Resource Ala Taherzadeh

|  |  |
| --- | --- |
| Code | Description |
| public List<State> FindStateFromSql(string searchFor)  {  var result = \_Context.States.FromSql($"FindState {searchFor}").ToList();  return result;  }  public List<State> FindStateFromSql(string searchFor)  {  var result = \_Context.States.FromSql($"dbo.FindState {searchFor}").ToList();  return result;  }  public List<State> FindStateFromSql(string searchFor)  {  string sql = $"EXEC dbo.FindState {searchFor}";  var trafficList = \_Context.stateViewModels.FromSqlRaw<StateViewModel>(sql).ToList();  return new List<State>();  }  public List<PartViewModel> FindPartAltFromSql(string searchFor)  {  var result = \_Context.PartViewModels.FromSqlRaw($"Exec FindstatesAlt {searchFor}").ToList();  return result;  } | Call Store procedure in .net core 7 without join table  FromSqlRaw امنیت پایین تری دارد. تغییرات زیر لازم است. |
| using BCrypt.Net;  using Microsoft.AspNetCore.Http;  using Microsoft.AspNetCore.Mvc;  using Microsoft.IdentityModel.Tokens;  using System.IdentityModel.Tokens.Jwt;  using System.Security.Claims;  using System.Text;  using TestApiToken.Models;  namespace TestApiToken.Controllers  {  [Route("api/[controller]")]  [ApiController]  public class AuthController : ControllerBase  {  public static User user = new User();  private readonly IConfiguration \_configuration;  //برای استفاده از appsetting باید سازنده را فراخوانی کنیم.  public AuthController(IConfiguration configuration)  {  \_configuration = configuration;  }  [HttpPost("register")]  public ActionResult<User> Register(UserDto requst)  {  //تبدیل رمز به هش کد  string passwordHash  = BCrypt.Net.BCrypt.HashPassword(requst.Password);  //Generate hashCod password password  //اینجا یک رمز به صورت رشته میگیرد و معادلش یک هش کد تولید میکند  user.UserName = requst.UserName;  user.PasswordHash = passwordHash;  return Ok(user);  }  //اگر کاربری در دیتابیس باشد آن را با داده دیتابیس مقایسه میکنیم.  //ولی اینجا برای اموزش api از داده ی خودمان استفاده کردیم  [HttpPost("Login")]  public ActionResult<User> Login(UserDto requst)  {  //چک کردن یوزر نیم با مقدار وارد شده یا مقدار در دیتابیس  if (user.UserName != requst.UserName)  {  return BadRequest("User not found!");  }  //چک کردن رمز با مقدار وارد شده یا در دیتابیس  if (!BCrypt.Net.BCrypt.Verify(requst.Password , user.PasswordHash))  {  return BadRequest("wrong pasword");  }  //return Ok(user); //قبل از نوشتن متد  string token = CreateToken(user); //بعد از نوشتن متد زیر اضافه شد  return Ok(token);  }    //call a private method to create token  private string CreateToken(User user)  {  List<Claim> claims = new List<Claim>  {  new Claim(ClaimTypes.Name, user.UserName),  };  //Generate token web JSON manualy  //تولید یک توکن وب جیسون به صورت دستی  //ابتدا نیاز به یک کلید امنیتی متقارن داریم  //باید رفرنس Microsoft.IdentityModel.Tokens نصب شود  //SymmetricSecurityKeyاین یک کلید است که برای ایجاد توکن وب جیسون نیاز به یک رشته بایتی دارد  //var key = new SymmetricSecurityKey()  //میتوانیم رشته را در appsetting.jason بسازیم  //هر زمان یوزر تماس برقرار کند یا برنامه با توکن وب جیسون تماس برقرار کند، رمز وب جیسون را تایید میکند  var key = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(  \_configuration.GetSection("AppSettings:Token").Value!));  //با انکدینگ دوباره این را رمز نگاری میکنیم  //حالا به اعتبار نامه های امضا نیاز داریم  //signin credentials  //درونش از key و الگورتمی که می خواهیم برای توکن وب جیسون خود استفاده کنیم را مینویسیم  var creds = new SigningCredentials(key, SecurityAlgorithms.HmacSha512Signature);  //Generate our token  var token = new JwtSecurityToken(  claims : claims,  expires : DateTime.Now.AddDays(1), //ست کردن زمان منقضی شدن توکن  signingCredentials : creds //اعتبارنامه امضا  );  //مرحله آخر نوشتن توکن است و برای آن JWT خود را ذخیره میکنیم.  var jwt = new JwtSecurityTokenHandler().WriteToken(token);  return jwt;  }  }  } | Generate token  تولید یک توکن با user و pass |
| string passwordHash  = BCrypt.Net.BCrypt.HashPassword(request.Password); | ساخت رمز hashcode |
| #region Authentication  builder.Services.AddAuthentication(options =>  {  options.DefaultAuthenticateScheme = CookieAuthenticationDefaults.AuthenticationScheme;  options.DefaultChallengeScheme = CookieAuthenticationDefaults.AuthenticationScheme;  options.DefaultSignInScheme = CookieAuthenticationDefaults.AuthenticationScheme;  }).AddCookie(options =>  {  options.LoginPath = "/Login";  options.LogoutPath = "/";  options.ExpireTimeSpan = TimeSpan.FromMinutes(60);  });  #endregion | مربوط به claims configuration  Program.cs |
| [HttpPost]  [Route("Login")]  public async Task<IActionResult> Login(LoginViewModel login)  {  if (!ModelState.IsValid)  {  return View(login);  }  var token = await \_loginService.Login(login.UserName, login.Password);  if (token.IsNullOrEmpty())  {  ViewBag.ErrorLogin = "اطلاعات کاربری یافت نشد!";  return View(new LoginViewModel());  }  GreetingUserInfoViewModel userInfo = await \_loginService.GetUserInfo(token);  #region SetCookie  var claims = new List<Claim>()  {  //Guid Need  new Claim(ClaimTypes.Name, login.UserName.Trim()),  new Claim("FullName", userInfo.Name + " " + userInfo.Family),  new Claim("Token", token),  new Claim("UserId", "1"),  //new Claim (ClaimTypes.NameIdentifier, login.UserName.Trim()),  //new Claim(ClaimTypes.Name, userInfo.Name + " " + userInfo.Family),  //new Claim("UserId", userInfo.UserId.ToString()),  };  var identity = new ClaimsIdentity(claims, CookieAuthenticationDefaults.AuthenticationScheme);  var principal = new ClaimsPrincipal(identity);  var properties = new AuthenticationProperties  {  IsPersistent = login.RememberMe,  };  await HttpContext.SignInAsync(principal, properties);  #endregion  if (TempData["Route"] != null)  {  return Redirect(TempData["Route"].ToString());  }  return RedirectToAction("Index", "Home");  } | ذخیره کردن token در claim  Login method for claim |
| public async Task<string> Login(string userName, string password)  {  var values = new Dictionary<string, string>();  values.Add("grant\_type", "password");  values.Add("username", userName.Trim());  values.Add("password", password);  using var client = new HttpClient();  using var req = new HttpRequestMessage  (HttpMethod.Post, "https://najm.samanpl.ir/oauth/token")  { Content = new FormUrlEncodedContent(values) };  using var res = await client.SendAsync(req);  var token = "";  if (res.IsSuccessStatusCode)  {  var jsonString = await res.Content.ReadAsStringAsync();  var response = JsonConvert.DeserializeObject<TokenViewModel>(jsonString);  token = response.access\_token;  }  return token;  } | کد دریافت توکن |
| برای یک پروژه با معماری تمیز در ASP.NET Core 3.1، می‌توانید از معماری Clean Architecture استفاده کنید که به شما کمک می‌کند کد خود را به قسمت‌های مختلف تقسیم کنید و لایه‌های مختلف برنامه را جدا از یکدیگر نگه دارید.  در این معماری، برنامه به چند لایه اصلی تقسیم می‌شود که عبارتند از:  1. Domain Layer: این لایه شامل مدل‌ها، انتیتی‌ها، و قوانین کسب و کار است.  2. Application Layer: در این لایه قوانین کسب و کار اجرا می‌شوند و ارتباط با لایه‌های دیگر برقرار می‌شود.  3. Infrastructure Layer: در این لایه قرار دادن وابستگی‌های بیرونی مانند پایگاه داده، فایل‌ها، و غیره انجام می‌شود.  4. Presentation Layer: این لایه شامل واسط کاربری مانند وب‌اپلیکیشن یا API است.  یک sample کد برای یک پروژه با معماری Clean Architecture به صورت خلاصه می‌تواند به این شکل باشد:  - MySolution  - MySolution.Domain  - Models  - Services  - MySolution.Application  - Interfaces  - Services  - MySolution.Infrastructure  - Data -Context  - Repositories  - MySolution.Web  - Controllers  - Views  در این ساختار، لایه Domain شامل مدل‌ها و قوانین کسب و کار است، لایه Application برای اجرای قوانین کسب و کار و ارتباط با لایه‌های دیگر مسئول است، لایه Infrastructure برای قرار دادن وابستگی‌های بیرونی مانند پایگاه داده مسئول است و لایه Presentation برای ارائه واسط کاربری به کاربر مسئول است.  این ساختار به شما کمک می‌کند تا کد خود را به قسمت‌های قابل مدیریت تقسیم کنید و امکان توسعه و نگهداری آن را بهبود بخشید. | معماری تمیز |
| int a = 25000;  Console.WriteLine(a.ToString("#,0 Rial")); | برای برنامه حسابداری و جدا کردن سه رقم سه رقم مبالغ از extention method ی به نام Tostring استفاده میکنیم |
|  |  |
| public class Student  {  public int Id { get; set; }  public string? Name { get; set; }  public int Mark { get; set; }  public int CourseId { get; set; }  public Course? Course { get; set; }  }  And the Course class:  public class Course  {  public int Id { get; set; }  public string? Title { get; set; }  public ICollection<Student>? Students { get; set; }  } |  |
| public List<UserBorrow> GetUserBorrows(Guid userId)  {  List<UserBorrow> result = null;  var userIdParam = new SqlParameter("@UserId",userId);  result = \_apiContext.UserBorrows  .FromSqlRaw("EXEC sp\_UserBorrowAPI @UserId", userIdParam)  .ToList();  return result;  } | Call sp با یک پارامتر ورودی |
| public List<UserBorrow> GetUserBorrows(Guid userId ,long NationalCode)  {  List<FinancialHistoryViewModel> result = null;  var userIdParam = new SqlParameter("@UserId", userId);  var nationalCodeParam = new SqlParameter("@NationalCode", nationalCode);  result = \_apiContext.financialHistoryViews  .FromSqlRaw("EXEC sp\_Services\_PayHistory @UserId, @NationalCode", userIdParam, nationalCodeParam)  .ToList();  return result;  { | Call sp با چند پارامتر ورودی |
| public int RequestDoc(BorrowRequestedDocuments brwRequest)  {  try  {  var result = new SqlParameter("@Result", SqlDbType.Int) { Direction = ParameterDirection.Output };  var orgId = new SqlParameter("@OrgId", brwRequest.OrgId);  var userId = new SqlParameter("@UserId", brwRequest.UserId);  var noskheId = new SqlParameter("@NoskheID", brwRequest.NoskheID);  var requestDate = new SqlParameter("@RequestDate", brwRequest.RequestDate);  var isPrint = new SqlParameter("@IsPrint", brwRequest.IsPrint);  var isReady = new SqlParameter("@IsReady", brwRequest.IsReady);  var readyTime = new SqlParameter("@ReadyTime", DBNull.Value);  \_apiContext.Database.ExecuteSqlRaw("EXEC sp\_borrow\_RequestDoc @OrgId, @UserId, @NoskheID, @RequestDate, @IsPrint, @IsReady, @ReadyTime, @Result",  orgId, userId, noskheId, requestDate, isPrint, isReady, readyTime, result);  return (int)result.Value;  }  catch (Exception ex)  {  // مدیریت خطا و بازگشت مقدار مناسب  return -1; // یک مقدار خطای پیش فرض  }  } | روش دیگر call SP |
| <https://stackoverflow.com/questions/45252959/entity-framework-core-using-stored-procedure-with-output-parameters> لینک  It should work, but I believe you also need to include the parameter names and the OUT keyword in the command statement  var sql = "exec spTestSp @ParamIn1, @ParamIn2, @ParamOut1 OUT, @ParamOut2 OUT";  var result = db.Database.ExecuteSqlCommand(sql, in1, in2, out1, out2);  var out1Value = (long) out1.Value;  var out2Value = (string) out2.Value; |  |
| //call View from Database  public async Task<List<LibOrgViewModelViews>> GetLibrariesListView()  {  try  {  var report = await \_apiContext.LibrariesList\_view.ToListAsync();  return report.Take(20).ToList();  }  catch (Exception ex)  {  return null;  }  } | برای فراخوانی view object که در sql ساخته شده:  LibrariesList\_view نام view ساخته شده در database است. |
| به جای استفاده از `JsonResult`، می‌توانید از دیگر نوع‌های `IActionResult` نیز استفاده کنید. به عنوان مثال:  0. if (copyId == null)  {  var data = new { StatusPhrase = "copyId is null!", Status = -1, Result = "" };  return new JsonResult(data);  }  1. \*\*BadRequest\*\*: اگر داده‌های ورودی نامعتبر باشند، می‌توانید از `BadRequest` استفاده کنید.  ```csharp  return BadRequest("copyId is null!");  ```  2. \*\*NotFound\*\*: اگر منبع مورد نظر یافت نشود، می‌توانید از `NotFound` استفاده کنید.  ```csharp  return NotFound("Resource not found");  ```  3. \*\*Ok\*\*: اگر عملیات با موفقیت انجام شود، می‌توانید از `Ok` استفاده کنید.  ```csharp  return Ok();  ```  4. \*\*StatusCode\*\*: اگر نیاز به برگرداندن یک کد وضعیت خاص دارید، می‌توانید از `StatusCode` استفاده کنید.  ```csharp  return StatusCode(200, "Success");  ```  یا  if (!User.Identity.IsAuthenticated)  {  return StatusCode(401, new { Message = "کاربر وارد سیستم نشده است" });  }  اینها تنها چند مثال از راه‌های دیگر برای برگرداندن `IActionResult` هستند. شما می‌توانید بر اساس نیاز و شرایط خاص خود از نوع مناسب برای برگرداندن نتیجه استفاده کنید. |  |
| در عبارت  `User.Claims.FirstOrDefault(u => u.Type == "UserId").Value ?? Guid.NewGuid().ToString()`،  علت استفاده از `??` این است که این عملگر یک عملگر coalescing null است که در C# برای انجام یک عملیات پیش‌فرض در صورتی که مقدار null باشد، استفاده می‌شود.  در اینجا، ابتدا ما از `User.Claims.FirstOrDefault(u => u.Type == "UserId").Value` استفاده می‌کنیم تا مقدار UserId کاربر را بدست آوریم. اما اگر این مقدار null باشد (به عبارت دیگر، اگر کلیم‌های Claims کاربر حاوی مقدار UserId نباشند)، ما از `Guid.NewGuid().ToString()` استفاده می‌کنیم تا یک GUID جدید ایجاد کرده و آن را به عنوان مقدار UserId استفاده کنیم. وقتی می‌گوییم که کلیم‌های `Claims` کاربر حاوی مقدار UserId نباشند، منظور این است که در اطلاعات کلیم‌های کاربر (مانند نام، ایمیل، نقش‌ها و غیره)، مقداری برای `UserId` تعریف نشده است یا مقدار `UserId` خالی یا null است.  در برنامه‌های وب، معمولاً اطلاعات کاربری در قالب کلیم‌های `Claims` ذخیره می‌شود تا بتوان به راحتی به اطلاعات کاربر دسترسی داشت. اگر مقدار `UserId` در کلیم‌های `Claims` کاربر تعریف نشده باشد، باید یک مقدار پیش‌فرض مانند یک GUID جدید برای `UserId` ایجاد و استفاده کنیم، که به این ترتیب از وجود مقدار `null` برای `UserId` جلوگیری می‌شود و برنامه به صورت درست ادامه می‌یابد.  به این ترتیب، با استفاده از `??` در اینجا، اطمینان حاصل می‌شود که در صورتی که مقدار UserId موجود نباشد، یک GUID جدید برای آن ایجاد و استفاده می‌شود. |  |
| public NoNoskhe12 GetCopyById(long id)  {  try  {  if (\_apiContext.NoNoskhe12.Any(n => n.NoskheID == id))  {  NoNoskhe12 copy = \_apiContext.NoNoskhe12.Find(id);  }  }  catch (Exception)  {  throw;  }  return new NoNoskhe12();  } | چک کردن وجود یا عدم وجود یک دیتا در جدول با any |
| var resultParam = new SqlParameter("@Result", SqlDbType.Int);  resultParam.Direction = ParameterDirection.Output;  try  {  var query = "EXEC sp\_Services\_PayHistory @OrgId, @UserId, @NoskheID, @RequestDate, @IsPrint, @IsReady, @ReadyTime, @Result OUT";  var parameters = new[] { orgIdParam, userIdParam, copyCodeParam, requestDateParam, isPrintParam, isReadyParam, readyTimeParam, resultParam };  \_apiContext.Database.ExecuteSqlRaw(query, parameters);  return (int)resultParam.Value;  }  catch (Exception ex)  {  return 0;  } | Call sp in .net core7 زمانیکه result داریم. |
| public int ReserveDoc(Domain.BrwReserveDocument brwReserve)  {  //db.BrwReserveDocuments.Add(brwReserve);  //db.SaveChanges();  var Result = new SqlParameter();  Result.ParameterName = "Result";  Result.SqlDbType = SqlDbType.Int;  Result.Direction = ParameterDirection.Output; | در .net framework |
| public string DigitEngToFa(string result)  {  string test = "";  if (result != null)  {  test = result;  test = test.Replace("1", "۱").Replace("2", "۲").Replace("3", "۳").Replace("4", "۴").Replace("5", "۵").Replace("6", "۶").Replace("7", "۷").Replace("8", "۸").Replace("9", "۹").Replace("0", "۰");  }  return test;  } | متد تغییر ارقام انگلیسی به فارسی |
| List<Guid> userInRole = \_apiContext.UsersInRoles.Where(r => r.RoleId == roleId).Select(u => u.UserId).ToList();  این کد با استفاده از Entity Framework Core و LINQ، کاربرانی را که در یک نقش خاص (با شناسه `roleId`) قرار دارند، از دیتابیس بازیابی می‌کند.  1. `\_apiContext.UsersInRoles`: این بخش از کد به `DbSet` کلاس `UsersInRoles` در مدل دیتابیس شما اشاره دارد، که احتمالا معادل یک جدول در دیتابیس شماست که اطلاعات مربوط به رابطه بین کاربران و نقش‌ها را نگهداری می‌کند.  2. `Where(r => r.RoleId == roleId)`: این قسمت از کد یک فیلتر را در نظر می‌گیرد که فقط کاربرانی را که دارای نقش با شناسه (`roleId`) مشخص هستند انتخاب می‌کند.  3. `Select(u => u.UserId)`: این بخش از کد مشخص می‌کند که فقط فیلد `UserId` از نتایج فیلتر شده برگردانده شود.  4. `ToList()`: این متد LINQ نتیجه نهایی را به یک `List<Guid>` تبدیل می‌کند و از دیتابیس بازیابی می‌کند.  با اجرای این کد، شما یک لیست از شناسه‌های کاربرانی که در نقش مشخص شده (`roleId`) قرار دارند، برگردانده خواهید شد. این لیست از `Guid`ها مشخصه‌های منحصر به فرد است که برای هر کاربر استفاده می‌شود. | معنی این خط کد |
| var builder = new ConfigurationBuilder()              .SetBasePath(Directory.GetCurrentDirectory())              .AddJsonFile("appsettings.json", optional: true, reloadOnChange: true);          IConfigurationRoot configuration = builder.Build();          var blindRoleId = configuration["BlindRoleId"];          if (membershipTypeId == 1 || (membershipTypeId == 4 && role.Contains(Guid.Parse(blindRoleId))))          {              userMember = GetUserMember(user.UserId, user.OrgId, Guid.Parse(blindRoleId));              //var dsr = new DataSourceResult();              if (userMember.Count > 0)              {                  var library = \_libOrgRepository.GetAllLibOrg();                  var membershipType = GetUserMembershipTypes();  در appsetting هم  "BlindRoleId": "51F4F935-9406-497D-92BC-DF080D176AA8", | استفاده از configuration و key |
| // "E:\\197\\New folder\\RefahFiles\\1\\python-programming-language-coding.jpg", برای چک کردن لوکال  string ImageAddress = "G:\\197\\New folder\\Catalog Image\\" + ImageFolder + "\\" + rcc + ".jpg"; آدرس در سرور لوکال IPLF  public string ImageToBase64(string strImagePath)  {  if (string.IsNullOrEmpty(strImagePath) || !System.IO.File.Exists(strImagePath))  {  return "";  }  var imageBytArray = System.IO.File.ReadAllBytes(strImagePath);  return Convert.ToBase64String(imageBytArray);  }  var houseInformation = new RefLibrarianHouseApiView()  {  id = selectedHouse.id,  StateName = selectedHouse.StateName,  Name = selectedHouse.Name,  TownshipName = selectedHouse.TownshipName,  Address = selectedHouse.Address,  Phone = selectedHouse.Phone,  Area = selectedHouse.Area,  Units = selectedHouse.Units,  Image = ImageToBase64(selectedHouse.Image),  PopularityTitle = selectedHouse.PopularityTitle,  };  houseInformation.Image = ImageToBase64(selectedHouse.Image);  houseInformation.Image2 = ImageToBase64(selectedHouse.Image2); | Image  تبدیل به base64 برای حجم کمتر |
| using Microsoft.AspNetCore.Mvc;  using System;  namespace YourNamespace.Controllers  {  [ApiController]  [Route("api/[controller]")]  public class YourController : ControllerBase  {  [HttpGet]  public IActionResult YourAction()  {  // اینجا می‌توانید یک متد خصوصی تعریف کنید  void YourPrivateMethod()  {  Console.WriteLine("This is a private method inside the action.");  }  YourPrivateMethod(); // فراخوانی متد خصوصی  return Ok();  }  }  } | برای تعریف یک متد درون یک action در یک API، می‌توانید با استفاده از کلیدواژه private، یک متد خصوصی (private method) داخل اکشن مورد نظر تعریف کنید. در زیر یک مثال از چگونگی تعریف یک متد خصوصی درون یک اکشن در ASP.NET Core API آورده شده است: |
| public string CleanText(string text)  {  if (text == null) return "";  text = text.Trim();  var charsToRemove = new string[] { "@", ",", ".", ";", "'", "،", "؛", "-", "\_", " " };  if (text.EndsWith("...؛"))  {  charsToRemove = new string[] { "@", ",", "", ";", "'", "،", "؛", "-", "\_", " " };  }  if (!string.IsNullOrEmpty(text) && !string.IsNullOrWhiteSpace(text))  {  char First;  char Last;  First = text[0];  Last = text[text.Length - 1];  while (charsToRemove.Contains(First.ToString()) && text.Length > 1)  {  text = text.Substring(1, text.Length - 1);  First = text[0];  }  while (charsToRemove.Contains(Last.ToString()) && (!string.IsNullOrEmpty(text) && !string.IsNullOrWhiteSpace(text)) && text.Length > 1)  {  text = text.Substring(0, text.Length - 1);  Last = text[text.Length - 1];  }  if (text.Length == 1)  {  if (charsToRemove.Contains(text))  {  text = "";  }  }  }  return text;  } | متد clean Text از کاراکترهای اضافه |
| public void Exercise4()  {  Console.WriteLine("Enter some worlds");  var totalWord = Console.ReadLine();  var varName = "";  foreach (var word in totalWord.Split(' '))  {  var wPascal = char.ToUpper(word[0]) + word.ToLower().Substring(1);  varName += wPascal;  }  Console.WriteLine(varName);  } | برنامه ای بنویسید که تعدادی کلمه با space بگیرد و به روش camel case آن ها را نمایش دهد.  ورودی:  "number of students"  خروجی: "NumberOfStudents"  ورودی: "NUMBER OF STUDENTS"  خروجی:  "NumberOfStudents" |
| https://learn.microsoft.com/en-us/answers/questions/619954/how-to-use-sql-view-in-mvc-core-page | نحوه ی ساخت view به روش code first و |
|  |  |
|  |  |
|  |  |
|  |  |
| l.Rowversion, |  |