Hvordan du tilføjer bruger login til et web.API projekt i ASP.Net Core.

Udgangspunktet for denne guide er demoen Data acces i ASP. Fremgangsmåden er at først oprettes en User klasse og så opdateres databasestrukturen. Derefter tilføjes bcrypt, og der laves en AccoutController til brugerregistrering og login. Disse vil i første trin blot returnerer et Userobjekt ved succes. Når så brugeroprettelsen er testet, forsættes med næste trin, som er at genererer en JWT ved succesfuld registrering og oprettelse.

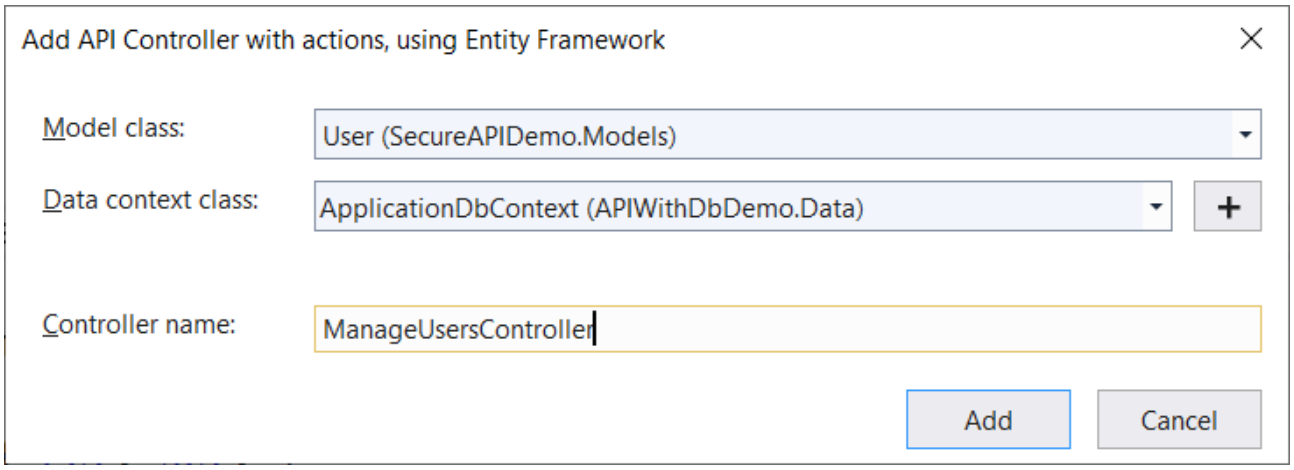
1. Tilføj en user klasse i Models mappen. Denne klasse skal bruges til at kommunikere med databasen og den vil definere hvordan user-tabellen i databasen er struktureret.
   F.eks.:

```
public class User
{
    [Key]
    public long UserId { get; set; }
    [MaxLength(64)]
    public string FirstName { get; set; }
    [MaxLength(32)]
    public string LastName { get; set; }
    [MaxLength(254)]
    public string Email { get; set; }
    [MaxLength(60)]
    public string PwHash { get; set; }
    // You may need a Role property, or a collection of claims
}
```

2. Tilføj en UserDto klasse til mappen Models. Denne klasse skal bruges til at kommunikere med klienten.
   F.eks.:

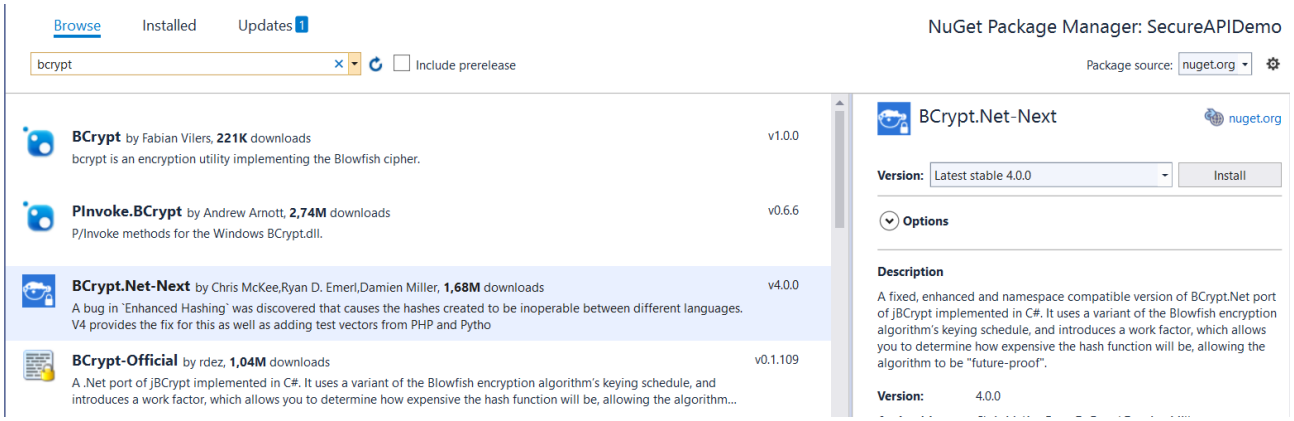
```
public class UserDto
{
    [MaxLength(64)]
    public string FirstName { get; set; }
    [MaxLength(32)]
    public string LastName { get; set; }
    [MaxLength(254)]
    public string Email { get; set; }
    [MaxLength(72)]
    public string Password { get; set; }
}
```

3. Tilføj en kontrollerklasse til at manage users:
   Højreklik på Controllers-mappen og vælg ”add Controller”.
   Vælg ”API controller with actions, using Entity Framework”.



4. Wizarden har tilføjet: `public DbSet<User> User { get; set; }` til ApplicationDbContext-klassen, så næste trin er at opdatere database-strukturen med disse kommandoer i Package manager console:
   `add-migration UserAdded`
   `update-database`

5. Installer bcrypt.
   Højre-klik på projektet og vælg ”Manage NuGet Packages”.
   Vælg Browse og søg efter bcrypt.
   Vælg BCrypt.Net-Next og klik install.



6. Tilføj en AccountController som skal være ansvarlig for bruger registrering og login.
   Højreklik på Controllers-mappen og vælg ”add Controller”.
   Vælg ”API controller - empty”.
   Navngiv controlleren AccountController (eller lignende).

7. Tilføj en public constructor til AccountControlleren som modtager ApplicationDbContext og lagrer den i et datamedlem (kopier koden fra ManageUsersControlleren).

8. Tilføj følgende using statements:

```
using Microsoft.EntityFrameworkCore;
using SecureAPIDemo.Models;
using static BCrypt.Net.BCrypt;
```

9. Tilføj `const int BcryptWorkfactor = 10;` til AccountControlleren.

10. Tilføj en Register action til AccountControlleren.

```
[HttpPost("register"), AllowAnonymous]
public async Task<ActionResult<UserDto>> Register(UserDto regUser)
{
    regUser.Email = regUser.Email.ToLower();
    var emailExist = await _context.Users.Where(u =>
         u.Email == regUser.Email).FirstOrDefaultAsync();
    if (emailExist != null)
        return BadRequest(new { errorMessage = "Email already in use" });
    User user = new User()
    {
        Email = regUser.Email,
        FirstName = regUser.FirstName,
        LastName = regUser.LastName
    };
    user.PwHash = HashPassword(regUser.Password, BcryptWorkfactor);
    _context.Users.Add(user);
    await _context.SaveChangesAsync();
    return CreatedAtAction("Get", new { id = user.UserId }, regUser);
}
```

11. Tilføj en Get action til AccountControlleren. Koden kan kopieres fra ManageUsersControlleren.

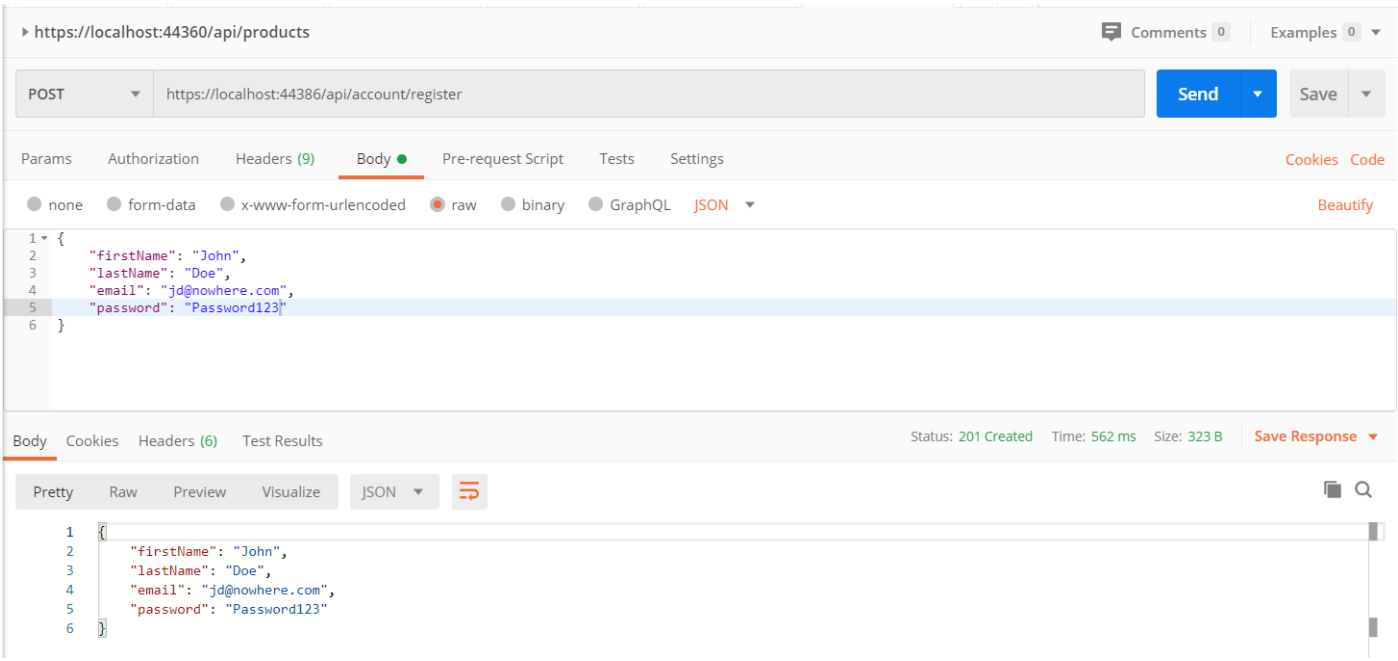
```
// GET: api/Account/5
[HttpGet("{id}", Name = "Get")]
public async Task<ActionResult<UserDto>> Get(int id)
{
    var user = await _context.Users.FindAsync(id);

    if (user == null)
    {
         return NotFound();
    }
    var userDto = new UserDto();
    userDto.Email = user.Email;
    userDto.FirstName = user.FirstName;
    userDto.LastName = user.LastName;
    return userDto;
}
```

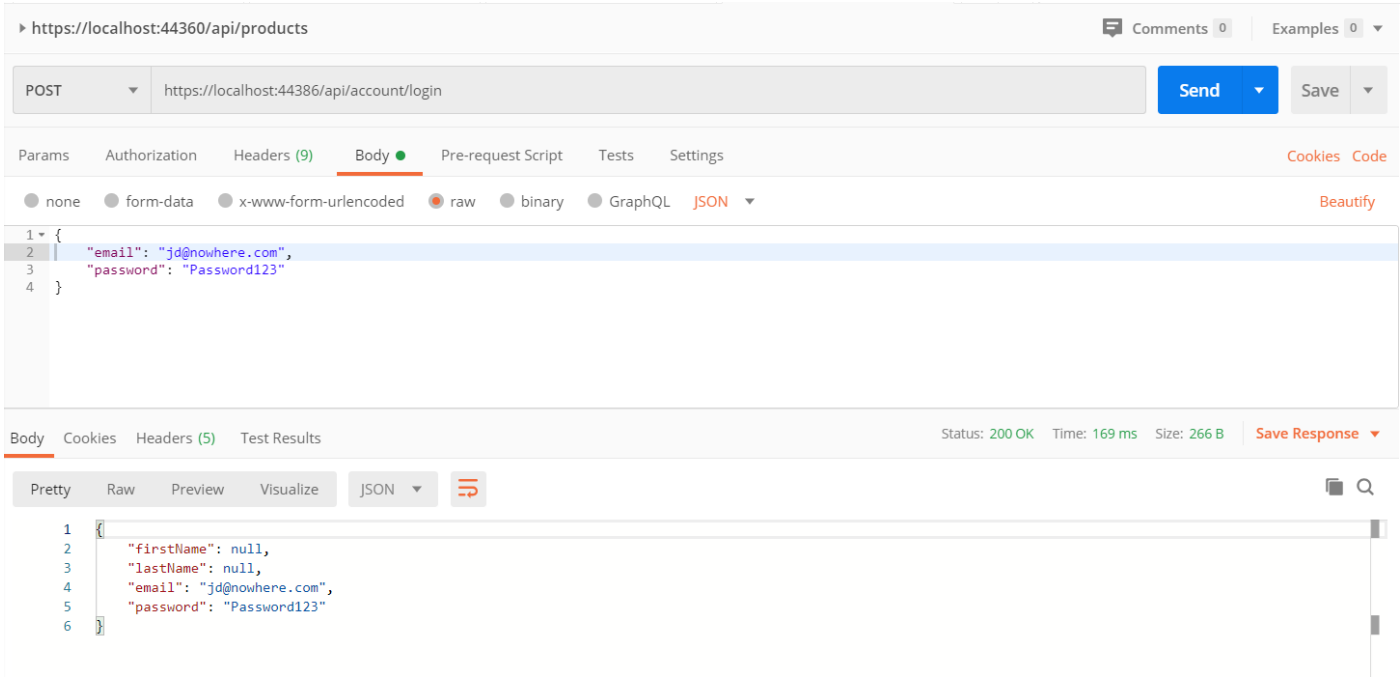
12. Tilføj en Login action til AccountControlleren.

```
[HttpPost("login"), AllowAnonymous]
public async Task<ActionResult<UserDto>> Login(UserDto login)
{
    login.Email = login.Email.ToLower();
    var user = await _context.Users.Where(u =>
                    u.Email == login.Email).FirstOrDefaultAsync();
    if (user != null)
    {
        var validPwd = Verify(login.Password, user.PwHash);
        if (validPwd)
        {
            return login;
        }
    }
    ModelState.AddModelError(string.Empty, "Forkert brugernavn eller password");
    return BadRequest(ModelState);
}
```

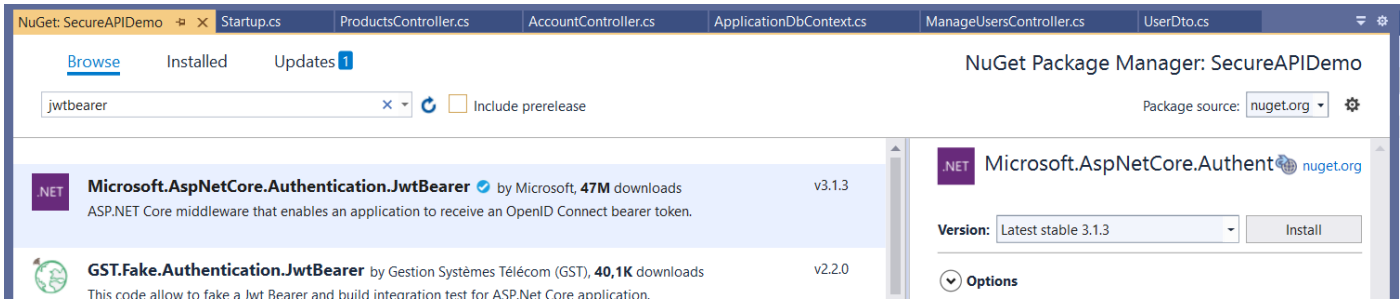
13. Test om registrering af brugere og login virker med brug af Postman.



H



14. Da salt og hash af password og efterfølgende validering nu virker, så er næste skridt at få genereret en jwt-token ved succesfuld registrering og login.
    Det først der skal ske, er at installerer Microsofts JwtBearer NuGet package:



15. Både når jwt-token skal genereres, og når den skal valideres skal der bruges en key, og den key skal være hemmelig. Da programkoden ofte lagres og deles på et git-repostitory bør denne key ikke defineres som en konstant i programkoden, men indlæses fra en settingsfil (eller endnu bedre fra en environment variabel, user secret eller key-vault store).
    Dette kan gøres ved at lave en klasse til opbevaring af programindstillinger, og så indlæse disse (keyen) fra applicationSettings.
    Klasse til typestærke programindstillinger (lagring af den indlæst key):

```
public class AppSettings
{
    public string SecretKey { get; set; }
}
```

Indlæsning af key fra settings og lagring i IoC container, så den kan injectes via dependency injection sker i StartUp.cs:

```
// configure strongly typed settings objects
var appSettingsSection = Configuration.GetSection("AppSettings");
services.Configure<AppSettings>(appSettingsSection);
```

16. Konfigurering af jwt-token i StartUp.cs – ConfigureServices():

```
// configure jwt authentication
var appSettings = appSettingsSection.Get<AppSettings>();
var key = Encoding.ASCII.GetBytes(appSettings.SecretKey);
services.AddAuthentication(x =>
{
    x.DefaultAuthenticateScheme = JwtBearerDefaults.AuthenticationScheme;
    x.DefaultChallengeScheme = JwtBearerDefaults.AuthenticationScheme;
})
.AddJwtBearer(x =>
{
    x.RequireHttpsMetadata = false;
    x.SaveToken = true;
    x.TokenValidationParameters = new TokenValidationParameters
    {
        ValidateIssuerSigningKey = true,
        IssuerSigningKey = new SymmetricSecurityKey(key),
        ValidateIssuer = false,
        ValidateAudience = false
    };
});
```

17. Konfigurering af authentication i StartUp.cs – Configure ():

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

18. Tilføj en klasse som skal bruges til at sende jwt-token til clienten:

```
public class TokenDto
{
    public string JWT { get; set; }
}
```

19. Tilføj generering af jwt-token i login og evt. også i register:

```
public class AccountController : ControllerBase
{
    private readonly ApplicationDbContext _context;
    private readonly AppSettings _appSettings;

    public AccountController(ApplicationDbContext context,
        IOptions<AppSettings> appSettings)
    {
        _context = context;
        _appSettings = appSettings.Value;
    }

    [HttpPost("login"), AllowAnonymous]
    public async Task<ActionResult<TokenDto>> Login(UserDto login)
    {
        login.Email = login.Email.ToLower();
        var user = await _context.Users.Where(u => u.Email == login.Email)
            .FirstOrDefaultAsync();
        if (user != null)
        {
            var validPwd = Verify(login.Password, user.PwHash);
            if (validPwd)
            {
                var token = new TokenDto();
                token.JWT = GenerateToken(user);
                return token;
            }
        }
        ModelState.AddModelError(string.Empty, "Forkert brugernavn eller password");
        return BadRequest(ModelState);
    }

    private string GenerateToken(User user)
    {
        var claims = new Claim[]
        {
            new Claim("Email", user.Email),
            new Claim(JwtRegisteredClaimNames.GivenName, user.FirstName),
            new Claim("UserId", user.UserId.ToString()),
            new Claim(JwtRegisteredClaimNames.Exp,
                new DateTimeOffset(DateTime.Now.AddDays(1)).ToUnixTimeSeconds().ToString()),
        };

        var key = Encoding.ASCII.GetBytes(_appSettings.SecretKey);
        var token = new JwtSecurityToken(
                    new JwtHeader(new SigningCredentials(
                          new SymmetricSecurityKey(key), SecurityAlgorithms.HmacSha256)),
                          new JwtPayload(claims));

        return new JwtSecurityTokenHandler().WriteToken(token);            }
    }
```

20. Tilføj en secret key til konfigurationsfilen appsettings.json:

```
  "AppSettings": {
    "SecretKey": "ncSK45=)7@#qwKDSopevvkj3274687236"
  },
```

21. API serveren er nu klar til at blive testet.