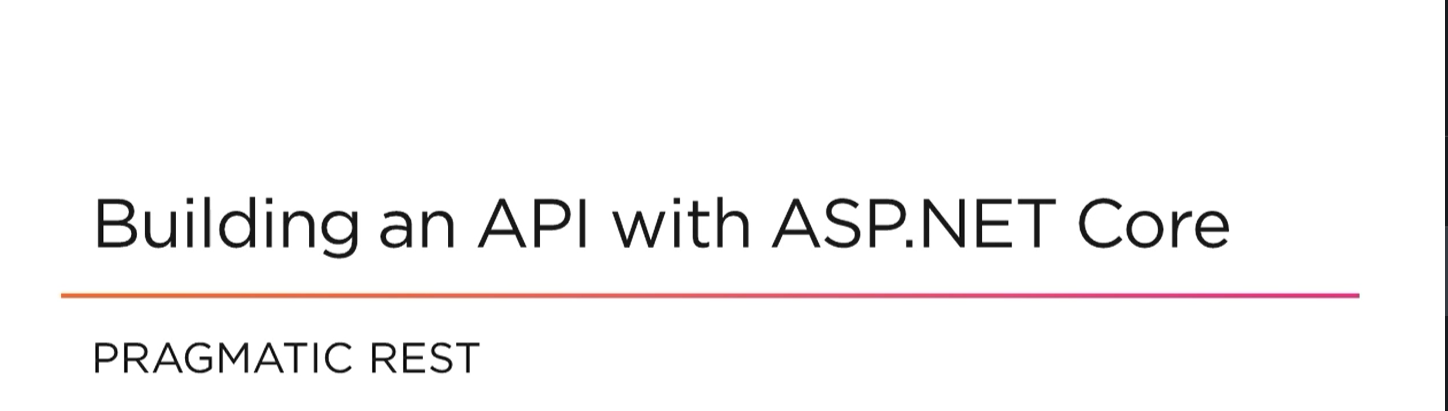
## View component







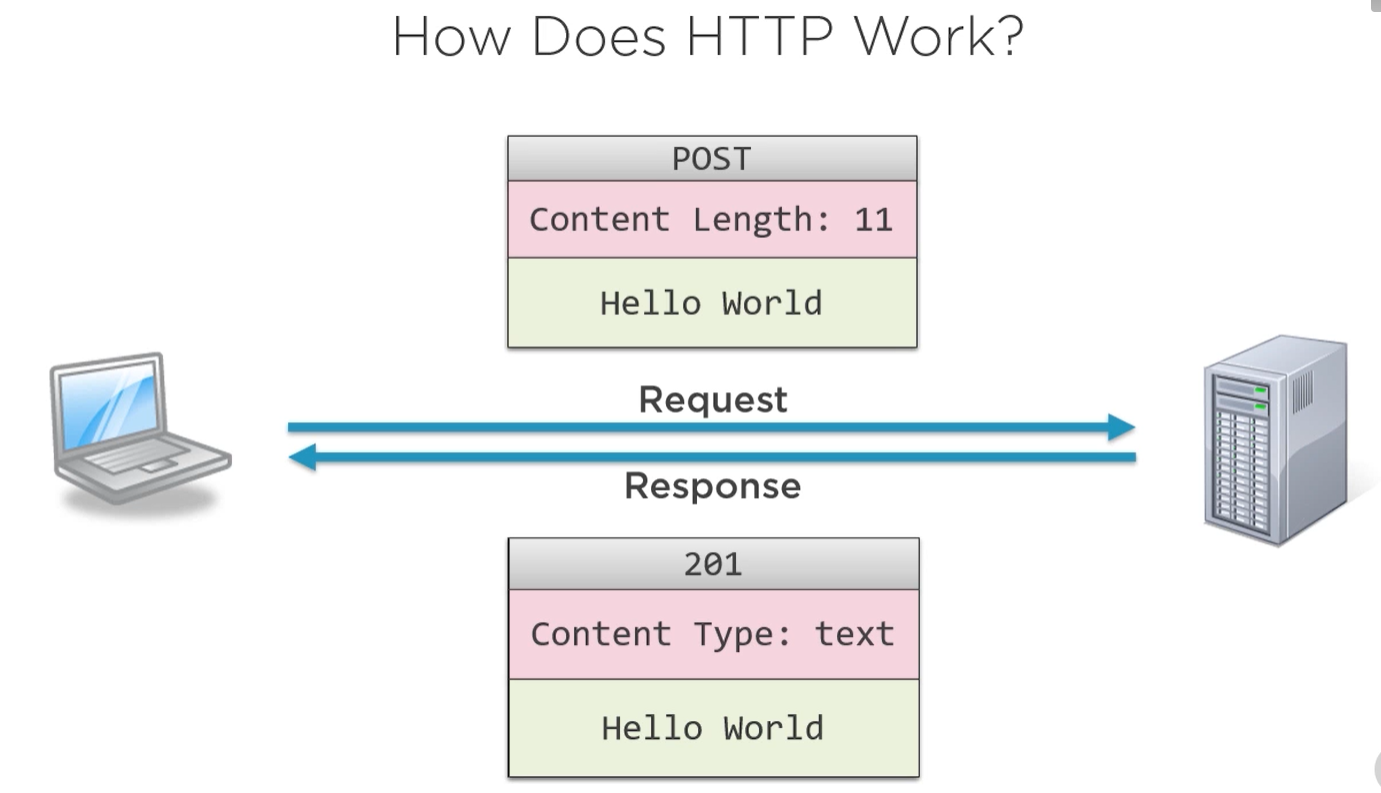
# ASI ASP.NET CORE



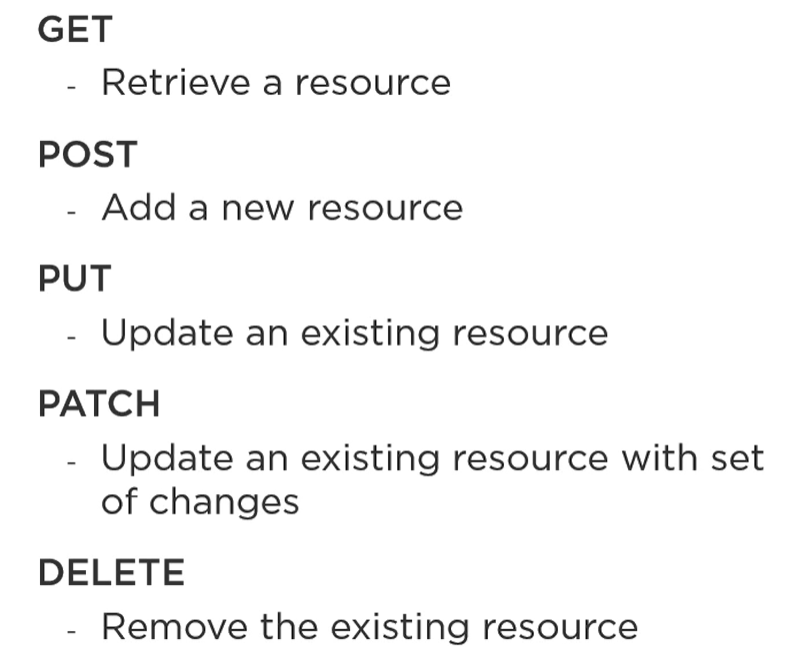
## Pragmatic



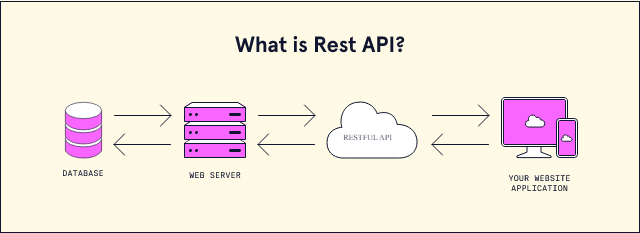
## http



Verbs



## REST



<https://www.codecademy.com/article/what-is-rest>

REST, or REpresentational State Transfer, is an architectural style for providing standards between computer systems on the web, making it easier for systems to communicate with each other. REST-compliant systems, often called RESTful systems, are characterized by how they are stateless and separate the concerns of client and server

### **Separation of Client and Server**

In the REST architectural style, the implementation of the client and the implementation of the server can be done independently without each knowing about the other. This means that the code on the client side can be changed at any time without affecting the operation of the server,

As long as each side knows what format of messages to send to the other, they can be kept modular and separate

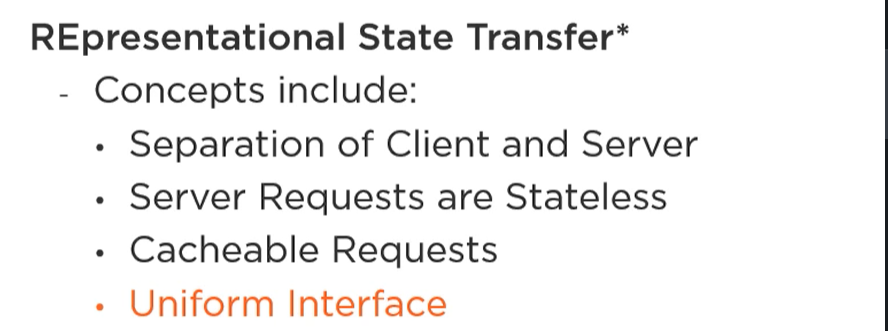
<https://www.redhat.com/en/topics/api/what-is-a-rest-api#:~:text=A%20REST%20API%20(also%20known,by%20computer%20scientist%20Roy%20Fielding>.

REST is a set of architectural constraints, not a protocol or a standard. API developers can implement REST in a variety of ways.

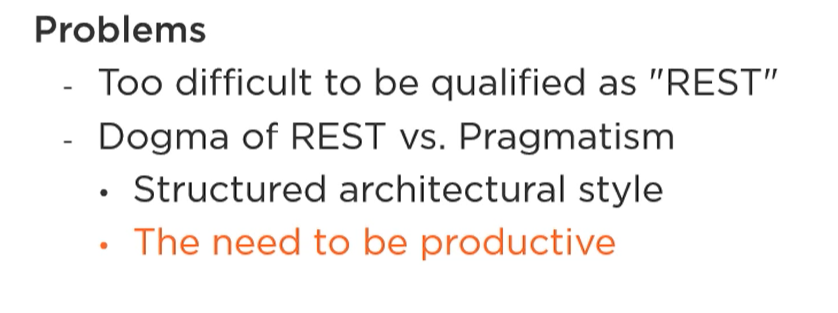
When a client request is made via a RESTful API, it transfers a representation of the state of the resource to the requester or endpoint. This information, or representation, is delivered in one of several formats via HTTP: JSON (Javascript Object Notation), HTML, XLT, Python, PHP, or plain text. JSON is the most generally popular file format to use because, despite its name, it’s language-agnostic, as well as readable by both humans and machines

**In order for an API to be considered RESTful, it has to conform to these criteria**:

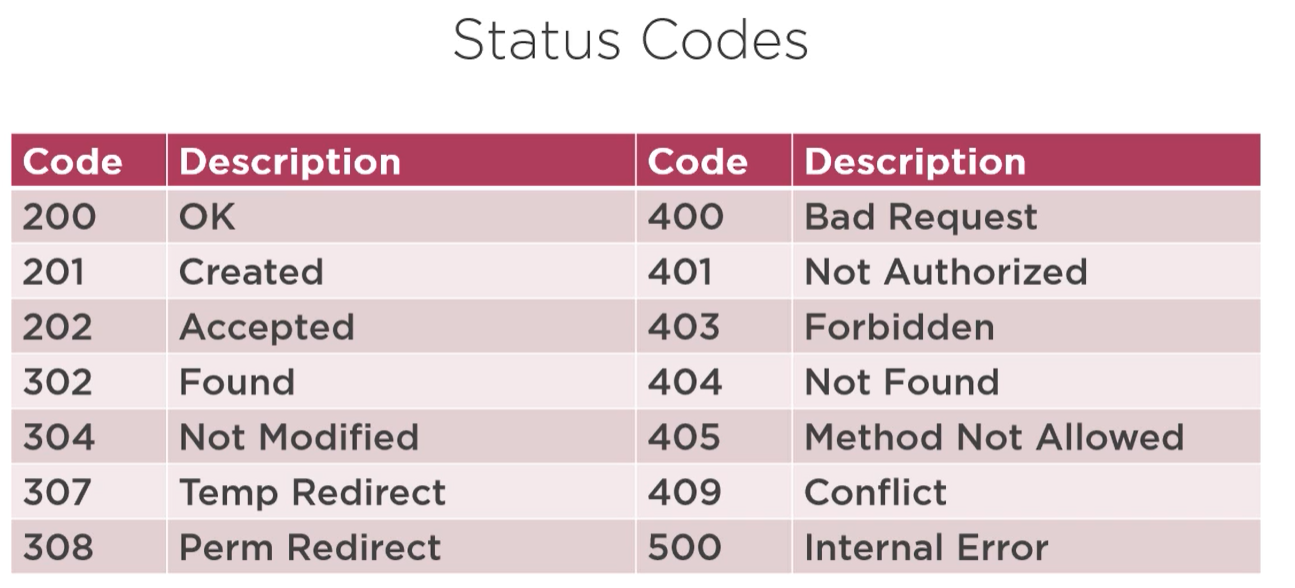
* A client-server architecture made up of clients, servers, and resources, with requests managed through HTTP.
* [Stateless](https://www.redhat.com/en/topics/cloud-native-apps/stateful-vs-stateless) client-server communication, meaning no client information is stored between get requests and each request is separate and unconnected.
* Cacheable data that streamlines client-server interactions.
* A uniform interface between components so that information is transferred in a standard form. This requires that:
  + resources requested are identifiable and separate from the representations sent to the client.
  + resources can be manipulated by the client via the representation they receive because the representation contains enough information to do so.
  + self-descriptive messages returned to the client have enough information to describe how the client should process it.
  + hypertext/hypermedia is available, meaning that after accessing a resource the client should be able to use hyperlinks to find all other currently available actions they can take.
* A layered system that organizes each type of server (those responsible for security, load-balancing, etc.) involved the retrieval of requested information into hierarchies, invisible to the client.
* Code-on-demand (optional): the ability to send executable code from the server to the client when requested, extending client functionality.



REST problem

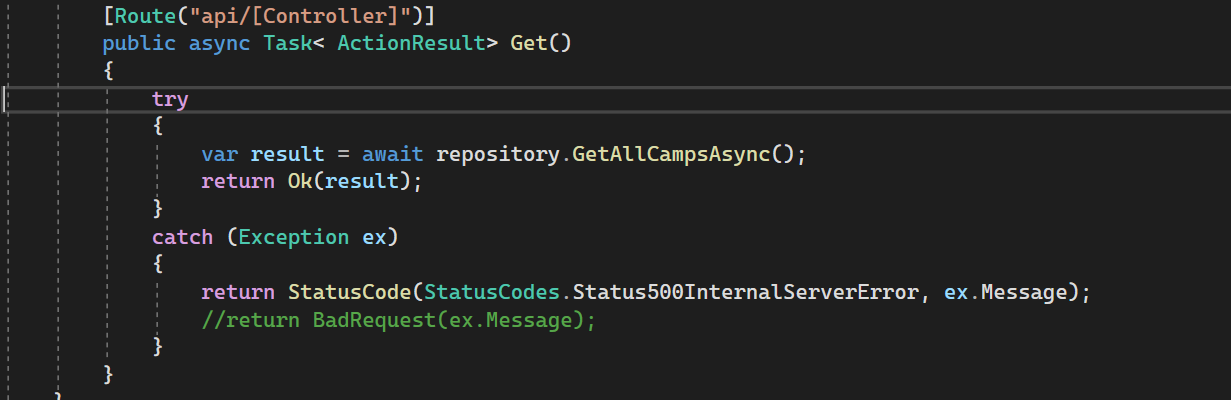


## Status codes

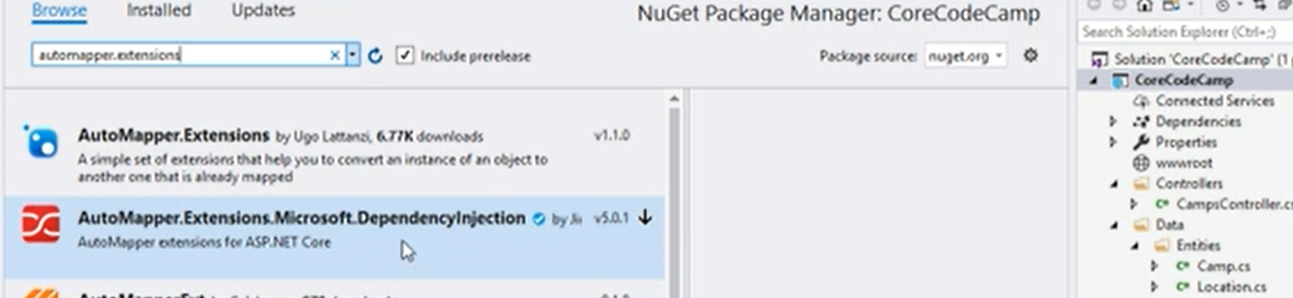








## Automapper



<https://nwb.one/blog/set-up-automapper-dotnet-core>