UD7 Servicios web.

7.5 Completando el ejemplo.

Una vez que tenemos nuestro servicio operativo, e implementados los mecanismos de autenticación y autorización, el siguiente paso sería que cada usuario trabaje con su propia lista de tareas. De este modo el servicio estaría completamente operativo y disponible para todos los usuarios registrados. Vamos a ver lo sencillo que va a resultar realizar esta tarea.

Modificación del modelo de datos.

Lo primero que tenemos que hacer es asociar las tareas con el usuario al que pertenecen. En nuestro ejemplo hemos utilizado el email tanto como nombre de usuario como id del mismo, por lo que necesitamos añadir una propiedad de tipo cadena en las tareas para almacenar dicho id.

Nuestro modelo para las tareas quedaría entonces de este modo:

```
public class TodoItem
{
    public long Id { get; set; }
    public string Name { get; set; }
    public bool IsComplete { get; set; }
    public string UserId { get; set; }
}
```

En este punto nos encontramos con una dificultad. La herramienta de migraciones nos ayuda en innumerables ocasiones, pero en este caso debemos tener en cuenta que si nuestra bd contiene actualmente tareas, todas estas están sin asociar a ningún usuario. Cuando realicemos la migración se añadirá el nuevo campo a la tabla, pero no se introducirá ningún valor a dicho campo. Esta tarea debería ser realizada por nosotros.

Una vez tenemos el modelo modificado, creamos una nueva migración y actualizamos la base de datos.

```
add-migration TodoItemsRelatedToUsers
update-database
```

Modificación de métodos de la API.

Lo único que nos queda por hacer es modificar los métodos de nuestra API, para que cuando se trabaje con nuestras tareas, se incluya la información relativa al usuario en las mismas.

Para ello, cuando el cliente llame a uno de nuestros métodos, lo primero que necesitaremos es averiguar la identidad del usuario. Esta información, como ya hemos explicado, se encuentra dentro de la colección claims de nuestro token, por lo que ya ha sido pasada a nuestro servicio.

1. Comenzamos creando un método auxiliar que nos devuelve el id del usuario, obteniendo este del token de autorización recibido. La información contenida en dicho token es accesible a través de objeto User.

```
private string GetUserId()
{
    return User.Claims.First(c => c.Type == ClaimTypes.NameIdentifier).Value;
}
```

Lo que estamos haciendo es buscar, en la colección de claims que contiene nuestro token, aquel que contiene el identificador del usuario.

2. Con la información del usuario el resto de modificaciones son triviales, comenzamos por el método que obtiene las listas de tareas, que en este caso devolverá sólo las del usuario autenticado.

```
using System.ComponentModel.DataAnnotations;

// GET: api/Todo
[HttpGet]
public ActionResult<IEnumerable<TodoItem>> GetTodoItems()
{
    return _context.TodoItems
        .Where(item => item.UserId.Equals(GetUserId()))
        .ToList();
}
```

Simplemente añadimos una condición, para que la lista de tareas devueltas sean sólo aquellas cuyo id de usuario coincide con el usuario que hace la petición.

3. Para el método que devuelve la información de una tarea concreta, añadimos una comprobación de que esa tarea pertenece al usuario que la solicita, en caso contrario devolvemos un error de autorización.

```
// GET: api/Todo/5
[HttpGet("{id}")]
public ActionResult<TodoItem> GetTodoItem(long id)
{
    var todoItem = _context.TodoItems.Find(id);

    if (todoItem == null)
    {
        return NotFound();
    }

    if (!todoItem.UserId.Equals(GetUserId()))
    {
        return Unauthorized();
    }

    return todoItem;
}
```

4. Para el resto de métodos, al recibir como parámetro un objeto *TodoItem*, la información del id de usuario debe estar incluida en la petición, por lo que podríamos dejarlos como están. Sin embargo, podría darse el caso de que un usuario cree tareas para otro usuario, o un usuario inexistente, por lo que para evitar problemas de consistencia, añadiremos en todos estos métodos una comprobación adicional de que el usuario que hace la petición es el mismo que figura en la información de las tareas que envía.
   A continuación se muestran las modificaciones que tenemos que realizar:

```
// POST: api/Todo
[HttpPost]
public ActionResult<TodoItem> PostTodoItem(TodoItem item)
{
    if (!item.UserId.Equals(GetUserId()))
    {
        return Unauthorized();
    }

    _context.TodoItems.Add(item);
    _context.SaveChanges();

    return CreatedAtAction(nameof(GetTodoItem), new { id = item.Id }, item);
}

// PUT: api/Todo/5
[HttpPut("{id}")]
public  IActionResult PutTodoItem(long id, TodoItem item)
{
    if (id != item.Id)
    {
        return BadRequest();
    }

    if (!item.UserId.Equals(GetUserId()))
    {
        return Unauthorized();
    }

    _context.Entry(item).State = EntityState.Modified;
    _context.SaveChanges();

    return NoContent();
}

// DELETE: api/Todo/5
[HttpDelete("{id}")]
public IActionResult DeleteTodoItem(long id)
{
    var todoItem = _context.TodoItems.Find(id);

    if (todoItem == null)
    {
        return NotFound();
    }

    if (!todoItem.UserId.Equals(GetUserId()))
    {
        return Unauthorized();
    }

    _context.TodoItems.Remove(todoItem);
    _context.SaveChanges();

    return NoContent();
}
```

Con esto habremos terminado nuestro simple servicio web de lista de tareas.