Tema III. Autentificare JWT + aplicatie de test

Echipa:

Draghici Andreea-Maria

Ciontu Claudia-Elena

Diaconescu Razvan

Sectia:

CR4.S1A

Profesor Coordonator:

Badea Gabriel

Cuprins

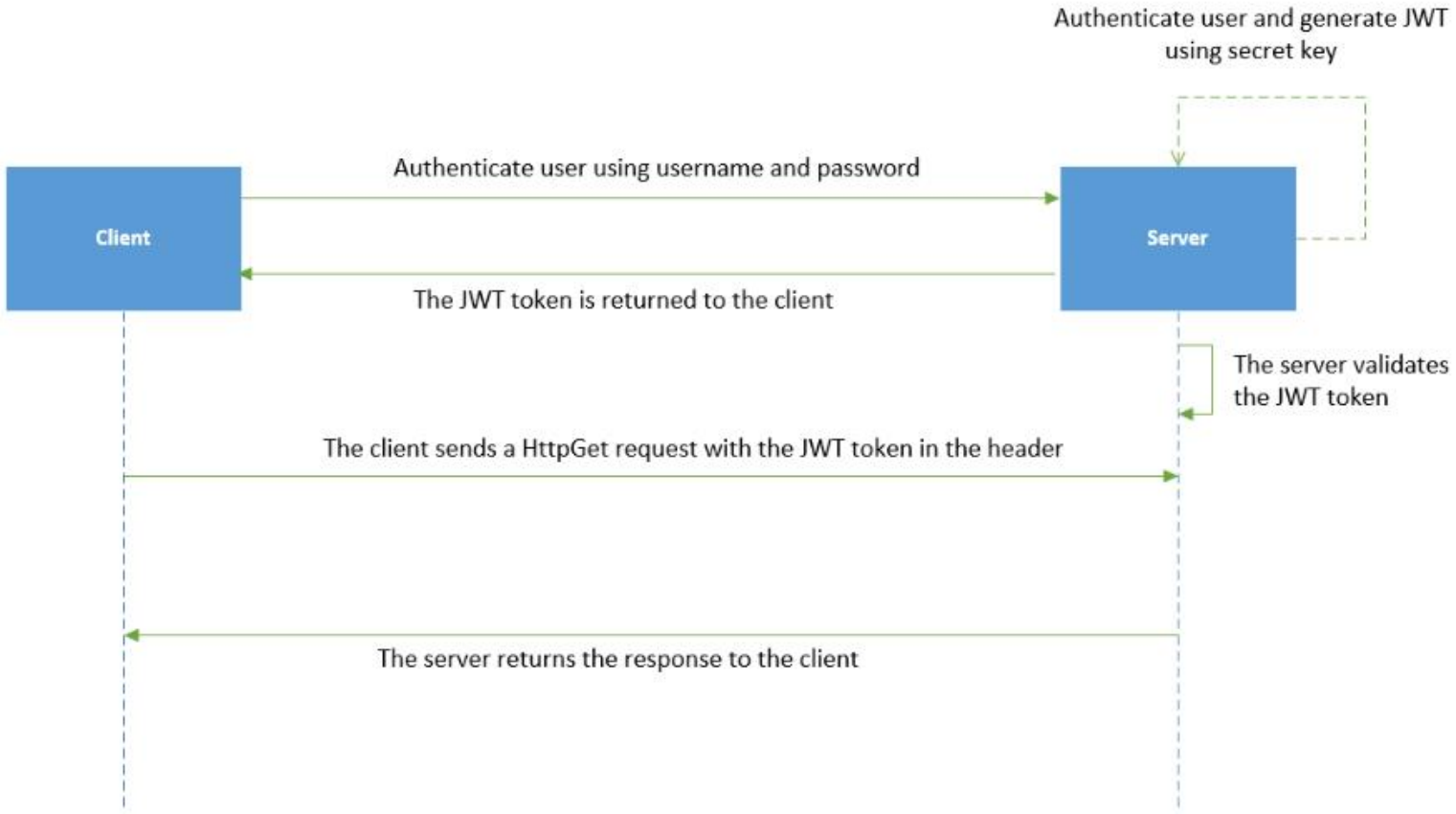
What Are JSON Web Tokens (JWT)?

JSON **W**eb **T**oken is an open standard (RFC 7519**)** **that defines a safe, compact, and self-contained, secured way for transmission of information between a sender and a receiver through a URL, a POST parameter, or inside the HTTP Header.** It should be noted that the information to be transmitted securely between two parties is represented in JSON format and it is cryptographically signed to verify its authenticity.

JWT is typically used for implementing authentication and authorization in Web applications. Because JWT is a standard, all JWTs are tokens but the reverse is not true. You can work with JSON Web Tokens in .NET, Python, Node.js, Java, PHP, Ruby, Go, JavaScript, etc.

ILLUSTRATES HOW A TYPICAL JWT AUTHENTICATION WORKS

JSON Web Token (JWT) is an open standard (RFC 7519) that defines how you can securely transfer information between two parties.



GETTING STARTED

Steps: Open VS Community 2019 -> Create a new project -> ASP.Net Core Web App(Model-View-Controller)

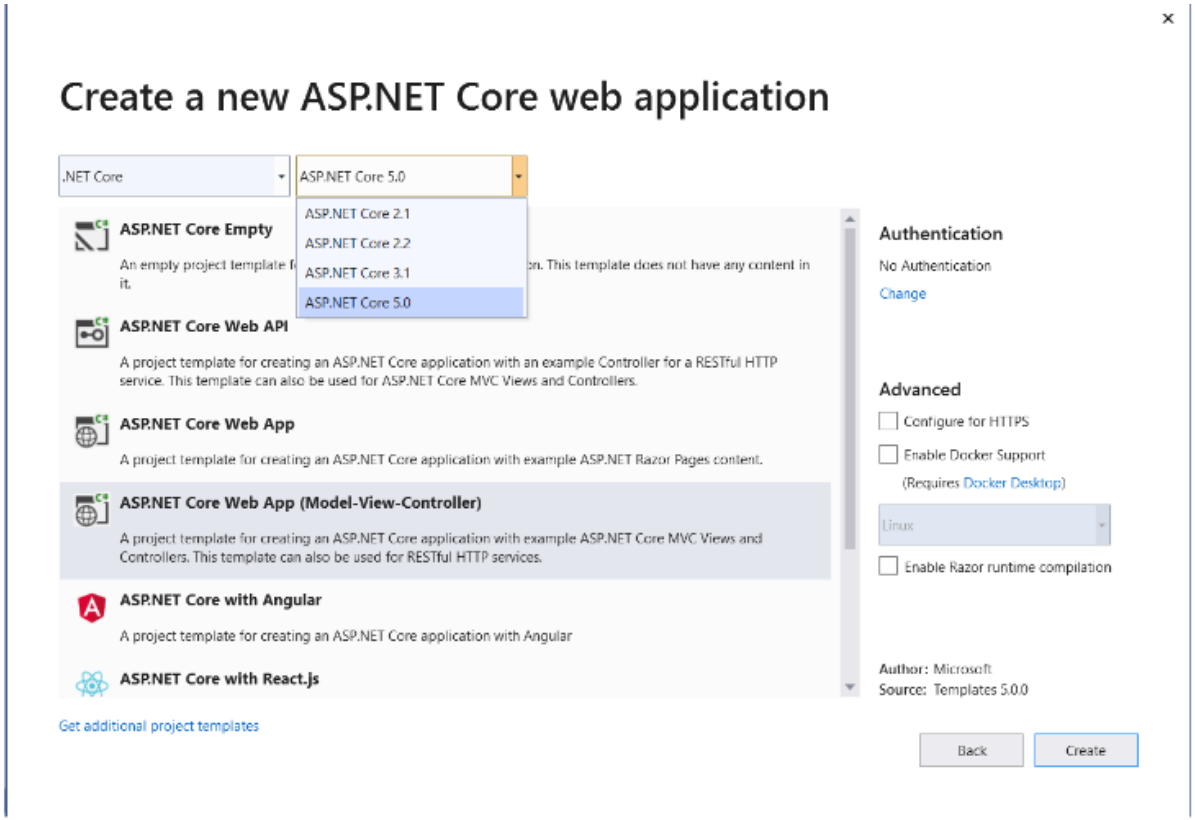


Figure 2: Select the project template and specify authentication and the target framework.

The next step is to install the necessary NuGet Package(s).

To install the required packages into your project, execute the following commands at the NuGet Package Manager Console.

- dotnet add package Microsoft.AspNetCore.Authentication
- dotnet add package Microsoft.AspNetCore.Authentication.JwtBearer

Implementing JWT in ASP.NET Core 5 MVC

- **Creating the Model Classes**

```
using System.ComponentModel.DataAnnotations;

namespace JWTASPNetCore.Models
{
    7 references
    public class UserModel
    {
        [Required]
        5 references
        public string UserName { get; set; }

        [Required]
        5 references
        public string Password { get; set; }
    }
}
```

```
namespace JWTASPNetCore.Models
{
    12 references
    public class UserDTO
    {
        7 references
        public string UserName { get; set; }
        6 references
        public string Password { get; set; }
        6 references
        public string Role { get; set; }
    }
}
```

The User**DTO** represents the user **data transfer object**.

Configuring JWT in the AppSettings File

Created a section in the appsettings.json file called Jwt with the following content inside:

Schema: https://json.schemastore.org/appsettings.json

```
1  {
2    "Logging": {
3      "LogLevel": {
4        "Default": "Information",
5        "Microsoft": "Warning",
6        "Microsoft.Hosting.Lifetime": "Information"
7      }
8    },
9    "Jwt": {
10     "Key": "ThisismySecretKey",
11     "Issuer": "www.draghiciandreea.net",
12     "Audience": "http://localhost:36145/"
13   },
14   "AllowedHosts": "*"
15 }
```

Configure Authentication with Bearer and JWT

```
// This method gets called by the runtime. Use this method to add services to the container.
0 references
public void ConfigureServices(IServiceCollection services)
{
    services.AddSession();
    services.AddControllersWithViews();
    services.AddTransient<IUserRepository, UserRepository>();
    services.AddTransient<ITokenService, TokenService>();
    services.AddTransient<IAuthenticationService, AuthenticationService>();

    services.AddAuthentication(auth =>
    {
        auth.DefaultAuthenticateScheme = JwtBearerDefaults.AuthenticationScheme;
        auth.DefaultChallengeScheme = JwtBearerDefaults.AuthenticationScheme;
    })
    .AddJwtBearer(options =>
    {
        options.TokenValidationParameters = new TokenValidationParameters
        {
            ValidateIssuer = true,
            ValidateAudience = true,
            ValidateLifetime = true,
            ValidateIssuerSigningKey = true,
            ValidIssuer = Configuration["Jwt:Issuer"],
            ValidAudience = Configuration["Jwt:Issuer"],
            IssuerSigningKey = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(Configuration["Jwt:Key"]))
        };
    });
}
```

In Startup.cs class, in the ConfigureServices method add the AddAuthentication feature as well as JwtBearer using AddJwtBearer method.

Added a transient service of type IUserRepository and ITokenService.

The code snippet below shows how you can retrieve the generated token from the session and then add it as a bearer token in the request header.

```
// This method gets called by the runtime. Use this method to configure the HTTP request pipeline.
0 references
public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
{
    if (env.IsDevelopment())
    {
        app.UseDeveloperExceptionPage();
    }
    else
    {
        app.UseExceptionHandler("/Home/Error");
    }

    app.UseSession();

    app.Use(async (context, next) =>
    {
        var token = context.Session.GetString("Token");
        if (!string.IsNullOrEmpty(token))
        {
            context.Request.Headers.Add("Authorization", "Bearer " + token);
        }
        await next();
    });

    app.UseStaticFiles();
    app.UseRouting();
    app.UseAuthentication();
    app.UseAuthorization();

    app.UseEndpoints(endpoints =>
    {
        endpoints.MapControllerRoute(
            name: "default",
            pattern: "{controller=Home}/{action=Index}/{id?}");
    });
}
```

Create the UserRepository Class

A repository class is an implementation of the Repository design pattern and is one that manages data access. The application takes advantage of the repository instance to perform CRUD operations against the database. In this example, the HomeController interacts with the UserRepository to retrieve a user based on the username and password.

The UserRepository class extends the IUserRepository interface and implements the GetUser method, it also builds a list of UserDTO objects.

Assumptions: The password here has been hardcoded for simplify.

```
namespace JWTASPNetCore.Interfaces
{
    4 references
    public interface IUserRepository
    {
        2 references
        UserDTO GetUser(UserModel userModel);
    }
}
```

```
namespace JWTASPNetCore.Repository
{
    2 references
    public class UserRepository : IUserRepository
    {
        private readonly List<UserDTO> users = new List<UserDTO>();
        0 references
        public UserRepository()
        {
            users.Add(new UserDTO { UserName = "draghiciandreea", Password = "draghi123", Role = "manager" });
            users.Add(new UserDTO { UserName = "diaconescurazvan", Password = "razvan321", Role = "developer" });
            users.Add(new UserDTO { UserName = "ciontuclaudia", Password = "clau123", Role = "tester" });
            users.Add(new UserDTO { UserName = "rodpaddock", Password = "rod123", Role = "admin" });
            users.Add(new UserDTO { UserName = "admin", Password = "admin321", Role = "admin" });
        }
        2 references
        public UserDTO GetUser(UserModel userModel)
        {
            return users.Where(x => x.UserName.ToLower() == userModel.UserName.ToLower()
                && x.Password == userModel.Password).FirstOrDefault();
        }
    }
}
```

Create the TokenService Class

Create an interface called ITokenService.
The TokenService class extends the ITokenService interface and implements its methods as shown:

```
namespace JWTASPNetCore.Interfaces
{
    4 references
    public interface ITokenService
    {
        2 references
        string BuildToken(string key, string issuer, UserDTO user);
        2 references
        bool IsTokenValid(string key, string issuer, string token);
    }
}
```

```
namespace JWTASPNetCore.Service
{
    1 reference
    public class TokenService : ITokenService
    {
        private const double EXPIRY_DURATION_MINUTES = 30;
        2 references
        public string BuildToken(string key,
        string issuer, UserDTO user)
        {
            var claims = new[] {
                new Claim(ClaimTypes.Name, user.UserName),
                new Claim(ClaimTypes.Role, user.Role),
                new Claim(ClaimTypes.NameIdentifier, Guid.NewGuid().ToString())
            };

            var securityKey = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(key));
            var credentials = new SigningCredentials(securityKey, SecurityAlgorithms.HmacSha256Signature);
            var tokenDescriptor = new JwtSecurityToken(issuer, issuer, claims,
                expires: DateTime.Now.AddMinutes(EXPIRY_DURATION_MINUTES), signingCredentials: credentials);
            return new JwtSecurityTokenHandler().WriteToken(tokenDescriptor);
        }

        2 references
        public bool IsTokenValid(string key, string issuer, string token)
        {
            var mySecret = Encoding.UTF8.GetBytes(key);
            var mySecurityKey = new SymmetricSecurityKey(mySecret);

            var tokenHandler = new JwtSecurityTokenHandler();
            try
            {
                tokenHandler.ValidateToken(token, new TokenValidationParameters
                {
                    ValidateIssuerSigningKey = true,
                    ValidateIssuer = true,
                    ValidateAudience = true,
                    ValidIssuer = issuer,
                    ValidAudience = issuer,
                    IssuerSigningKey = mySecurityKey,
                }, out SecurityToken validatedToken);
            }
            catch
            {
                return false;
            }
            return true;
        }
    }
}
```

Create the HomeController Class

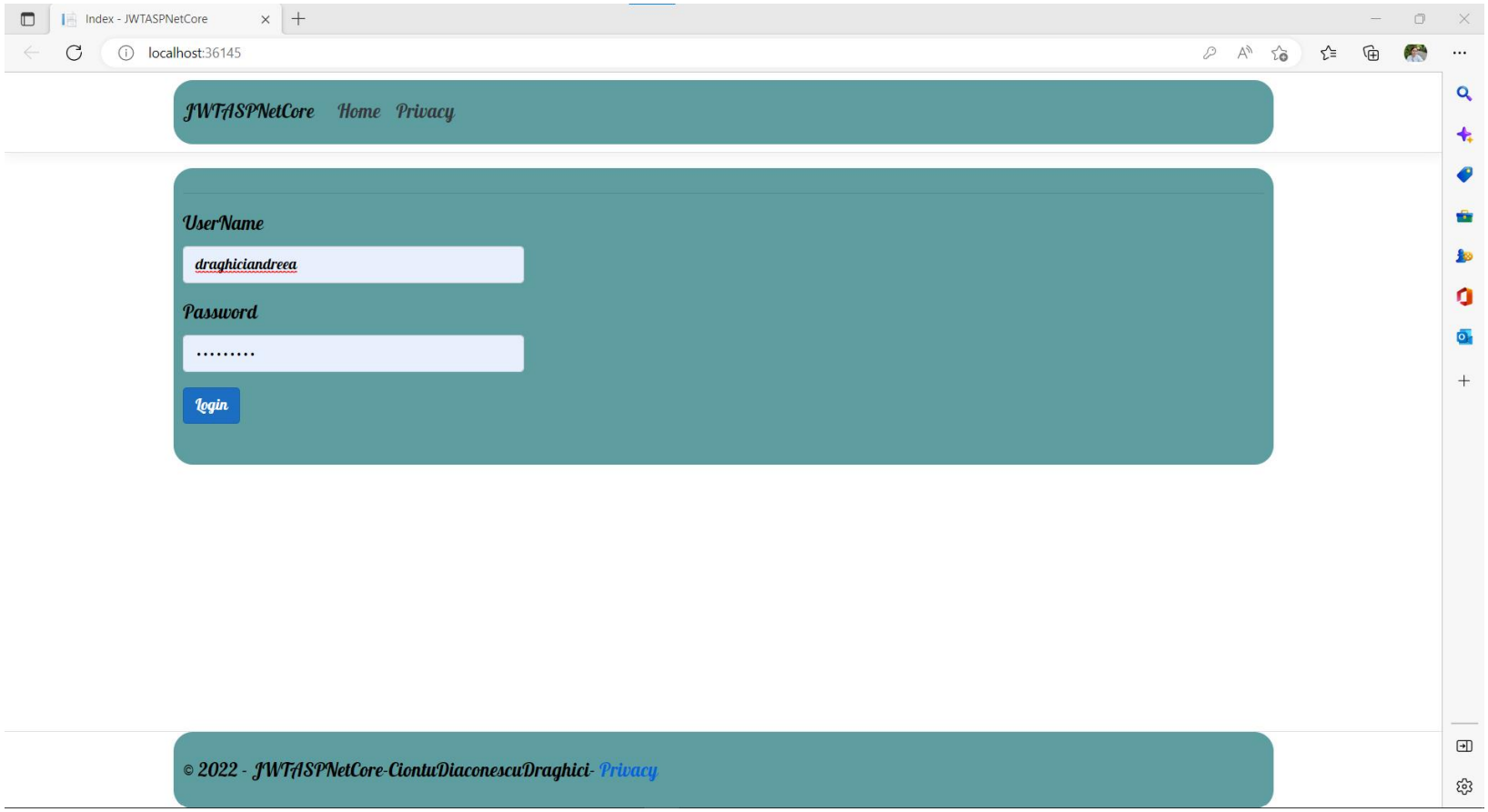
In the HomeController class, we will take advantage of dependency injection to be able to use instances of the Configuration, TokenService and UserRepository classes.

```
namespace JWTASPNetCore.Controllers
{
    1 reference
    public class HomeController : Controller
    {
        //read-only instances for each of the three interfaces
        private readonly IConfiguration _config;
        private readonly IUserRepository _userRepository;
        private readonly ITokenService _tokenService;
        private string generatedToken = null;

        //constructor injection is used in the HomeController class for each of the instances
        0 references
        public HomeController(IConfiguration config, ITokenService tokenService, IUserRepository userRepository)
        {
            _config = config;
            _tokenService = tokenService;
            _userRepository = userRepository;
        }
        0 references
        public IActionResult Index()
        {
            return View();
        }

        [AllowAnonymous]
        [Route("login")]
        [HttpPost]

        0 references
        public IActionResult Login(UserModel userModel)
        {
            if (string.IsNullOrEmpty(userModel.UserName) || string.IsNullOrEmpty(userModel.Password))
            {
                return (RedirectToAction("Error"));
            }

            IActionResult response = Unauthorized();
            var validUser = GetUser(userModel);

            if (validUser != null)
            {
                generatedToken = _tokenService.BuildToken(_config["Jwt:Key"].ToString(), _config["Jwt:Issuer"].ToString(),
                validUser);

                if (generatedToken != null)
                {
                    HttpContext.Session.SetString("Token", generatedToken);
                    return RedirectToAction("MainWindow");
                }
                else
                {
                    return (RedirectToAction("Error"));
                }
            }
```

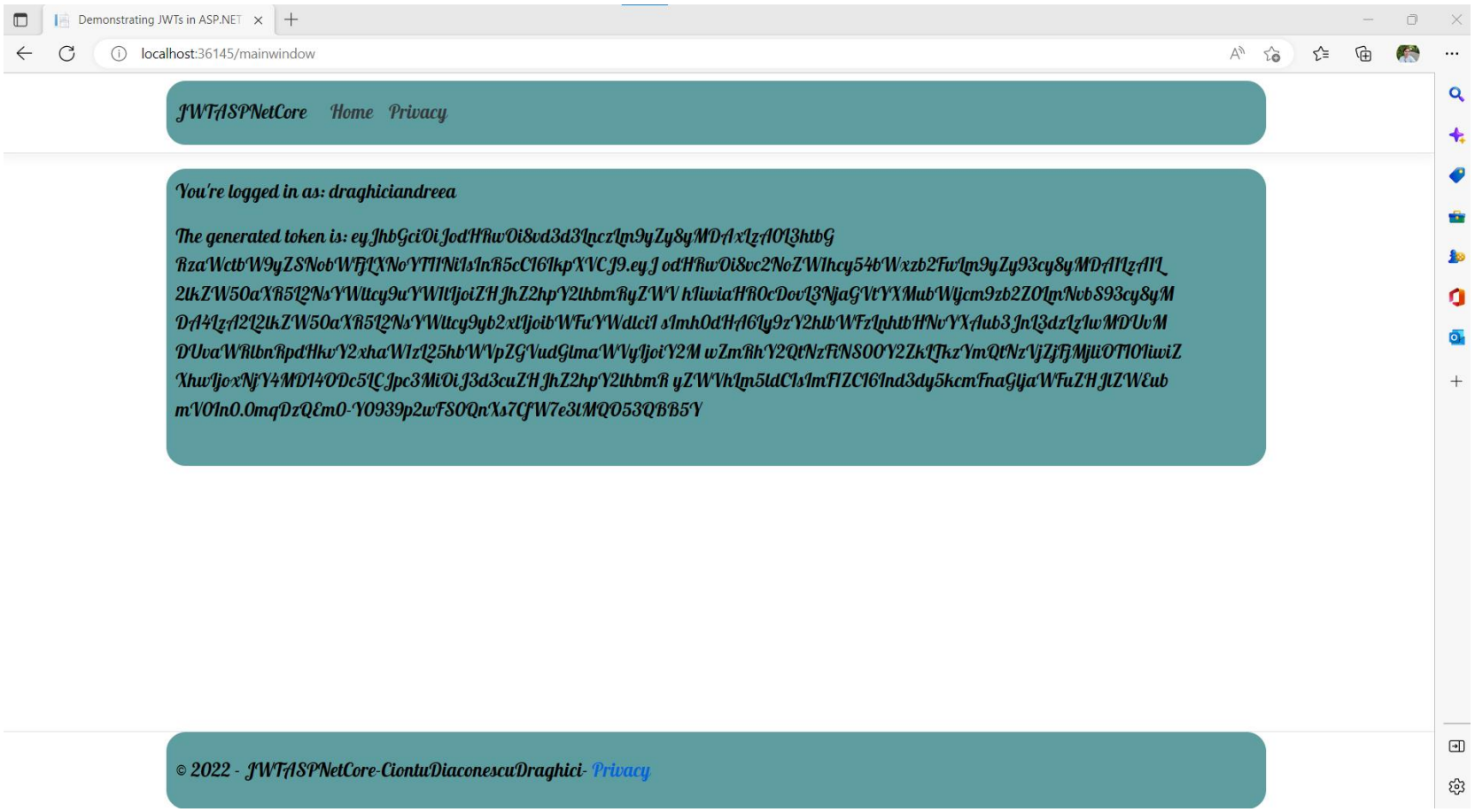
```
            else
            {
                return (RedirectToAction("Error"));
            }
        }
        1 reference
        private UserDTO GetUser(UserModel userModel)
        {
            //Write your code here to authenticate the user
            return _userRepository.GetUser(userModel);
        }
        [Authorize]
        [Route("mainwindow")]
        [HttpGet]
        0 references
        public IActionResult MainWindow()
        {
            string token = HttpContext.Session.GetString("Token");

            if (token == null)
            {
                return (RedirectToAction("Index"));
            }

            if (!_tokenService.IsTokenValid(_config["Jwt:Key"].ToString(),
                _config["Jwt:Issuer"].ToString(), token))
            {
                return (RedirectToAction("Index"));
            }

            ViewBag.Message = BuildMessage(token, 50);
            return View();
        }
        0 references
        public IActionResult Error()
        {
            ViewBag.Message = "An error occured...";
            return View();
        }
        1 reference
        private string BuildMessage(string stringToSplit, int chunkSize)
        {
            var data = Enumerable.Range(0, stringToSplit.Length / chunkSize)
                .Select(i => stringToSplit.Substring(i * chunkSize, chunkSize));

            string result = "The generated token is:";

            foreach (string str in data)
            {
                result += Environment.NewLine + str;
            }
            return result;
        }
    }
}
```

Run the Application

The GetUser method of the HomeController class calls the GetUser method of the UserRepository class to retrieve an instance of the UserDTO class based on the user credentials entered in the Login screen shown:



Once you specify the user's credentials and click on Login, you'll be redirected to another Web page that shows the name of the logged in user and the generated token, as shown:

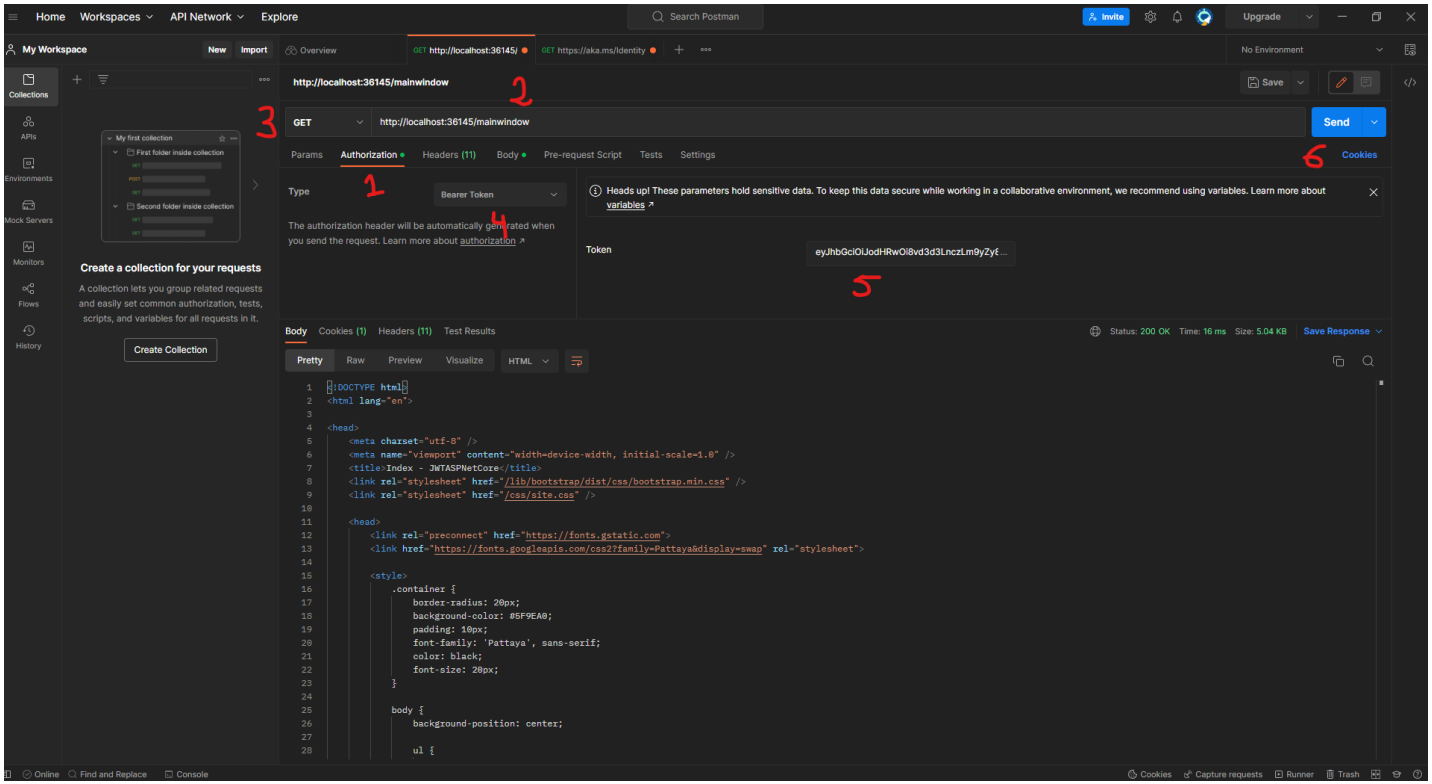


Test the API endpoint in Postman with Token

Postman is an API platform for developers to design, build, test and iterate their APIs.

Add a new Get request in postman and add the JWT token Authorization Tab -> Select Bearer -> Insert token and click on send button to test the authorization with given token.

Steps are presented in the next figure:



Resources

- https://blog.logrocket.com/jwt-authentication-best-practices/
- https://jwt.io/introduction
- https://learning.postman.com/docs/sending-requests/authorization/