**AUTHENTICATION AND USER ACCOUNTS**

1. **Facebook**
   1. Go to *developers.facebook.com/apps* and register
   2. Add new app
   3. Setup Facebook login
      1. Enter the domain name and add /signin-facebook
      2. Copy the AppID and App Secret from Settings>Basic to the *startup.auth.cs > app.UseFacebookAuthentication*
2. **Google**
   1. *Console.developers.google.com* and create project
   2. Change or keep project ID
   3. Select app
   4. Go to Library and add Google+ API
   5. Credentials > Create > OAuth client ID
   6. Edit OAuthConsent screen tab
   7. Create credentials > OAuth Client ID
   8. Authorized redirect URIs > add domain name followed by */signin-google*
   9. Copy Client Id and Client Secret to *startup.auth.cs > app.UseGoogleAuthentication*
3. **Microsoft**
   1. *Apps.dev.Microsoft.com* and create app
   2. Add platform (web app), paste the domain name followed by /signin-microsoft
   3. Generate new password, copy AppId and Password to *startup.auth.cs > app.UseMicrosoftAccountAuthentication*
4. **Twitter**
   1. Create new Twitter app and paste the domain name, followed by */signin-twitter*
   2. Copy Consumer Key and Consumer Secret to *startup.auth.cs > app.UseTwitterAuthentication*
5. **GitHub or others**
   1. NuGet: Owin.security.providers
   2. *startup.auth.cs >* add *app.UseGitHubAuthentication (clientId: “”, clientSecret:””) or other authentication supported*
   3. *create new app as above, add urls etc*
6. **Windows**
   1. New project with windows authentication
   2. Login etc are generated automatically
   3. See more in the future

**Email confirmation**

1. NuGet: SendGrid
2. Get SendGrid API: sendgrid.com > login > Settings > API Keys > create
3. IdentityConfig.cs > *EmailService > SendAsync method*

public async Task SendAsync (IdentityMessage message)

{

string key = @””;

*where the email will come from*

var client = new SendGridClient(key);

var from = new EmailAddress (“apiKeyEmail”,”Name”);

*where the email will go*

var subject = message.Subject;

var to = new EmailAddress (message.Destination, “New User”);

var plainTextContent = message.Body;

var htmlContent = message.Body;

*Create new email with the MailHelper class*

var email = MailHelper.CreateSingleEmail (from, to, subject, plainTextContent,htmlContent);

await client.SendEmailAsync(email);

1. AccountController:

Register action:

* 1. Comment out SignInManager etc
  2. Uncomment lines after *send an email with this link*
  3. Return View(“Info”) or other

Login Action

1. Add code to oblige user to confirm the email

var user = await UserManager.FindByNameAsync(model.Email);

if (user != null)

{

if (!await UserManager.IsEmailConfirmedAsync(user.Id))

{

ViewBag.errorMessage = “You must confirm email blah blah blah”;

return View(“Error”);

}

}

1. Error View: change accordingly, by displaying the ViewBag

**Two Factor authentication with SMS**

1. NuGet: Twilio
2. Get API key from Twilio.com
3. IdentityConfig.cs > SmsService >

public Task SendAsync (IdentityMessage message)

{

const string accountSID = @””;

const string authToken = @””;

TwilioClient.Init(accountSID, authToken);

Var sms = MessageResource.Create(

body: message.Body,

*get phone number from Twilio Account*

from: new Twilio.Types.PhoneNumber(“”),

to: new Twilio.Types.PhoneNumber(message.Destination));

return Task.FromResult(0);

}

1. Views > Manage > Index: uncomment Phone Number and Two Factor Authentication

**AUTHORIZATIONS**

1. Simple: checks if user is logged in. Add [Authorize] before each controller that needs authorization or [AllowAnonymous].
2. Role: [Authorize(Roles=”role1, role2”)] for each role or [Authorize(Roles =”role1”)] [Authorize(Roles =”role2”)] , if the user needs to have both roles. Allow or deny access to actions by adding the attribute before each action.
3. View: @User has properties that can be checked and views can be displayed if conditions are matched. Otherwise we can use a view model which returns to the view from the controller that has necessary authorizations.

**SECURITY**

1. [Bind] controller parameter against Object Binding vulnerability
2. File types: include accepted file types (or exclude unwanted)
3. IdentityConfig.cs > Create method: Password security
4. XSS: ASP.NET has built in security
5. Third party libraries: Third party scripts only via HTTPS
6. CSRF: [ValidateAntiForgeryToken] in the controller that needs it