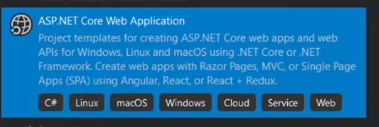
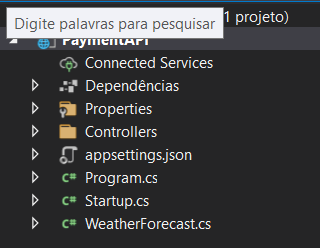


Em Inglês

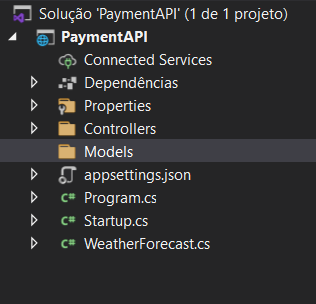


Salve em uma pasta: AngularCrudProjetoFinal

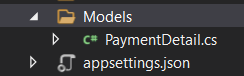
Nome do Projeto: **PaymentAPI**



Criar pasta **Models**



Dentro dela uma classe chamada: **PaymentDetail.cs**



Conteúdo de PaymentDetail.cs

Obs: Inserir os usings…

using System.ComponentModel.DataAnnotations;

using System.ComponentModel.DataAnnotations.Schema;

|  |
| --- |
| public class PaymentDetail  {  [Key]  public int PaymentDetailId { get; set; }  [Required]  [Column(TypeName = "nvarchar(100)")]  public string CardOwnerName { get; set; }  [Required]  [Column(TypeName = "varchar(16)")]  public string CardNumber { get; set; }  [Required]  [Column(TypeName = "varchar(5)")]  public string ExpirationDate { get; set; }  [Required]  [Column(TypeName = "varchar(3)")]  public string SecurityCode { get; set; }  } |

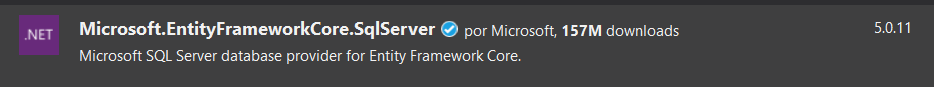
Agora vamos definir a  DbContext classe

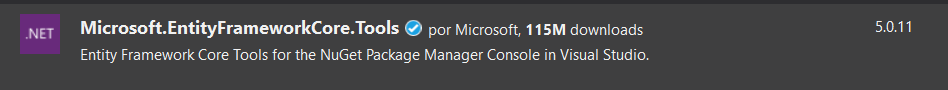
Criar o Arquivo  /Models/PaymentDetailContext.cs

**Banco de dados de configuração**

Vamos criar um banco de dados para este projeto. Dentro deste projeto, usaremos o Entity Framework Core para criar e interagir com o banco de dados. Portanto, em primeiro lugar, devemos instalar os pacotes NuGet correspondentes. Clique com o botão direito do mouse no nome do projeto no Solution Explorer, clique em Manage NuGet Packages, na guia Browse, instale os 3 pacotes a seguir com a mesma versão do Asp.Net Core.







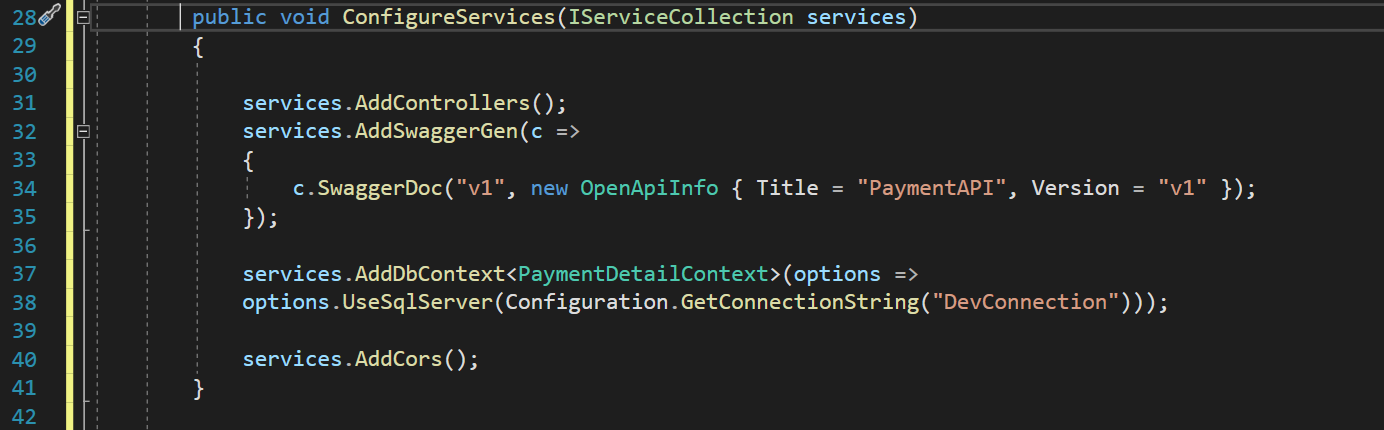
Classe DbContext -  PaymentDetailContext decide o que deve ser adicionado ao banco de dados físico real durante a migração do banco de dados. Portanto, adicionamos  DbSet propriedade para a  PaymentDetail classe Model, após a PaymentDetailstabela de migração será criada no Banco de Dados SQL Server.

|  |
| --- |
| using Microsoft.EntityFrameworkCore;  using System;  using System.Collections.Generic;  using System.Linq;  using System.Threading.Tasks;  namespace PaymentAPI.Models  {  public class PaymentDetailContext : DbContext  {  public PaymentDetailContext(DbContextOptions<PaymentDetailContext> options) : base(options)  { }  public DbSet<PaymentDetail> PaymentDetails { get; set; }  }  } |

Neste parâmetro do construtor de classe de modelo  options , temos que passar qual DbProvider (SQL Server, MySQL, PostgreSQL, etc) para usar e a string de conexão de banco de dados correspondente também. Para isso, usaremos injeção de dependência no ASP.NET Core com o Startup.cs

arquivo a seguir.

Acrescente a linha 37 inteira até a linha 31



Efetuar o import do Using model

Obs.: No caso de não reconhecimento de alguma dependência(UseSqlServer, por exemplo) soilução: desisntalar o package pelo NuGet.. e reinstalar

Aqui, usamos injeção de dependência para  DbContext classe, por meio da qual o SQL Server é definido como um DbProvider com uma string de conexão. Agora salve a string de conexão no appsettings.json arquivo usando a  DevConnection chave da seguinte maneira.

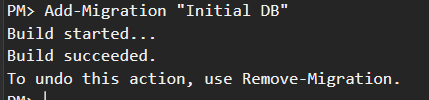
appsettings.json

|  |
| --- |
| {  "Logging": {  "LogLevel": {  "Default": "Information",  "Microsoft": "Warning",  "Microsoft.Hosting.Lifetime": "Information"  }  },  "AllowedHosts": "\*",  "ConnectionStrings": {  "DevConnection": "Server=DESK\\SQLEXPRESS;Database=PaymentDetailDB;Trusted\_Connection=True;MultipleActiveResultSets=True;"  }  }  "AllowedHosts": "\*"  } |

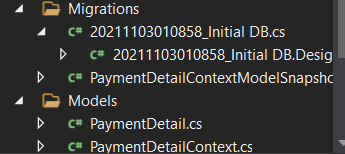
Agora vamos fazer a migração. Selecione o projeto no explorador de soluções e vá para  **Ferramentas> Gerenciador de pacotes NuGet> Console do gerenciador de pacotes** . Em seguida, execute os seguintes comandos um por um.

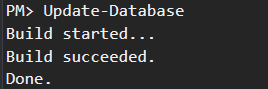
***Faça um Build do projeto antes do migrate***

Após isto



Criada a migration



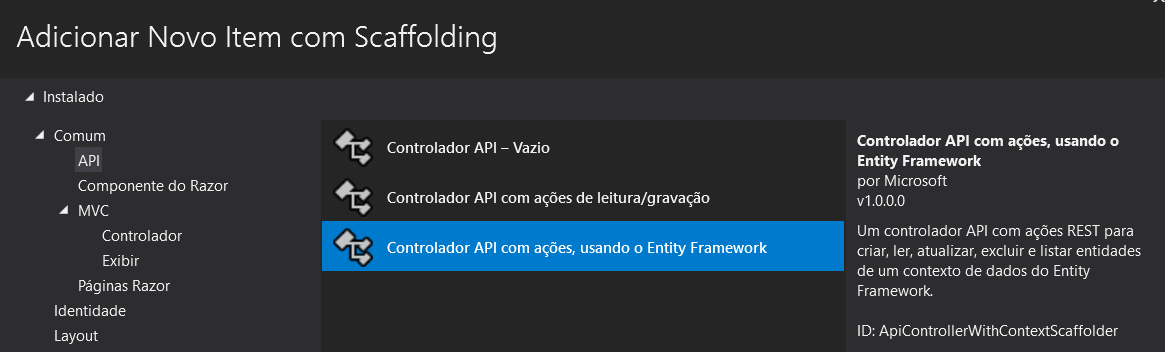


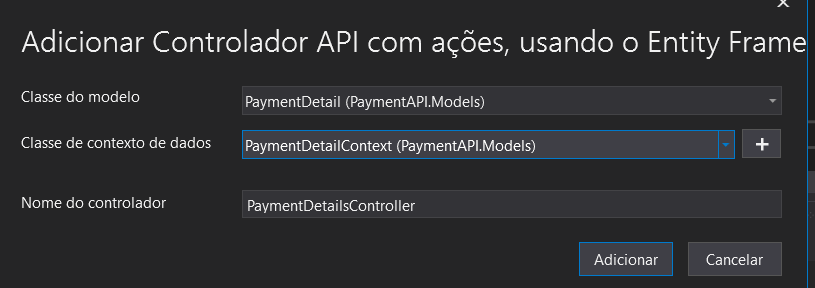
Após a migração bem-sucedida, de acordo com a string de conexão, um novo banco de dados – PaymentDetailDB será criado com a tabela  PaymentDetails. Além disso, haverá uma nova Migrations pasta criada com os arquivos C # correspondentes.

Deverá existir o banco de dados no sqlserver

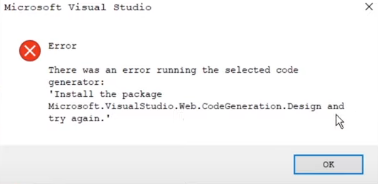
## Criar controlador de API para operações CRUD

Para criar um novo controlador de API, clique com o botão direito na pasta Controllers  **Adicionar> Controlador** , Selecionar API Controller with actions, using Entity Framework.





Pode vir a dar um erro basta instalar o pacote sugerido

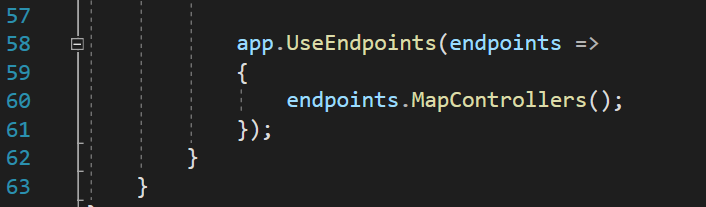


Com a ajuda do scaffold teremos este resultado

PaymentDetailController.cs

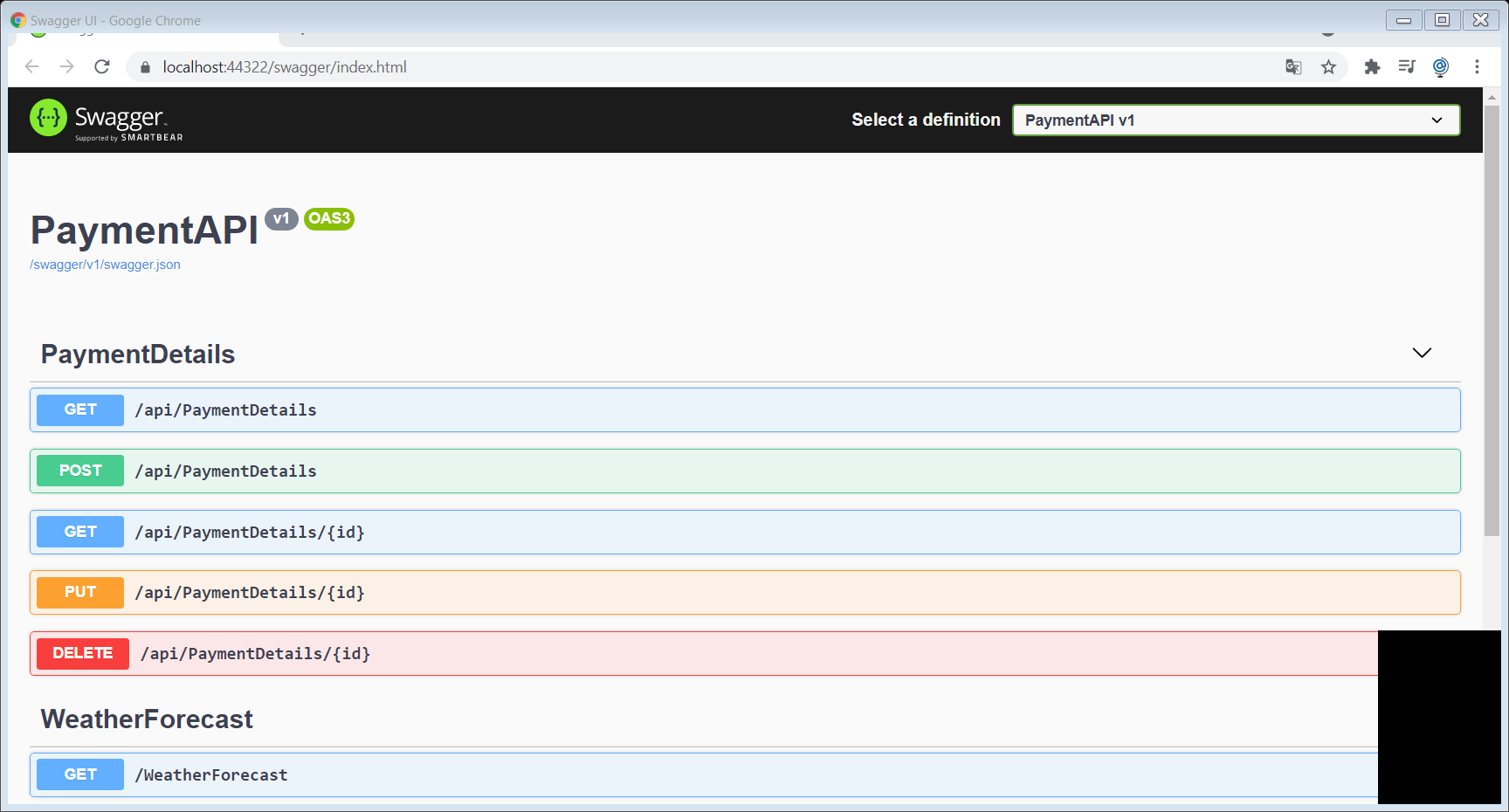
|  |
| --- |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Threading.Tasks;  using Microsoft.AspNetCore.Http;  using Microsoft.AspNetCore.Mvc;  using Microsoft.EntityFrameworkCore;  using PaymentAPI.Models;  namespace PaymentAPI.Controllers  {  [Route("api/[controller]")]  [ApiController]  public class PaymentDetailsController : ControllerBase  {  private readonly PaymentDetailContext \_context;  public PaymentDetailsController(PaymentDetailContext context)  {  \_context = context;  }  // GET: api/PaymentDetails  [HttpGet]  public async Task<ActionResult<IEnumerable<PaymentDetail>>> GetPaymentDetails()  {  return await \_context.PaymentDetails.ToListAsync();  }  // GET: api/PaymentDetails/5  [HttpGet("{id}")]  public async Task<ActionResult<PaymentDetail>> GetPaymentDetail(int id)  {  var paymentDetail = await \_context.PaymentDetails.FindAsync(id);  if (paymentDetail == null)  {  return NotFound();  }  return paymentDetail;  }  // PUT: api/PaymentDetails/5  // To protect from overposting attacks, see https://go.microsoft.com/fwlink/?linkid=2123754  [HttpPut("{id}")]  public async Task<IActionResult> PutPaymentDetail(int id, PaymentDetail paymentDetail)  {  if (id != paymentDetail.PaymentDetailId)  {  return BadRequest();  }  \_context.Entry(paymentDetail).State = EntityState.Modified;  try  {  await \_context.SaveChangesAsync();  }  catch (DbUpdateConcurrencyException)  {  if (!PaymentDetailExists(id))  {  return NotFound();  }  else  {  throw;  }  }  return NoContent();  }  // POST: api/PaymentDetails  // To protect from overposting attacks, see https://go.microsoft.com/fwlink/?linkid=2123754  [HttpPost]  public async Task<ActionResult<PaymentDetail>> PostPaymentDetail(PaymentDetail paymentDetail)  {  \_context.PaymentDetails.Add(paymentDetail);  await \_context.SaveChangesAsync();  return CreatedAtAction("GetPaymentDetail", new { id = paymentDetail.PaymentDetailId }, paymentDetail);  }  // DELETE: api/PaymentDetails/5  [HttpDelete("{id}")]  public async Task<IActionResult> DeletePaymentDetail(int id)  {  var paymentDetail = await \_context.PaymentDetails.FindAsync(id);  if (paymentDetail == null)  {  return NotFound();  }  \_context.PaymentDetails.Remove(paymentDetail);  await \_context.SaveChangesAsync();  return NoContent();  }  private bool PaymentDetailExists(int id)  {  return \_context.PaymentDetails.Any(e => e.PaymentDetailId == id);  }  }  } |

Ele contém os métodos da web POST, GET, PUT e DELETE para as operações de Criar, Recuperar, Atualizar e Excluir, respectivamente. Como um parâmetro do construtor, temos  context o tipo  PaymentDetailContext . a instância / valor para este parâmetro será passado da injeção de dependência da  StartUp classe.



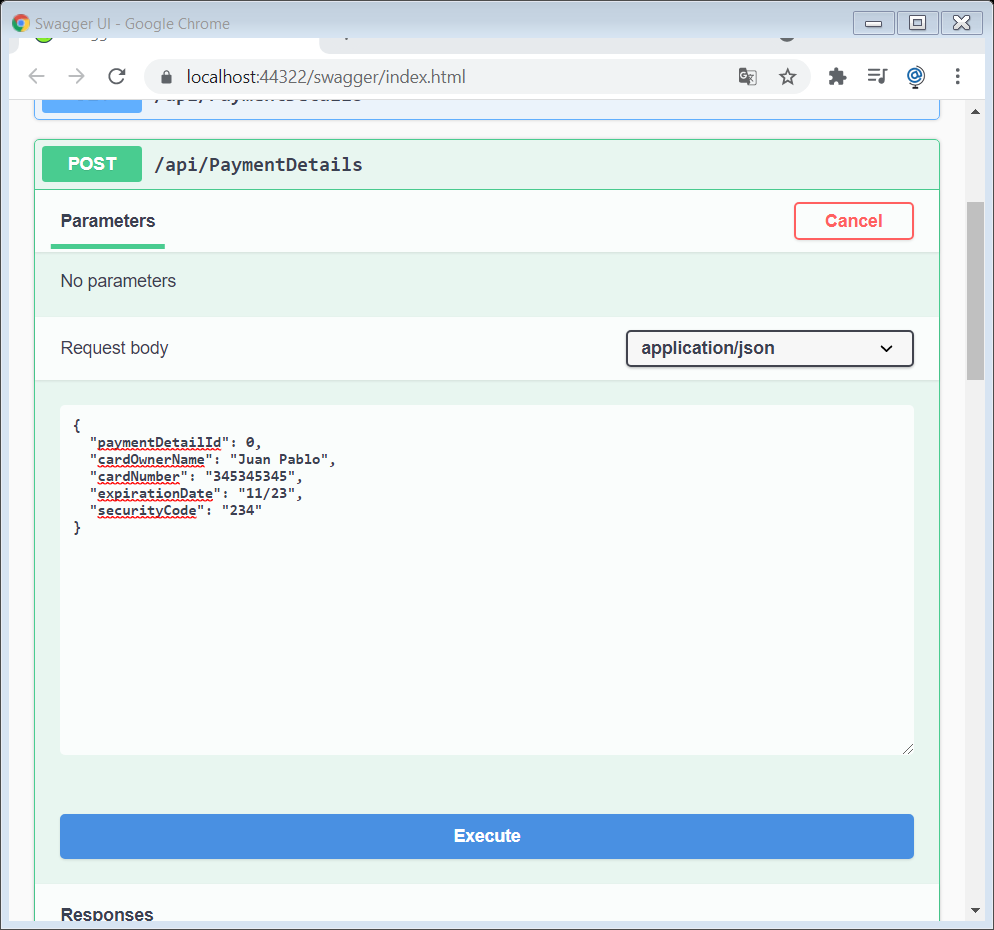
Para este projeto, não precisamos mudar nada nos métodos da web e você pode testar qualquer uma dessas operações CRUD usando um software como o  [postman](https://www.postman.com/) ou usar o suporte a API aberta com a interface swagger.

Vamos ver no Swagger pressione F5

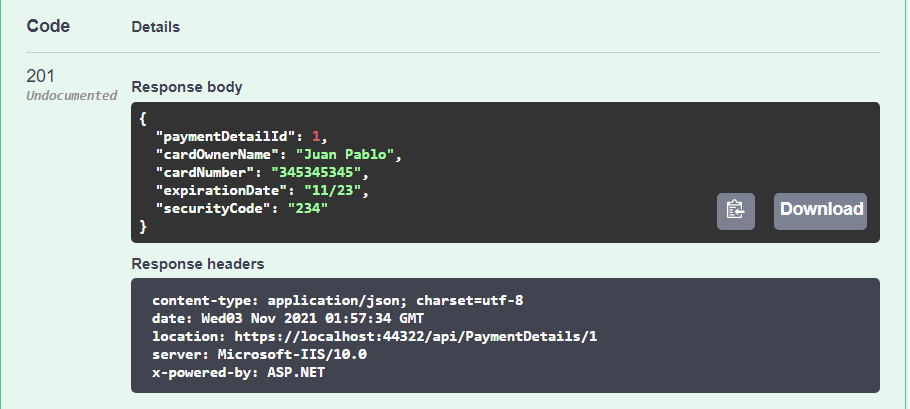


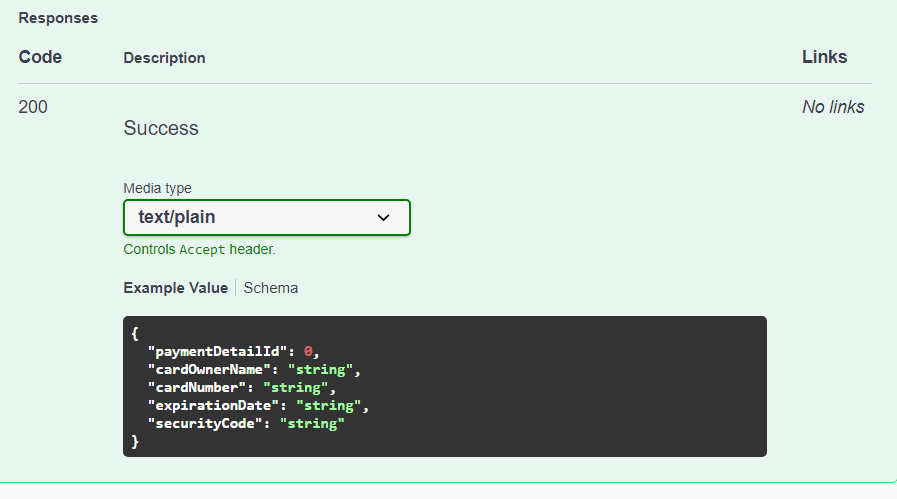
Lembra dele

Vamos testar o post que é para inserir

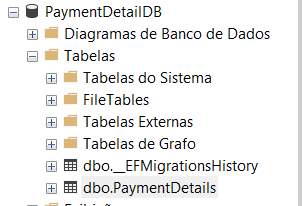


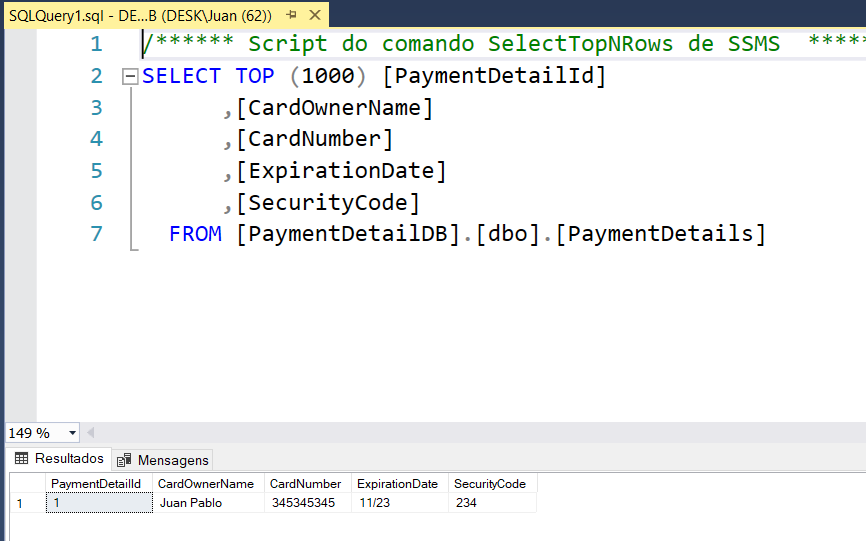
Clique em execute



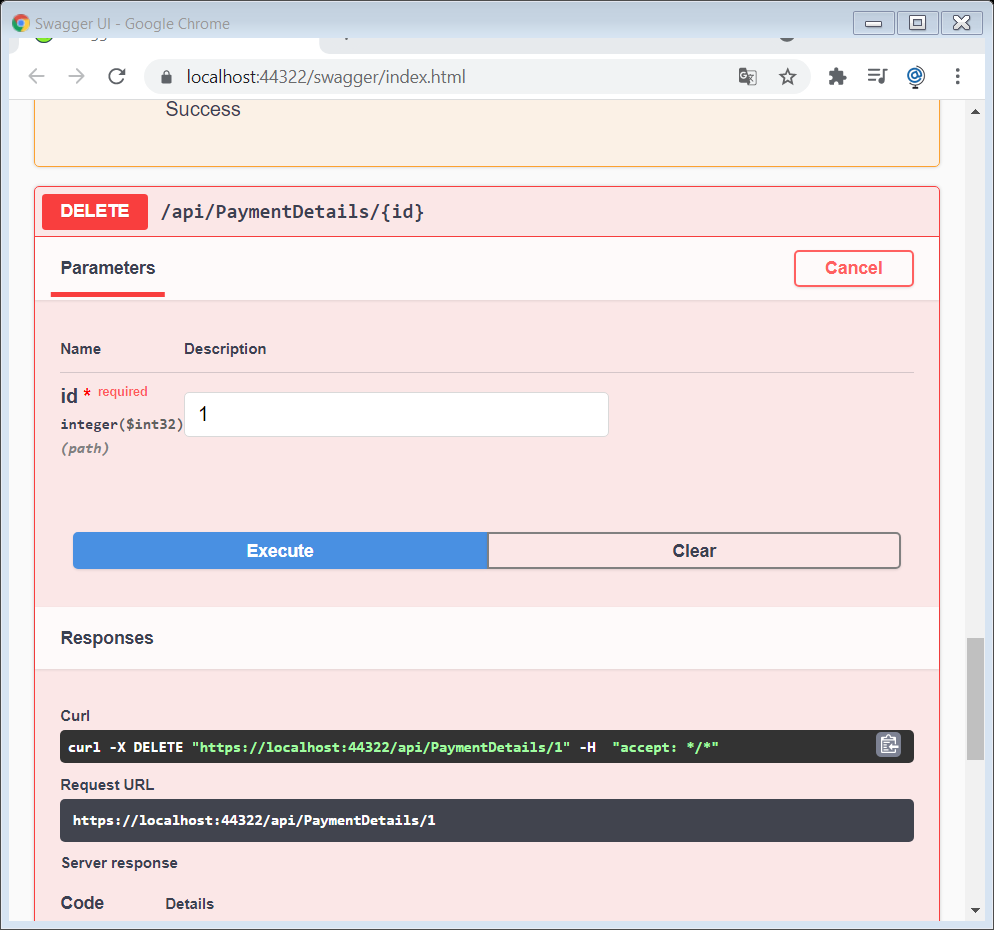


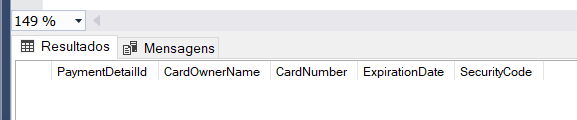
**Abra o SqlServer**





Delete





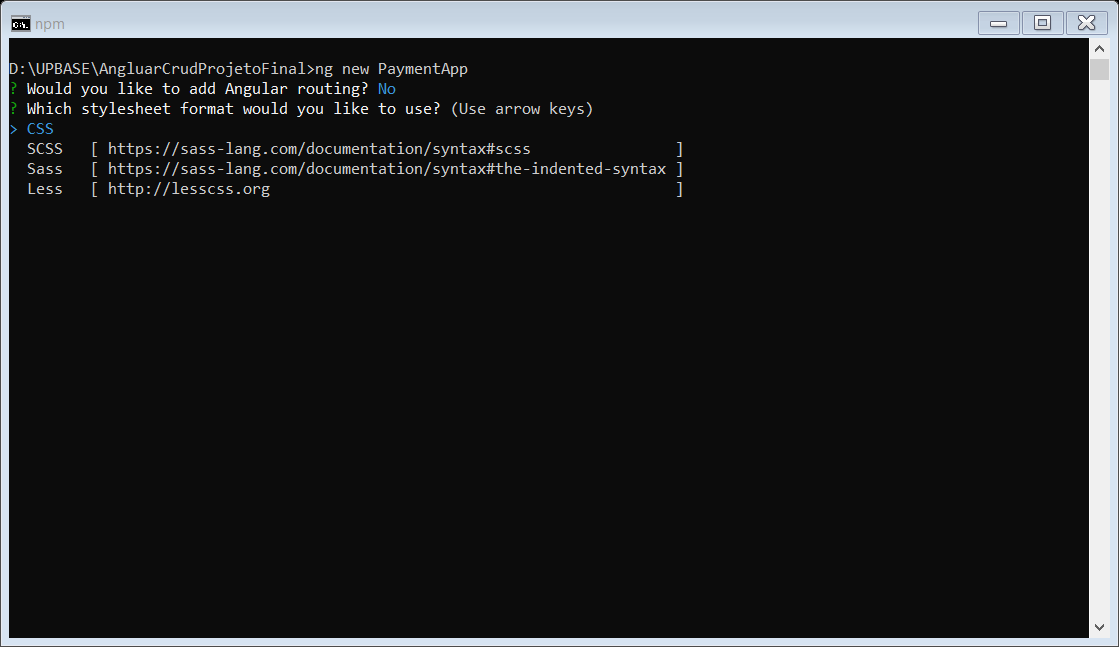
## Criar aplicativo angular

Agora vamos criar um aplicativo front-end do lado do cliente no Angular 12. Para isso, execute os seguintes comandos Angular-CLI um por um.

Criar routing 🡪 N

**CSS**

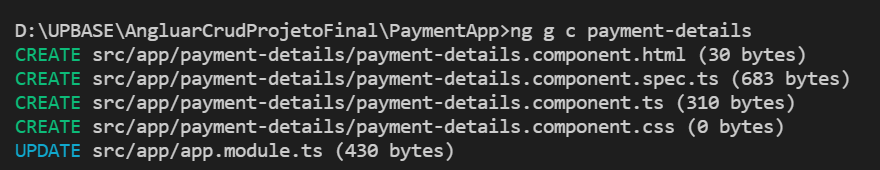
Deixar instalar o projeto Angular



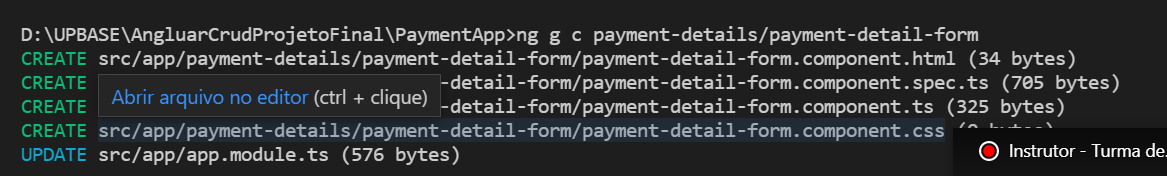
Acessar a pasta PaymentApp 🡪 cd PaymentApp

criar 2 componentes após a instalação, você pode executar os seguintes comandos.

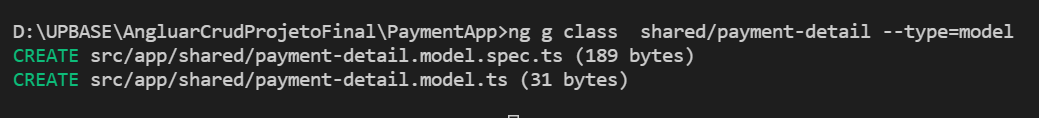
**ng g c payment-details**

****

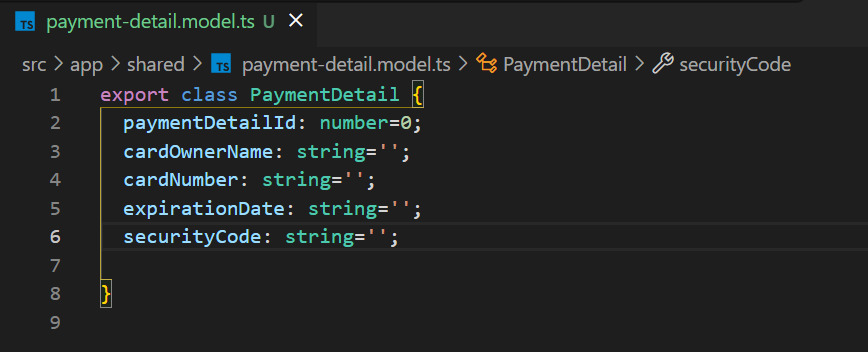
**ng g c payment-details/payment-detail-form**



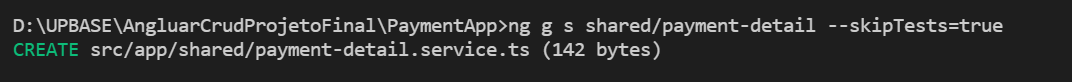
Criar services



Abra o arquivo de model dentro do Angular: **payment-detail.model.ts**



Criar classe de service

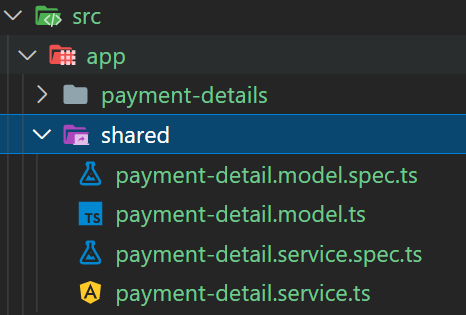


--skipeTests=true 🡪 não cria o arquivo .spec.ts

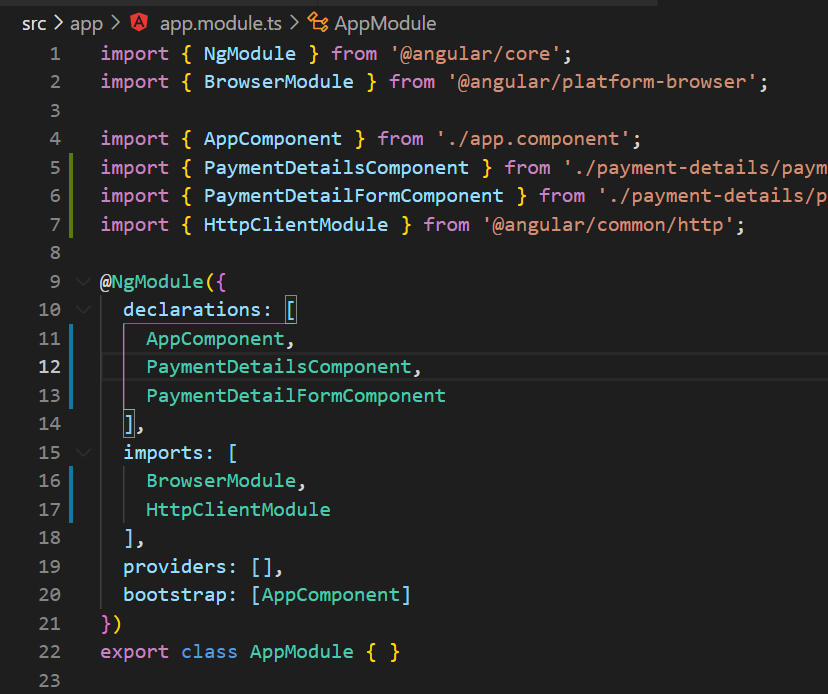
Conteúdo de payment-detail.service.ts

|  |
| --- |
| import { PaymentDetail } from './payment-detail.model';  import { Injectable } from '@angular/core';  import { HttpClient } from "@angular/common/http";  @Injectable({  providedIn: 'root'  })  export class PaymentDetailService {  formData: PaymentDetail= new PaymentDetail();  readonly baseURL = 'http://localhost:61236/api/PaymentDetail';  list : PaymentDetail[];  constructor(private http: HttpClient) { }  postPaymentDetail() {  return this.http.post(this.baseURL, this.formData);  }  putPaymentDetail() {  return this.http.put(`${this.baseURL}/${this.formData.paymentDetailId}`, this.formData);  }  deletePaymentDetail(id: number) {  return this.http.delete(`${this.baseURL}/${id}`);  }  refreshList() {  this.http.get(this.baseURL)  .toPromise()  .then(res =>this.list = res as PaymentDetail[]);  }  } |

Ao final



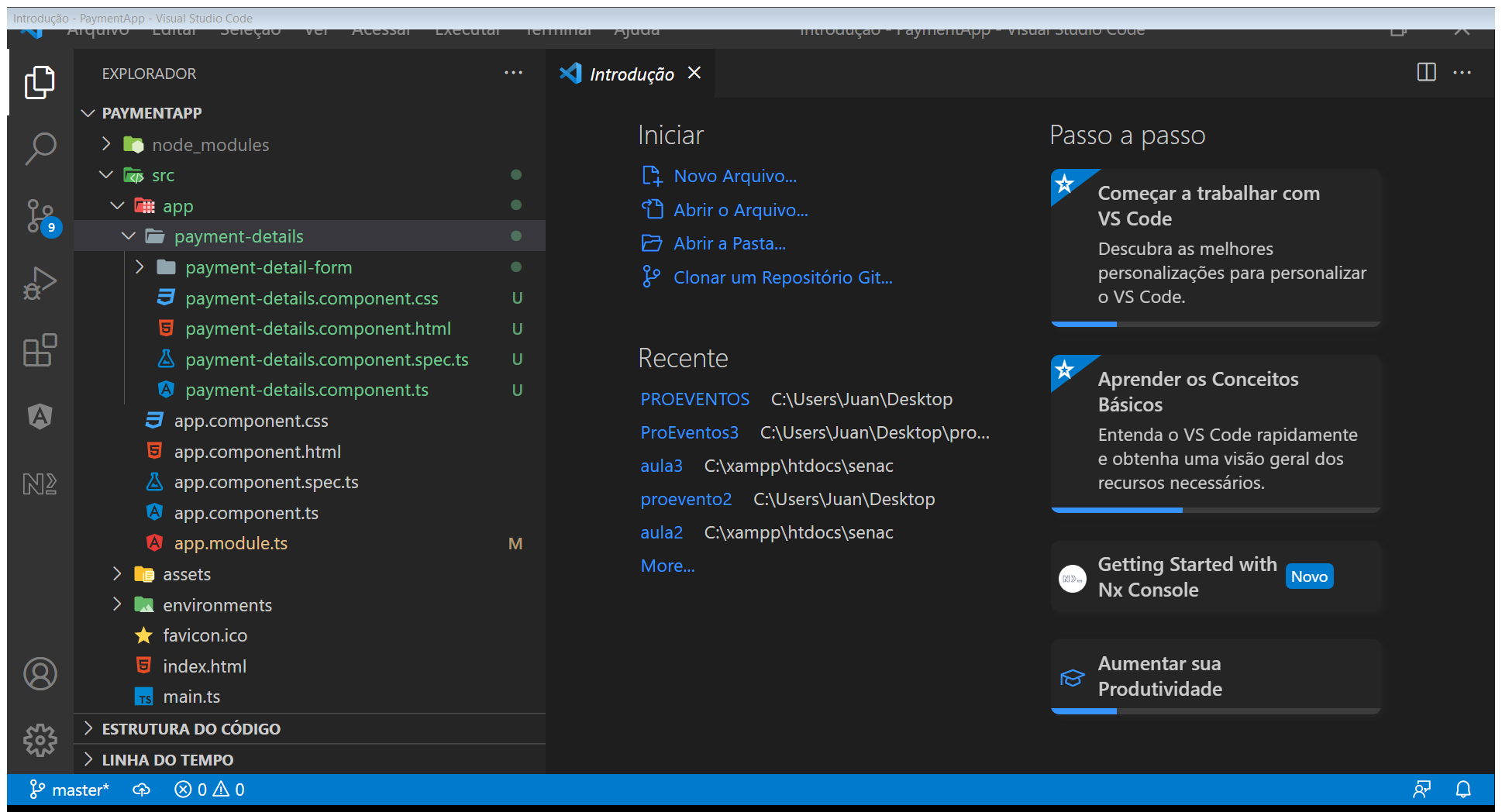
Inserir no app.module.ts as linhas 7 e 17



Estrutura do projeto

|  |
| --- |
| ● src  +---● app  | +--● payment-details  | | |--payment-details.component.ts|.html  | | |  | | +--● payment-detail-form  | | |--payment-detail-form.component.ts|.html  | |  | +--● shared  | | |--payment-detail.service.ts  | | |--payment-detail.model.ts  | |  | |--app.module.ts  |  |--index.html (cdn path for bootstrap & fa icons) |

Após instalação abrir este projeto angular no VSCODE

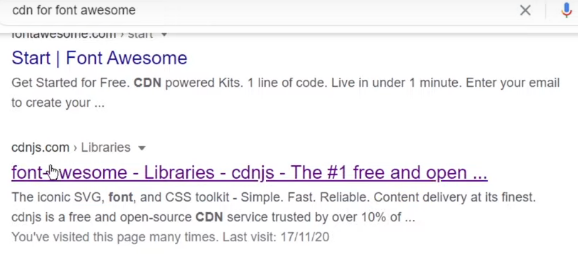


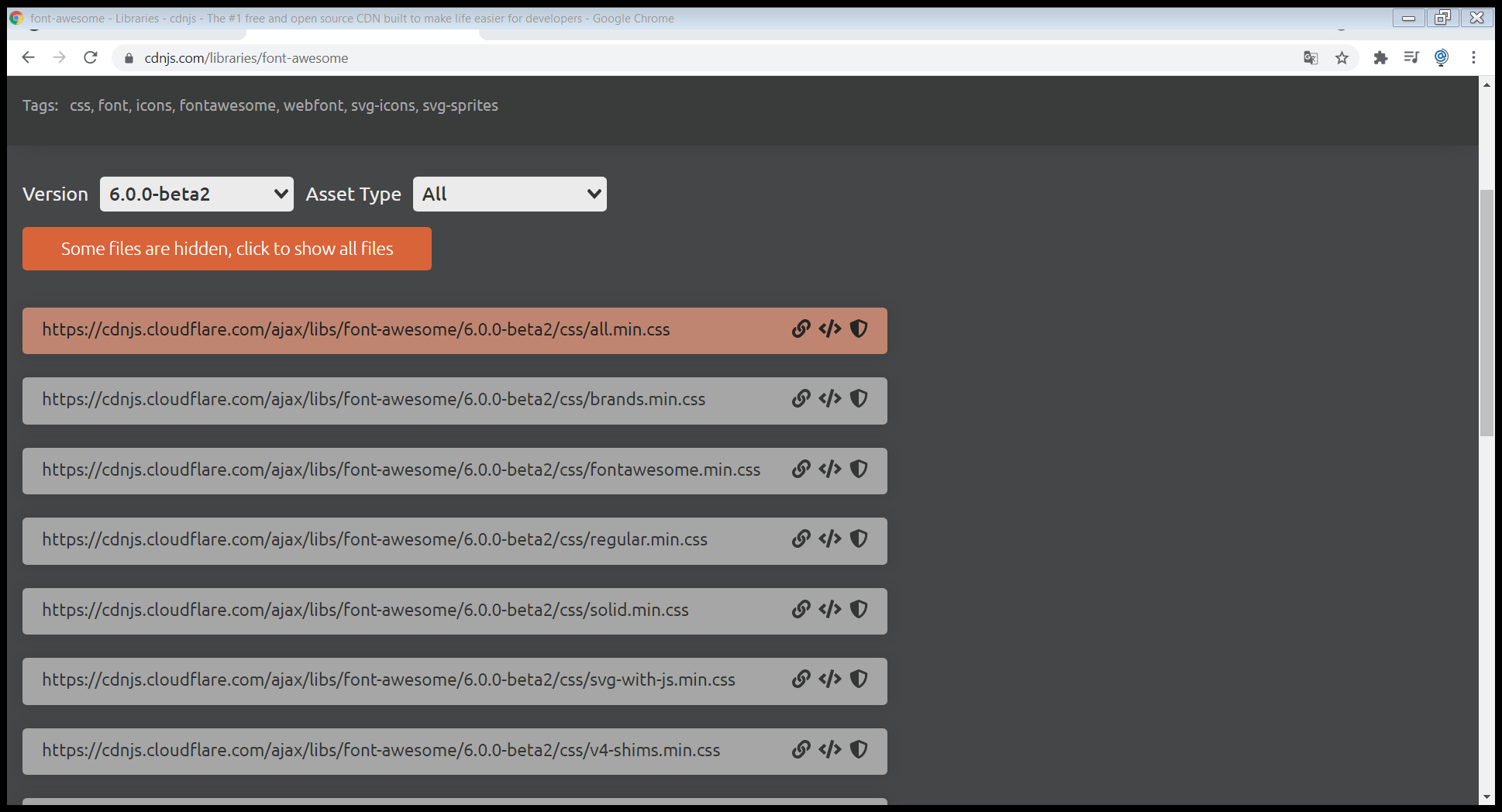
Inserir bootstrap via cdn

<link href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/css/bootstrap.min.css" rel="stylesheet" integrity="sha384-1BmE4kWBq78iYhFldvKuhfTAU6auU8tT94WrHftjDbrCEXSU1oBoqyl2QvZ6jIW3" crossorigin="anonymous">

Colar no head do index.html

Inserir o Fontawesome





**https://cdnjs.cloudflare.com/ajax/libs/font-awesome/6.0.0-beta2/css/brands.min.css**

index.html

|  |
| --- |
| <!doctype html>  <html lang="en">  <head>    <meta charset="utf-8">    <title>PaymentApp</title>    <base href="/">    <meta name="viewport" content="width=device-width, initial-scale=1">    <link rel="icon" type="image/x-icon" href="favicon.ico">    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/css/bootstrap.min.css" rel="stylesheet" integrity="sha384-1BmE4kWBq78iYhFldvKuhfTAU6auU8tT94WrHftjDbrCEXSU1oBoqyl2QvZ6jIW3" crossorigin="anonymous">    <link href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/6.0.0-beta2/css/brands.min.css">  </head>  <body>    <app-root></app-root>  </body>  </html> |

Payment-details.component.html

|  |
| --- |
| <div class="jumbotron py-3">  <h1 class="display-4 text-center">Payment Detail Register</h1>  </div>  <div class="row">  <div class="col-md-5">  <app-payment-detail-form></app-payment-detail-form>  </div>  <div class="col-md-7">  <div>table with list of records from the table</div>  </div>  </div> |

Style.css (global)

|  |
| --- |
| .form-group label {  color: grey;  }  .form-group input {  font-weight: 500;  }  .form-group input::placeholder {  color: #c0bdbd;  font-weight: 300;  }  thead th{  font-weight: 400;  }  table tr:hover {  background-color: #fffbf2;  cursor: pointer;  }  /\*for invalid form controls\*/  input.invalid {  border-color: red;  } |

Abrir Payment-detail.models.ts

Adicionar as propriedades que fazem parte da tabela criada..

|  |
| --- |
| export class PaymentDetail {    paymentDetailId: number=0;      cardOwnerName: string='';      cardNumber: string='';      expirationDate: string='';      securityCode: string='';  } |

Abrir payment-detail.service.ts

|  |
| --- |
| import { PaymentDetail } from './payment-detail.model';  import { Injectable } from '@angular/core';  import { HttpClient } from "@angular/common/http";  @Injectable({    providedIn: 'root'  })  export class PaymentDetailService {    formData: PaymentDetail= new PaymentDetail();    readonly baseURL = 'http://localhost:61236/api/PaymentDetail';    list : PaymentDetail[] | undefined;    constructor(private http: HttpClient) { }    postPaymentDetail() {      return this.http.post(this.baseURL, this.formData);    }    putPaymentDetail() {      return this.http.put(`${this.baseURL}/${this.formData.paymentDetailId}`, this.formData);    }    deletePaymentDetail(id: number) {      return this.http.delete(`${this.baseURL}/${id}`);    }    refreshList() {      this.http.get(this.baseURL)        .toPromise()        .then(res =>this.list = res as PaymentDetail[]);    }  } |

formData propriedade pode ser usada para projetar o formulário para operações CRUD,  list array é usado para armazenar todos os registros da API. baseURLcontém a URL base para o controlador de API da Web. Agora vamos executar a API do Visual Studio -  **Debug> Start Debugging (F5)** .

HttpClient é usado para fazer uma solicitação HTTP ao servidor. Junto com os métodos para operações CRUD,  refreshList funcionamos para preencher os registros existentes nas  list propriedades.

Para usar  HttpClient , também precisamos importar  HttpClientModule . Portanto, acrescente /atualize da **app/app.module.ts** com as linhas

|  |
| --- |
| ...  import { HttpClientModule } from '@angular/common/http';  @NgModule({  ...  imports: [HttpClientModule, ...],  ...  }) |

Agora vamos substituir o arquivo de componente padrão, app.component.htmlcomo segue.

|  |
| --- |
|  |

