Summary

Information

Class:	FuelQuoteApp_p1.Controllers.AccountController
Assembly:	FuelQuoteApp_p1
File(s):	E:\UH\Summer\SD\FuelQuoteApp_p1\FuelQuoteApp_p1\Controllers\AccountController.cs

Line coverage

100%

Covered lines:	23
Uncovered lines:	0
Coverable lines:	23
Total lines:	152
Line coverage:	100%

Branch coverage

100%

Covered branches:	10
Total branches:	10
Branch coverage:	100%

Method coverage

Method coverage is only available for sponsors.

Upgrade to PRO version

Methods/Properties

- .ctor(Microsoft.AspNetCore.Identity.User…
- RegisterDataValidation(FuelQuoteApp_p…

Metrics

Method	Branch coverage	Cyclomatic complexity	Line coverage
.ctor(...)	100%	1	100%
RegisterDataValidation(...)	100%	10	100%

File(s)

E:\UH\Summer\SD\FuelQuoteApp_p1\FuelQuoteApp_p1\Controllers\AccountController.cs

#	Line	Line coverage
	1	using FuelQuoteApp_p1.EntModels.Models;
	2	using FuelQuoteApp_p1.Models.Account;
	3	using FuelQuoteApp_p1.Provider;
	4	using FuelQuoteApp_p1.Repository;
	5	using Microsoft.AspNetCore.Authorization;
	6	using Microsoft.AspNetCore.Http;
	7	using Microsoft.AspNetCore.Identity;
	8	using Microsoft.AspNetCore.Mvc;
	9	using Newtonsoft.Json;
	10	using System;
	11	using System.Diagnostics.CodeAnalysis;
	12	using System.Text.RegularExpressions;
	13	using System.Threading.Tasks;
	14	
	15	namespace FuelQuoteApp_p1.Controllers
	16	{
	17	public class AccountController : Controller
	18	{
	19	private readonly UserManager<IdentityUser> userManager;
	20	private readonly SignInManager<IdentityUser> signInManager;
	21	private readonly IFuelQuoteProvider _FuelQuotePro;
	22	
1	23	public AccountController(UserManager<IdentityUser> userManager, SignInManager<IdentityUser> signInManager, IFuel
1	24	{
1	25	this.userManager = userManager;
1	26	this.signInManager = signInManager;
1	27	_FuelQuotePro = FuelQuotePro;
1	28	}
	29	[HttpGet]
	30	[ExcludeFromCodeCoverage]
	31	public IActionResult Display()
	32	{
	33	return View();
	34	}
	35	
	36	[HttpGet]
	37	[ExcludeFromCodeCoverage]
	38	[AllowAnonymous]
	39	public IActionResult Login()
	40	{
	41	return View();
	42	}
	43	
	44	[HttpPost]
	45	[ExcludeFromCodeCoverage]
	46	[AllowAnonymous]
	47	public async Task<IActionResult> Login(Login model)
	48	{
	49	if (ModelState.IsValid)
	50	{
	51	
	52	var result = await signInManager.PasswordSignInAsync(model.Email, model.Password, model.RememberMe, fals
	53	// Uses Key Derivation function To encrypt the password (Default password encryption in Identity User)
	54	if (result.Succeeded)
	55	{
	56	int userID = _FuelQuotePro.GetUserID(model.Email);
	57	User userinfo = new User

```
                    {
                        Id = userID,
                        Email = model.Email
                    };

                    HttpContext.Session.SetString("SessionUser", JsonConvert.SerializeObject(userinfo));

                    bool clientinfo = _FuelQuotePro.GetClientInfo(userID);
                    if (clientinfo)
                    {
                        return RedirectToAction("ClientDashBoard", "Client");
                    }
                    else
                    {
                        TempData["ClientProfileInfo"] = "Fill the profile details before requesting for a quote";
                        return RedirectToAction("ClientProfile", "Client");
                    }

                }
                ModelState.AddModelError(string.Empty, "Invalid Credentials! Please check and enter again");
            }
            return View(model);
        }

        [HttpGet]
        [ExcludeFromCodeCoverage]
        [AllowAnonymous]
        public IActionResult Register()
        {
            return View();
        }

        [HttpPost]
        [ExcludeFromCodeCoverage]
        [AllowAnonymous]
        public async Task<IActionResult> Register(Register registerInfo)
        {
            if (ModelState.IsValid)
            {
                var user = new IdentityUser { UserName = registerInfo.Email, Email = registerInfo.Email };
                var result = await userManager.CreateAsync(user, registerInfo.Password);

                if (result.Succeeded)
                {
                    await signInManager.SignInAsync(user, isPersistent: false);
                    User userinfo = new User
                    {
                        UserName = registerInfo.UserName,
                        Email = registerInfo.Email
                    };
                    _FuelQuotePro.AddUser(userinfo);

                    TempData["RegistrationSuccessful"] = "You're registered succesfully!";
                    return RedirectToAction("Login", "Account");
                }

                foreach (var error in result.Errors)
                {
                    ModelState.AddModelError("", error.Description);
                }
            }
            return View();
        }

        [HttpPost]
        [ExcludeFromCodeCoverage]
        public IActionResult Logout()
        {
            signInManager.SignOutAsync();
            return RedirectToAction("Login", "Account");

        }

        public bool RegisterDataValidation(Register registerinfo)
        {
            bool flag = false;
            if ((registerinfo.UserName.Length <= 50) && (registerinfo.UserName != String.Empty))
            {
                if ((Regex.IsMatch(registerinfo.Email, @"^[\w!#$%&'*+\-/=?\^_`{|}~]+(\.[\w!#$%&'*+\-/=?\^_`{|}~]+)*")) &
                {
                    if (registerinfo.Password == registerinfo.ConfirmPassword)
                    {
                        flag = true;
                    }
                }
            }
            else
            {
                flag = false;
            }

            return flag;
        }
    }
}
```