Adding User Authentication (Using Identity)

What Is Authentication

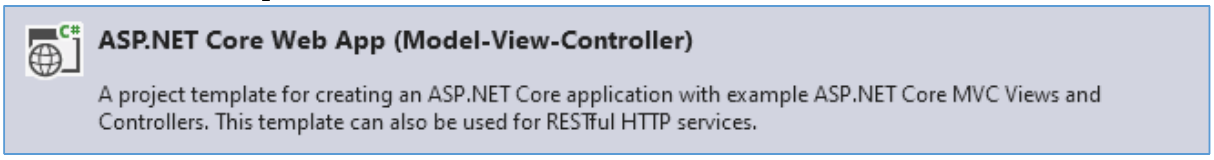
Authentication is a process in which a user provides credentials (such as a username and password) in order to gain access to an application. The credentials provided are compared to those stored in a database (or other resource), and if they match, the user is allowed to log in.

Authorization refers to the actions the user is allowed to perform (once they have logged in).

ASP.NET Core Identity is an API that supports user interface login features. It manages user names, passwords, profile data, roles, e-mail confirmation, etc.Users can create an account with basic login information (or use an external login provider such as Facebook, Google or Twitter).

Creating A .NET Web Application With Authentication

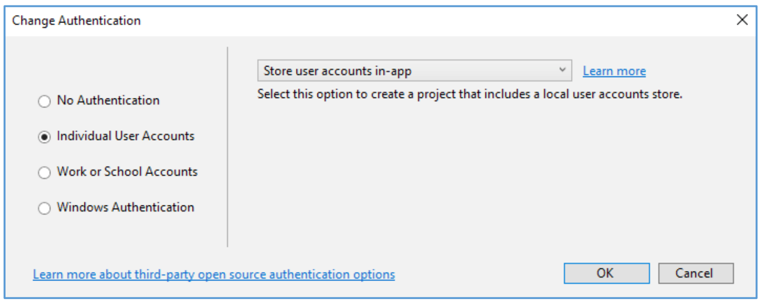
Create a new project and choose ASP.NET Core web application.
Select the MVC template, and look at the Authentication section.



ASP.NET Core Web App (Model-View-Controller)

A project template for creating an ASP.NET Core application with example ASP.NET Core MVC Views and Controllers. This template can also be used for RESTful HTTP services.

Click on the "Change Authentication" link under Authentication
Choose "Individual User Accounts and click "OK"



Change Authentication

No Authentication [✓ Individual User Accounts] Work or School Accounts Windows Authentication

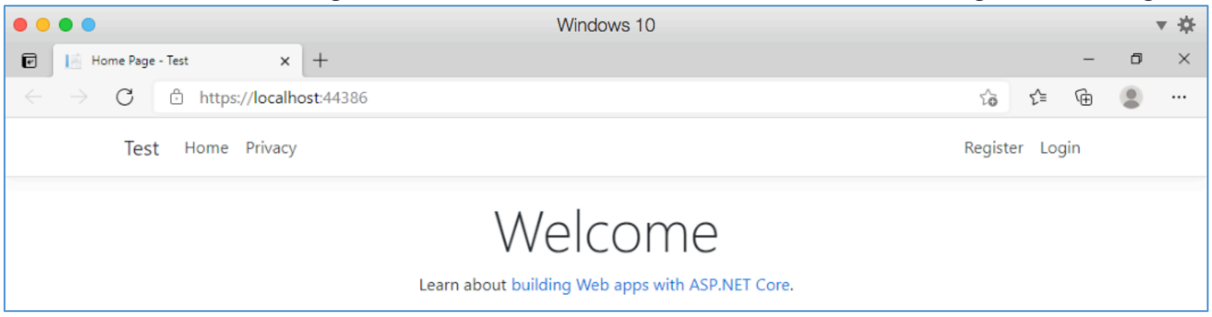
Store user accounts in-app | Learn more

Select this option to create a project that includes a local user accounts store.

Learn more about third-party open source authentication options

OK | Cancel

Run your application.
You will see that the navigation bar now contains two additional items – Register and Login



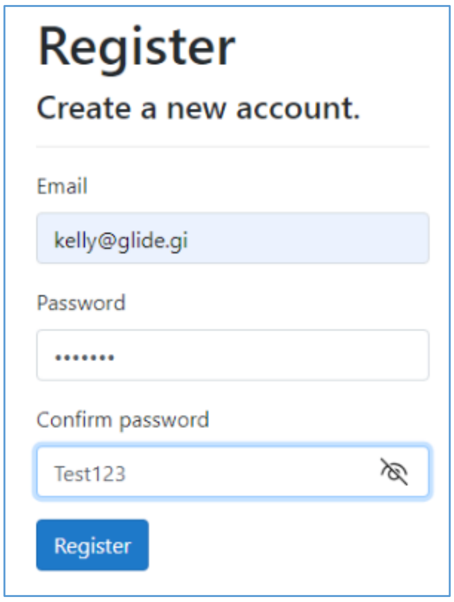
Welcome

Learn about building Web apps with ASP.NET Core.

Register an account.

You will see that it does not work yet because some migrations are pending.
The migration that is pending is what is required to create the database tables to store users.

Open the Package Manager Console and apply the Update-Database command.



Register

Create a new account.

Email: kelly@glide.gi

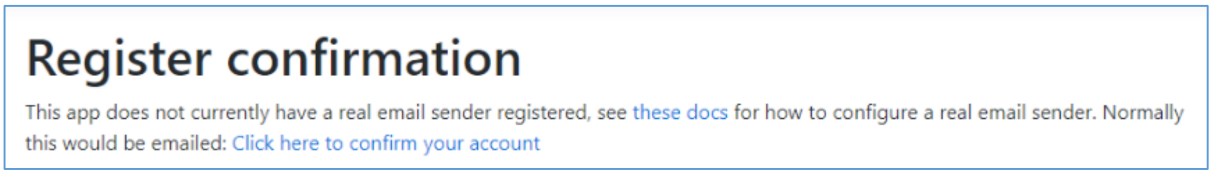
Password: •••••••

Confirm password: Test123

Then run your application and register an account.

You must enter a valid e-mail address and a password which meets the requirements
The validation is already set up for you.

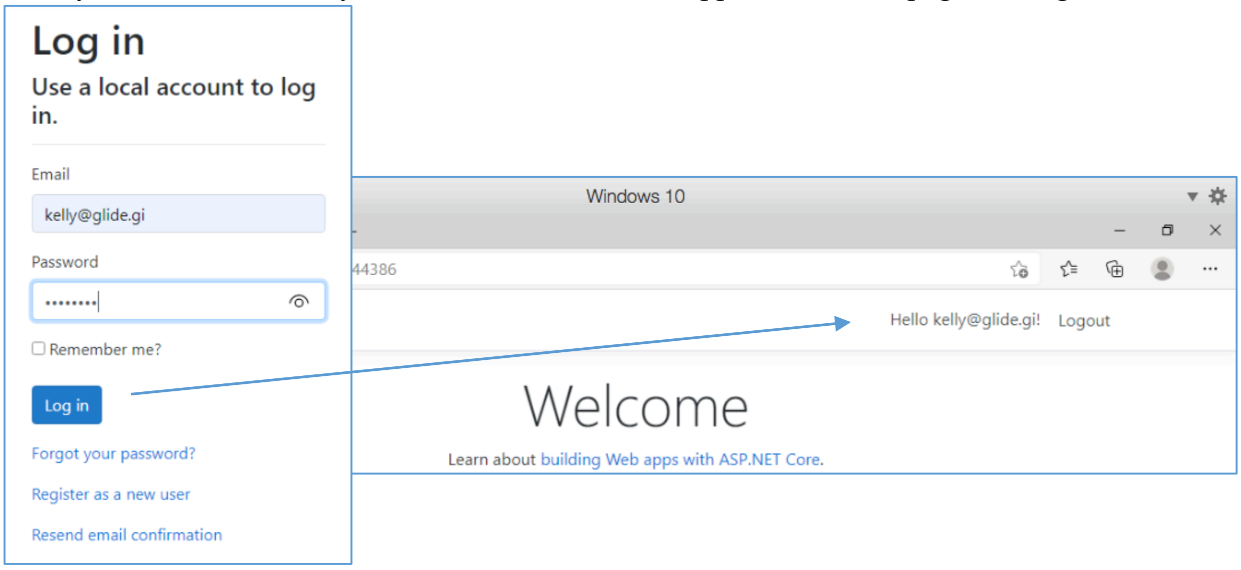
Complete the steps to mimic the confirmation of the account by clicking on the link provided.



Register confirmation

This app does not currently have a real email sender registered, see these docs for how to configure a real email sender. Normally this would be emailed: Click here to confirm your account

Once you have confirmed your account, return to the application homepage and log in.



Log in

Use a local account to log in.

Email: kelly@glide.gi

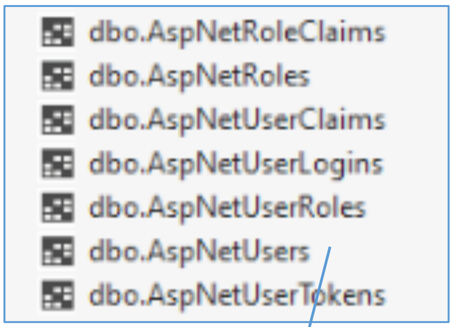
Password: ••••••••

☐ Remember me?

Welcome

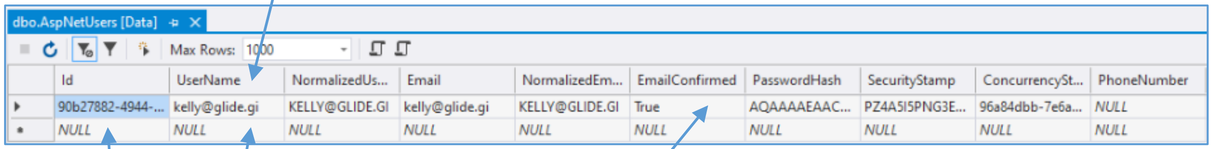
Learn about building Web apps with ASP.NET Core.

Inside The Database


dbo.AspNetRoleClaims
dbo.AspNetRoles
dbo.AspNetUserClaims
dbo.AspNetUserLogins
dbo.AspNetUserRoles
dbo.AspNetUsers
dbo.AspNetUserTokens

If you look inside the database, you will see a set of tables that have been created for you.

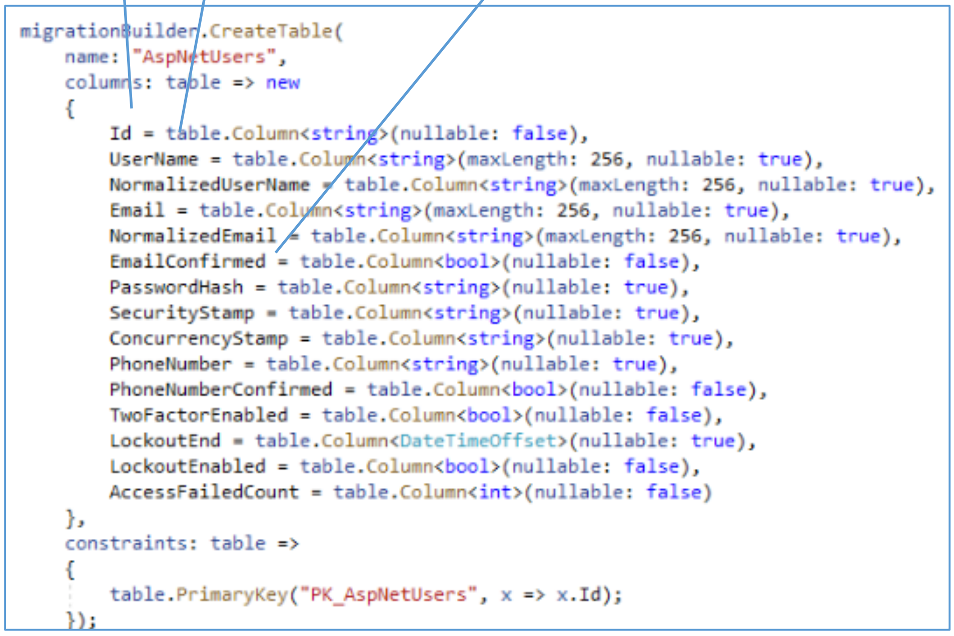
Take a look inside the Users table (right click to see the menu, and click “View Data”) and you should see the details of the user you successfully registered.



	Id	UserName	NormalizedUs...	Email	NormalizedEm...	EmailConfirmed	PasswordHash	SecurityStamp	ConcurrencySt...	PhoneNumber
▸	90b27882-4944-...	kelly@glide.gi	KELLY@GLIDE.GI	kelly@glide.gi	KELLY@GLIDE.GI	True	AQAAAAEAAC...	PZ4A5I5PNG3E...	96a84dbb-7e6a...	*NULL*
*	*NULL*	*NULL*	*NULL*	*NULL*	*NULL*	*NULL*	*NULL*	*NULL*	*NULL*	*NULL*

Migrations

You can also take a look at the Migration that needed to be run in order for these tables to be created. The fields that are generated automatically for you are in this migration file.



```
migrationBuilder.CreateTable(
    name: "AspNetUsers",
    columns: table => new
    {
        Id = table.Column<string>(nullable: false),
        UserName = table.Column<string>(maxLength: 256, nullable: true),
        NormalizedUserName = table.Column<string>(maxLength: 256, nullable: true),
        Email = table.Column<string>(maxLength: 256, nullable: true),
        NormalizedEmail = table.Column<string>(maxLength: 256, nullable: true),
        EmailConfirmed = table.Column<bool>(nullable: false),
        PasswordHash = table.Column<string>(nullable: true),
        SecurityStamp = table.Column<string>(nullable: true),
        ConcurrencyStamp = table.Column<string>(nullable: true),
        PhoneNumber = table.Column<string>(nullable: true),
        PhoneNumberConfirmed = table.Column<bool>(nullable: false),
        TwoFactorEnabled = table.Column<bool>(nullable: false),
        LockoutEnd = table.Column<DateTimeOffset>(nullable: true),
        LockoutEnabled = table.Column<bool>(nullable: false),
        AccessFailedCount = table.Column<int>(nullable: false)
    },
    constraints: table =>
    {
        table.PrimaryKey("PK_AspNetUsers", x => x.Id);
    });
```

Next Steps:

- Understand why and how the password has been hashed (to be stored securely).
-