.NET Core -ACCENTURE- 26MAY21

Avisos

ACTIVIDAD 6 - IMPLEMENTAR LA SEGURIDAD

Su progreso

email task6

Video Actividad 6 completo

Implementar el inicio de sesión

La mayoría de las aplicaciones requieren de restricciones que permitan determinar quiénes pueden usar el sistema y qué cosas pueden hacer en el mismo, esto se conoce como **autenticación** y **autorización**. La autenticación sucede primero e involucra la interacción entre el navegador y el servidor web, cuando el navegador envía una petición para acceder a algún recurso protegido el servidor web retorna un mensaje indicando que se debe iniciar sesión, éste mensaje también indica cómo se debe proceder para iniciar sesión. Si el inicio de sesión tiene exito entonces el servidor web sabrá quién es el usuario, puede entonces buscar las reglas que indican qué puede hacer el usuario o con qué recursos puede interactuar. Incluso si un usuario ha iniciado sesión puede que no tenga permisos para acceder al recurso solicitado.

En esta actividad se creará un método para iniciar sesión basado en email y contraseña, para ello se deberá:

- modificar la entidad Player para incluir la contraseña
- agregar un controlador para manejar el inicio y cierre de sesión
- agregar un controlador para permitir crear nuevos Players

Modificar la entidad Player para soportar el inicio de sesión

Se debe moficar la clase Player para agregar un nuevo campo llamado Password de tipo string. Ya que se modificó una entidad del modelo de datos se deberá agregar un migration para que los cambios sean aplicados a la tabla en la base de datos. Para ello se ejecutará el comando Add-Migration y el nombre del mismo será addPasswordToPlayerEntity.

```
Add-Migration addPasswordToPlayerEntity
```

Se podrá notar que el migration recientemente creado solo contiene el cambio de la creación del campo password en la entidad Player. Luego ejecutar el comando Update-Database para que se apliquen los cambios en la base de datos. Se debe verificar que en la base de datos la tabla Players contenga la nueva columna.

Cuando la aplicación reciba una petición de inicio de sesión se deberá buscar la información del Player en la base de datos (en la aplicación los usuarios son los jugadores), es por ello que se debe crear el repositorio de Player en la carpeta repositories IPlayerRepository y su implementación PlayerRepository, guíate de los casos anteriores para crearlos. La diferencia de éste repositorio es que solo tendremos un método llamado findByEmail(string email), el cual recibe un parámetro que será el email a buscar en la base de datos. Este método deberá llamar a uno de los métodos de RepositoryBase así que revísa sus métodos y piensa cuál debe ser, recuerda que se requiere buscar un solo Player dado un email. Cuando encuentres el método te darás cuenta que se le debe pasar por parámetro una expresión, ésto quiere decir que se debe indicar una función de comparación que le dirá al entity framework cómo filtrar los registros de la tabla Players en la base de datos, la misma se puede indicar con una expresión lambda, puedes leer sobre ello en https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/statements-expressions-operators/lambda-expressions, dicha expresión quedaría:

(player => player.Email == email)

La expresión indica que lo que reciba se llamará player y retornará el resultado de comparar el campo Email de ese objeto que recibió contra el parámetro email. Esto le indica el entity framework que deberá armar una consulta en la base de datos que recorra todas las filas de la tabla Players preguntando fila por fila si el valor de la columna Email es igual al indicado. Como ésto se encuentra abstraido no lo vemos pero entity framework ejecuta en la base de datos la consulta:

*select * from Players where email = 'j.bauer@ctu.gov' (el email variaría)*

Copia ésta consulta y ve al SQLManagement Studio pulsa sobre el botón new Query o Nueva Consulta en la barra de herramientas y pégalo en el editor de texto, luego pulsa el botón Execute o Ejecutar y podrás notar que en el resultado se encuentra un solo registro con la información solicitada.

Nota: en éste punto se deberá usar FirstOrDefault() de nuevo, https://docs.microsoft.com/en-us/dotnet/api/system.linq.enumerable.firstordefault?view=netcore-2.2

Importante: no olvidar agregar el repositorio en el DI container (clase Startup)

Ahora se debe modificar el DTO de player PlayerDTO para que contenga el nuevo campo password. Además actualizaremos los registros de players en la base de datos para agregarles contraseñas. Ve al MSSql Managemente Studio y en la tabla Players pulsa con el boton derecho del mouse en la opción editar 200 filas, utiliza el archivo Salvo test database para ver los valores del campo password por cada usuario, al ingresar el valor y cambiar de fila ya se guardan los datos.

Antecedentes

Cuando se configuran los frameworks usualmente se emplea código que se repite en muchos proyectos ya que usualmente éstas configuraciones se hacen de la misma manera en cada uno de ellos, ésto se le conoce como "boilerplate code" https://www.freecodecamp.org/news/whats-boilerplate-and-why-do-we-use-it-let-s-check-out-the-coding-style-guide-ac2b6c814ee7/ es por ello que ésta practica indicará el paso a paso de lo que se debe realizar.

Configurar el inicio de sesión

.Net Core tiene varias maneras de manejar la autenticación como con su módulo Identity así como también autenticación de windows o el uso de servicios como cuentas de google, facebook, azure, twitter, entre otros... puedes leer sobre ello en https://docs.microsoft.com/en-us/aspnet/core/security/?view=aspnetcore-2.2. Para la aplicación se utilizará el método más simple que es usar una cookie de autenticación, las cookies son archivos que se crean en el navegador para almacenar información, éstas viajan con las peticiones que realice el mismo por lo que se utilizan comúnmente para recordar información del usuario como preferencias, sesiones activas, entre otras, puedes leer sobre las cookies en https://es.wikipedia.org/wiki/Cookie_(inform%C3%A1tica).

Siguiendo la documentación encontrada en https://docs.microsoft.com/en-us/aspnet/core/security/authentication/cookie?view=aspnetcore-2.2 lo primero que se debe hacer es crear el servicio middleware de autenticación en la clase Startup:

```
services.AddAuthentication(CookieAuthenticationDefaults.AuthenticationScheme)
  .AddCookie(options =>
  {
     options.ExpireTimeSpan = TimeSpan.FromMinutes(10);
     options.LoginPath = new PathString("/index.html");
  });
```

Se puede notar que además se está indicando el tiempo de expiración de la cookie y a qué ruta debe redirigir el servidor cuando no existe una sesión.

Luego en una línea antes de *app.UseMvc();* se debe indicar que la aplicación tendrá autenticación:

```
app.UseAuthentication();
```

Implementar el controlador de autenticación

Ahora se necesita un controlador que escuche la petición de inició de sesión para crear la cookie, en la carpeta controllers se debe crear un nuevo controlador llamado AuthController, la ruta del controlador se debe modificar a "api/auth", recuerda agregar el constructor con la inyección de IPlayerRepository. Seguidamente se debe modificar el método llamado Post para que quede de la siguiente forma:

```
// POST: api/Auth/login
[HttpPost("login")]
public async Task<IActionResult> Login([FromBody] PlayerDTO player)
{
      try
    {
      Player user = _repository.FindByEmail(player.Email);
      if (user == null || !String.Equals(user.Password, player.Password))
         return Unauthorized();

      var claims = new List<Claim>
         {
            new Claim("Player", user.Email)
         };

      var claimsIdentity = new ClaimsIdentity(
         claims, CookieAuthenticationDefaults.AuthenticationScheme);

      await HttpContext.SignInAsync(
            CookieAuthenticationDefaults.AuthenticationScheme,
            new ClaimsPrincipal(claimsIdentity));

      return Ok();
    }
    catch (Exception ex)
    {
      return StatusCode(500, "Internal server error");
    }
}
```

Hay varios conceptos nuevos en éste punto, repasaremos uno por uno. Hasta ahora en los controladores solo se han manejado peticiones HTTP de tipo GET (get es para obtener datos) que no se indica explicitamente ya que es la opción por defecto pero existen otros tipos de peticiones como se mencionó con anterioridad (POST, PUT, DELETE, PATH) cada uno de llos se conoce como métodos http https://docs.microsoft.com/en-us/dotnet/api/system.net.http.httpmethod?view=netcore-2.2, con el modificador [HttpPost] indica que el método Login() se ejecutará cuando al servidor le llegue una petición HTTP de tipo Post en la ruta "api/auth/login", el modificador [FromBody] indica que el valor del parámetro player de tipo PlayerDTO será tomado del cuerpo de la petición HTTP (en el cuerpo de la petición se encontrará una representación JSON de un PlayerDTO), verás... la petición se compone de varios elementos, ya vimos que uno es el método otro es el cuerpo o la data que transaporta la misma, también está la dirección del servidor o domino, cabeceras, tamaño en bytes de la petición, entre otros. Puedes leer sobre todos los componentes de una petición HTTP en https://docs.microsoft.com/en-us/dotnet/api/microsoft.aspnetcore.http.httprequest?view=aspnetcore-2.2 revisa la sección properties.

Seguidamente encontramos *async Task<IActionResult>*, lo que ésto indica es el tipo de retorno de la función Login(), Retornará una tarea (Task) de tipo IActionResult la tarea es una manera que tiene el framework de .Net para representar trabajos que se pueden ejecutar de manera asíncrona https://docs.microsoft.com/en-us/dotnet/api/system.threading.tasks?view=netcore-2.2 mientras que IActionResult es una interfaz que representa el resultado de ejecutar un método de accción https://docs.microsoft.com/en-us/dotnet/api/microsoft.aspnetcore.mvc.iactionresult?view=aspnetcore-2.2, una de sus implementaciones es https://docs.microsoft.com/en-us/dotnet/api/microsoft.aspnetcore.mvc.actionresult?view=aspnetcore-2.2 fijate en el listado de derivados donde podrás ver todos los tipos de resultados. Otra palabra clave es async que indica que la ejecución del método Login() será asíncrono de manera que quién haga la petición a éste método deberá seguir con la ejecución normal de su programa mientras espera por el resultado o respuesta de la misma. La ejecución de un proceso sincrónico indica que al ejecutarse la operación, por ejemplo un llamado a una función, el programa debe esperar por el resultado para seguir con su ejecución mientras que si se ejecuta un proceso asincrono el programa ejecuta la función e indica que cuando termine la ejecución de esa función se llame a otra función que procesará el resultado de la misma, seguidamente sigue con su ejecución normal. Es recomendable leer https://docs.microsoft.com/es-es/dotnet/standard/async-in-depth para entender más sobre el tema. Se debe colocar async ya que

en el cuerpo de la función existe una instrucción *await HttpContext.SignInAsync(),* await se explica en el enlace anterior.

Si el usuario que se recibe en la petición no se encuentra en la base de datos o si su contraseña no coincide el método login retornará un código de estado 401 Forbidden, los códigos de estado los utiliza el servidor para indicar qué sucedió con una petición y están clasificados por rangos:

- 1xx, información: indican información sobre el estado de la petición
- 2xx, exito: indican que la petición ha sido bien recibida, aceptada y procesada
- 3xx, redirección: indican que se deben realizar otras acciones para completar la solicitud
- 4xx, errores de cliente: indican que la petición contiene errores o no puede ser procesada.
- 5xx, errores del servidor, indican que el servidor falló en procesar una petición aparentemente válida.

La clase BaseController de .Net Core nos permite crear éstos estados de manera directa teniendo un método para cada tipo https://docs.microsoft.com/en-us/dotnet/api/microsoft.aspnetcore.mvc.controllerbase?view=aspnetcore-2.2 revisa la sección métodos, de igual manera se puede construir un estado con un mensaje personalizado utilizando el método *StatusCode()* como se muestra en la instrucción *StatusCode(500, "Internal server error").*

Por último se tienen los *Claims* que puedes verlo como: qué dice ser el usuario autenticado o sus pretenciones (si la traducción lo permite), un objeto *ClaimsIdentity* se compone de varios claims o varias pretenciones, no confundir esto con qué puede hacer el usuario,así que entonces como ejemplos de claims tendríamos: Es Empleado, Es Jugador, Tiene Nombre, Tiene Edad, etc.. cada claim tiene un valor: Es Empleado: 0001, Es jugador: 'nick001', Tiene Nombre: 'Jose', Tiene Edad: 23. Con el objeto *ClaimsIndentity* se construye un *ClaimsPrincipal* que contiene varias *ClaimsIdentity* y es el objeto que representa al usuario autenticado, seguidamente se construye la sesión con el método *HttpContext.SignInAsync* pasándole el *ClaimsPrincipal* creado. Lee más en https://docs.microsoft.com/en-us/dotnet/api/system.security.claims.claim?view=netcore-2.2, https://docs.microsoft.com/en-us/dotnet/api/system.security.claims.claimsidentity?view=netcore-2.2, https://docs.microsoft.com/en-us/dotnet/api/system.security.claims.claimsprincipal?view=netcore-2.2

Ya exlicado el método Login() se creará el método de Logout():

```
// POST: api/Auth/logout
[HttpPost("logout")]
public async Task<IActionResult> Logout()
{
    try
    {
        await HttpContext.SignOutAsync(
        CookieAuthenticationDefaults.AuthenticationScheme);
        return Ok();
    }
    catch (Exception ex)
    {
        return StatusCode(500, "Internal server error");
    }
}
```

Los demás métodos del controller deben ser eliminados.

Implementar la configuración de autorización

Una vez completada la autenticación ahora se debe configurar la autorización. Al igual que con la autenticación .Net Core provee de varias maneras de implementar la autorización: basada en roles, por políticas, basada en los recursos a los que se acceden, entre otras. Puedes leer más en https://docs.microsoft.com/en-us/aspnet/core/security/authorization/introduction?view=aspnetcore-2.2 para la aplicación se usará la autorización basada en Claims y ya que los requisitos por Claims se basan en políticas se deberá crear una política para expresar qué Claims debe tener el usuario autenticado, lee más en https://docs.microsoft.com/en-us/aspnet/core/security/authorization/claims?view=aspnetcore-2.2

En Startup así como se agregó a services el middleware AddAuhtentication para la autenticación de igual manera se agregará AddAuthorization para la autorización:

```
services.AddAuthorization(options =>
{
    options.AddPolicy("PlayerOnly", policy => policy.RequireClaim("Player"));
});
```

La política indica que se llamará PlayerOnly y que para acceder a los recursos con esa política los usuarios autenticados deberán tener el Claim Player dicho de otra manera deberán ser Player. Ahora se debe indicar qué recursos debe tener esa política, ése recurso es el acceso a /api/gamePlayers/{id} por lo que se agregará el modificador [Authorize] a los controladores o a los métodos de los controladores que se requieran, dependiendo de si se quiere restringir el acceso a todo el controlador o a un método en específico. Como se mencionó previamente el controlador GamePlayersConstroller (donde reside la ruta /api/gamePlayers/{id}) se restringirá con la política recientemente creada:

```
[Route("api/gamePlayers")]
[ApiController]
[Authorize("PlayerOnly")]
public class GamePlayersController : ControllerBase {...}
```

De igual manera se hará con el controlador GamesController:

```
[Route("api/games")]
[ApiController]
[Authorize]
public class GamesController : ControllerBase {...}
```

Con la diferencia de que éste último debe permitir que cualquier usuario pueda ver el listado de juegos, por ello se agrega el modificador [AllowAnonymous] al método Get:

```
// GET: api/Games
[HttpGet]
[AllowAnonymous]
public IActionResult Get(){...}
```

Lee más sobre [Authorization] en *https://docs.microsoft.com/en-us/aspnet/core/security/authorization/simple?view=aspnetcore-2.2*

Listo!, ahora se debe probar la aplicación, se ejecuta y al iniciar ir a la url: https://localhost:5001/index.html, luego pulsar la tecla F12 (abrir herramientas de desarrollador) https://developers.google.com/web/tools/chrome-devtools/open para chrome y https://developer.mozilla.org/en-US/docs/Tools para firefox. Intentar ingresar a la url: https://localhost:5001/api/gamePlayers/1 para verificar que el navegador redirija al index.html ya que no se encuentra autenticado, luego en la consola de javascript copiar y pegar:

```
axios.post("/api/auth/login",{email: 'j.bauer@ctu.gov', password: '55'})
    .then(response=>{console.log("Sesion iniciada!!")})
    .catch(error => {console.log("error, código de estatus: " + error.response.status)});
```

Luego pulsar la tecla enter y se debe mostrar el mensaje "error, código de estatus: 401" ya que la contraseña suministrada no es correcta, ahora se debe cambiar la contraseña por la correcta y repetir el proceso:

```
axios.post("/api/auth/login",{email: 'j.bauer@ctu.gov', password: '24'})
    .then(response=>{console.log("Sesion iniciada!!")})
    .catch(error => {console.log("error, código de estatus: " + error.response.status)});
```

Se debe mostrar el mensaje "Sesion iniciada!!", ir de nuevo a la url: https://localhost:5001/api/gamePlayers/1 y se debe mostrar el JSON con la vista del juego. Por último ejecutar en la consola:

```
axios.post("/api/auth/logout")
    .then(response=>{console.log("Sesion cerrada!!")})
    .catch(error => {console.log("error, código de estatus: " + error.response.status)});
```

Se debe mostrar el mensaje "Sesion cerrada!!", verificar que al ir a la url: https://localhost:5001/api/gamePlayers/1 vuelva a redirigir a index.html.

Importante: axios es una librería de javascript para hacer peticiones http la cual está incorporada en el archivo index.html por ello las instrucciones deben ser ejecutadas estando en la ruta https://localhost:5001/index.html, de lo contrario el navegador indicará que no se encontró la misma.

Implementar la seguridad - Envía el proyecto en un .zip

web-listar-juegos-3

Agregar el formulario de inicio de sesión

Descarga el paquete web-listar-juegos-3 y copiar su contenido en la carpeta wwwroot. Una vez copiado el contenido en la carpeta ejecutar el proyecto y colocar en el navegador la url *https://localhost:5001/index.html* se debe ver la página principal del juego con el formulario de inició de sesión, pureba iniciar sesión, indicar credenciales inválidas y cerrar sesión.

Agregar la información del usuario autenticado al listado de Games

Se debe modificar el listado que retorna el controller GamesController para que además de los juegos indique el mail del usuario autenticado, de no haber un usuario autenticado entonces el valor será "Guest". Quedando el JSON:

```
{
    "email": "Guest"
    ,"games": [{
        "id":1
        ,"creationDate":"2019-01-01T00:23:39.8783606"
        ,"gamePlayers":[{
            "id":1,"joinDate":"2019-01-01T00:26:22.6004604"
            ,"player":{"id":1,"email":"j.bauer@ctu.gov","password":null},"point":1.0}
        }]
    }]
}

//con usuario autenticado

{
    "email": "j.bauer@ctu.gov"
    ,"games": [{
        "id":1
        ,"creationDate":"2019-01-01T00:23:39.8783606"
        ,"gamePlayers":[{
            "id":1,"joinDate":"2019-01-01T00:26:22.6004604"
            ,"player":{"id":1,"email":"j.bauer@ctu.gov","password":null},"point":1.0}
        }]
    }]
}
```

Se puede crear un nuevo DTO que contenga los dos campos necesarios o se puede crear directamente el nuevo objeto, un ejemplo para crear un objeto anonimo sería:

```
var listado = new {
    nombre = "JOSE",
    edad = 30
}
```

Lee más sobre los objetos anonimos en https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/classes-and-structs/anonymous-types de cualquier manera servirá.

Para obtener la información del usuario autenticado, específicamente cómo obtener el valor de un Claim se deberá revisar ésta publicación en stackoverflow: https://stackoverflow.com/questions/36641338/how-get-current-user-in-asp-net-core la idea es obtener el Claim Player.

Una vez modificado el controlador se debe ejecutar la aplicación e ir a la url: https://localhost:5001/api/games para verificar que el JSON tenga el valor "Guest" en el campo email ya que no se ha iniciado sesión, luego ejecutar la instrucción de login de la actividad anterior (recordar estar

en index.html) y realizar de nuevo la petición para obtener los juegos, se debe ver el email del usuario autenticado en el campo email.

Implementar el método de creación de Players

Crear el controlador PlayersController y utilizar el método [HttpPost] para recibir una entidad de tipo PlayerDTO y crear un nuevo player en la base de datos. El método deberá verificar si ya existe el usuario en la base de datos, si es así deberá retornar un estado 403 (Prohibido) con un mensaje que indique el motivo, ejemplo: "Email en uso". De igual manera debe verificar que los campos no se encuentren vacíos de lo contrario retornando un codigo de estado 403 con el mensaje "Datos inválidos". **Nota:** al crear el controlado cambiar el modificador del método Get al nombre [HttpGet("{id}", Name = "GetPlayer")],

Si no hay problemas el método deberá guardar el player en la base de datos y retornar un estado 201 (creado).

Nota: no guardar el objeto PlayerDTO, se debe crear una nueva instancia de Player con los datos del DTO recibido.

Importante: se deberá modificar la interfaz IPlayerRepository y la clase PlayerRepository para implementar el método Create() de BaseRepository:

```
//IPlayerRepository
void Save(Player player);

//PlayerRepository
public void Save(Player player)
{
    Create(player);
    SaveChanges();
}
```

Además de modificar IBaseRepository e implementar en BaseRepository el método SaveChanges():

```
//IRepositoryBase
void SaveChanges();

//RepositoryBase
public void SaveChanges()
{
    this.RepositoryContext.SaveChanges();
}
```

En el controlador PlayersController se deberá tener en algún punto la instrucción:

```
_repository.Create(newPlayer);
```

Siendo _repository una instancia de PlayerRepository y newPlayer una instancia del Player a crear.

Cuando se termine de modificar el controlador se debe iniciar la aplicación y ejecutar la siguiente instrucción en la consola de javascript:

```
axios.post("https://localhost:5001/api/players",{email: 'j.bauer@ctu.gov', password: 'jose'})
.then(response=>{console.log("Usuario creado!!")})
.catch(error => {console.log("error, código de estatus: " + error.response.status + " mensaje: " +
error.response.data)});
```

Se debe mostrar el mensaje "error, código de estatus: 403 mensaje: email en uso". O el mensaje que se haya indicado. Por último se ejecuta:

```
axios.post("https://localhost:5001/api/players",{email: 'jose@gmail.com', password: 'jose'})
.then(response=>{console.log("Usuario creado!!")}) .catch(error => {console.log("error, código de
estatus: " + error.response.status + " mensaje: " + error.response.data)});
```

Se debe mostrar el mensaje "Usuario creado!!"

Agregar la información del usuario - Envía el proyecto en un .zip

web-listar-juegos-4

Agregar el botón de registro de usuario

Descarga el paquete web-listar-juegos-4 y copiar su contenido en la carpeta wwwroot. Una vez copiado el contenido en la carpeta ejecutar el proyecto y colocar en el navegador la url *https://localhost:5001/index.html* se debe ver la página principal del juego con el formulario de inició de sesión y el botón de registro, prueba registrar un nuevo usuario e intentar registrar uno existente.

SECCIÓN ANTERIOR
Actividad 5 - Crear entidades de dominio

SIGUIENTE SECCIÓN
Actividad 7 - Implementar ingresar y crear juegos

Ir a...