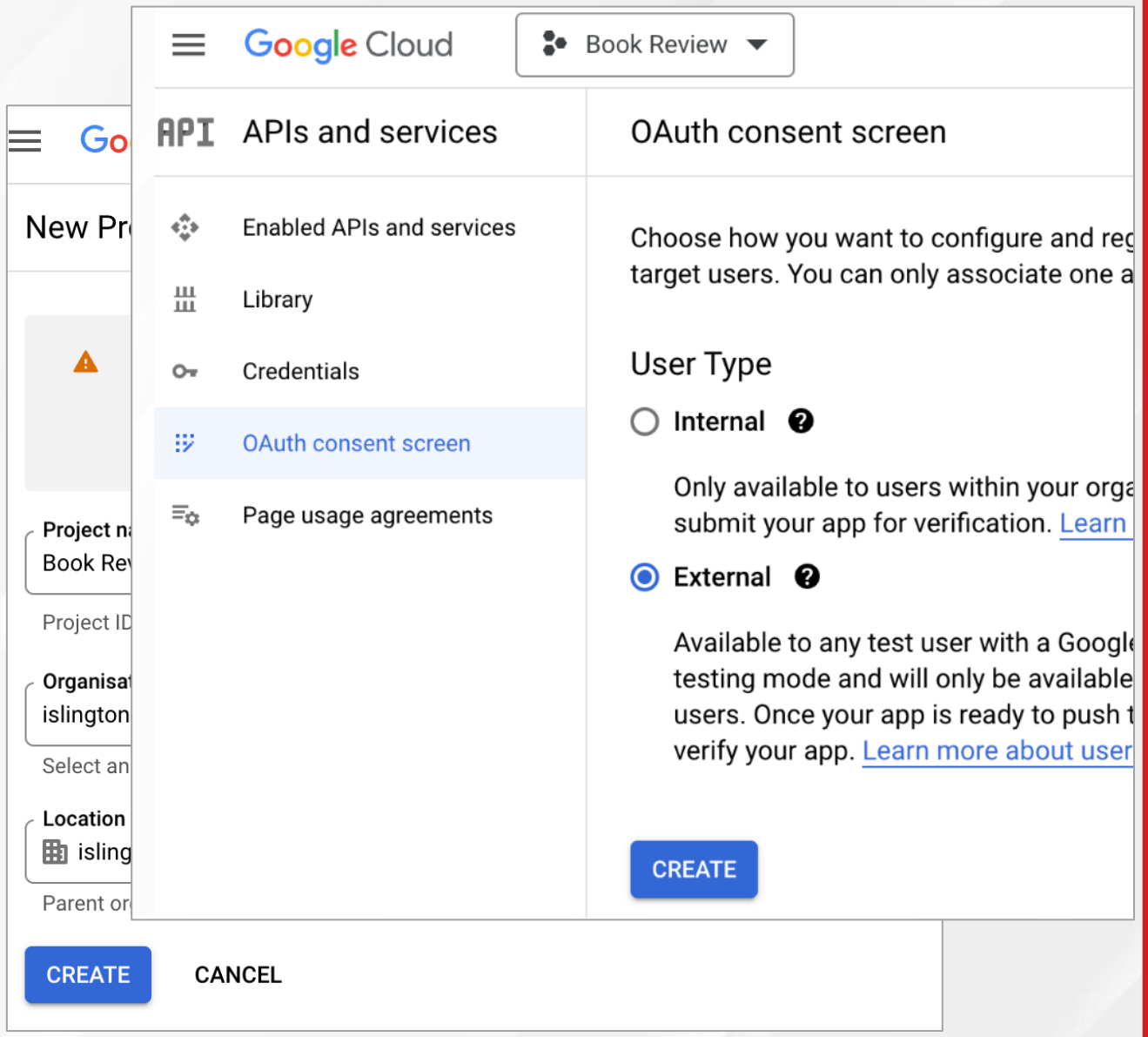
CS6004NT
Application Development

WEEK - 09

External Identity Provider

1. Get Google OAuth Credential

a. Go to Google API & Services.
b. Select a Project (create if not exist)
c. Go to **Oauth consent screen**:
    i. Select User Type - **External** and **CREATE**
    ii. Input app **name**, user **support email**, etc.
    iii. Select **Scopes**
        1. `.../auth/userinfo.email`
        2. `.../auth/userinfo.profile`
    iv. Add **Test users** emails
    v. Review the OAuth consent screen and continue
d. Go to **Credentials**:
    i. Select **CREATE CREDENTIALS** > **OAuth client ID**
    ii. Select **Application type** > **Web application**
    iii. Input **Name**, add **Authorised JavaScript origins** and **Authorized redirect URIs**
        1. JS Origin: https://localhost:3001
        2. Redirect URI: https://localhost:3001/external-login
    iv. Select the **CREATE** button.
    v. Save the Client ID and Client Secret



External Identity Provider

2. API: Add Client ID and Client Secret to appsettings

```
1. "Authentication": {
2.   "Google": {
3.     "ClientId": "627943747507-hg2458llufmjnr1odq69hsg05e0dlrj3.apps.googleusercontent.com"
4.   }
5. }
```

3. API: Install Google Auth package

```
dotnet add package Google.Apis.Auth
```

External Identity Provider

4. Blazor *(index.html)*: Add Google Identity Services library

```
...
 <script src="https://accounts.google.com/gsi/client" async defer></script>
 <script>
     function initGoogleSignIn() {
         google.accounts.id.initialize({
             client_id: "627943747507-hg2458llufmjnr1odq69hsg05e0dlrj3.apps.googleusercontent.com",
             callback: function (googleUser) {
                 var searchParams = new URLSearchParams(window.location.search);
                 searchParams.set("provider", "google");
                 searchParams.set("token", googleUser.credential);
                 window.location.search = searchParams.toString();
             }
         });
         google.accounts.id.renderButton(
             document.getElementById("googleLogin"),
             {theme: "outline", size: "large"}
         );
     }
 </script>
</body>
```

External Identity Provider

5. Blazor *(Login.razor)*: Invoke JS function for Google Login button

```
1.  protected override async Task OnAfterRenderAsync(bool firstRender)
2.  {
3.      if (firstRender)
4.      {
5.          await JsRuntime.InvokeVoidAsync("initGoogleSignIn");
6.      }
7.  }

8.  protected override async Task OnInitializedAsync()
9.  {
10.     var uri = new Uri(NavManager.Uri);
11.     var queryString = uri.Query;
12.     var queryDictionary = HttpUtility.ParseQueryString(queryString);
13.     if (queryDictionary.Count > 0)
14.     {
15.         var provider = queryDictionary["provider"];
16.         var token = queryDictionary["token"];
17.         if (provider != null && token != null)
18.         {
```

```
20.             try
21.             {
22.                 var loginRequest = new ExternalLoginRequest
23.                 {
24.                     Provider = provider,
25.                     Token = token,
26.                 };
27.                 await AuthService.LoginWithGoogleAsync(loginRequest);
28.                 var currentUser = await AuthService.GetProfileAsync();
29.                 StateService.SetCurrentUser(currentUser);
30.                 NavManager.NavigateTo("/");
31.             }
32.             catch (Exception e)
33.             {
34.                 Console.WriteLine(e);
35.             }
36.         }
37.     }
38. }
```

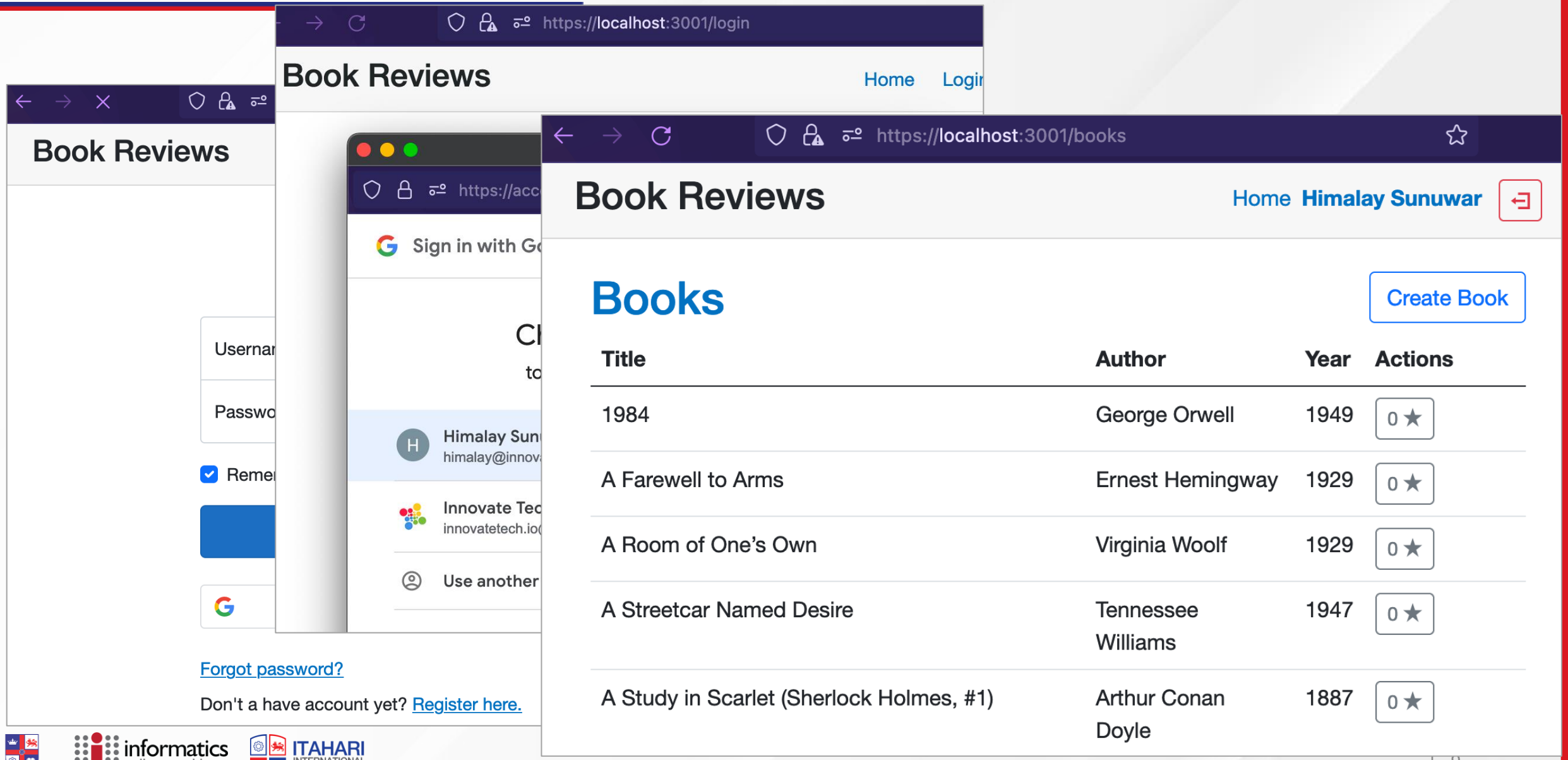
External Identity Provider

6. API *(AuthService.cs)*: External Login Handler

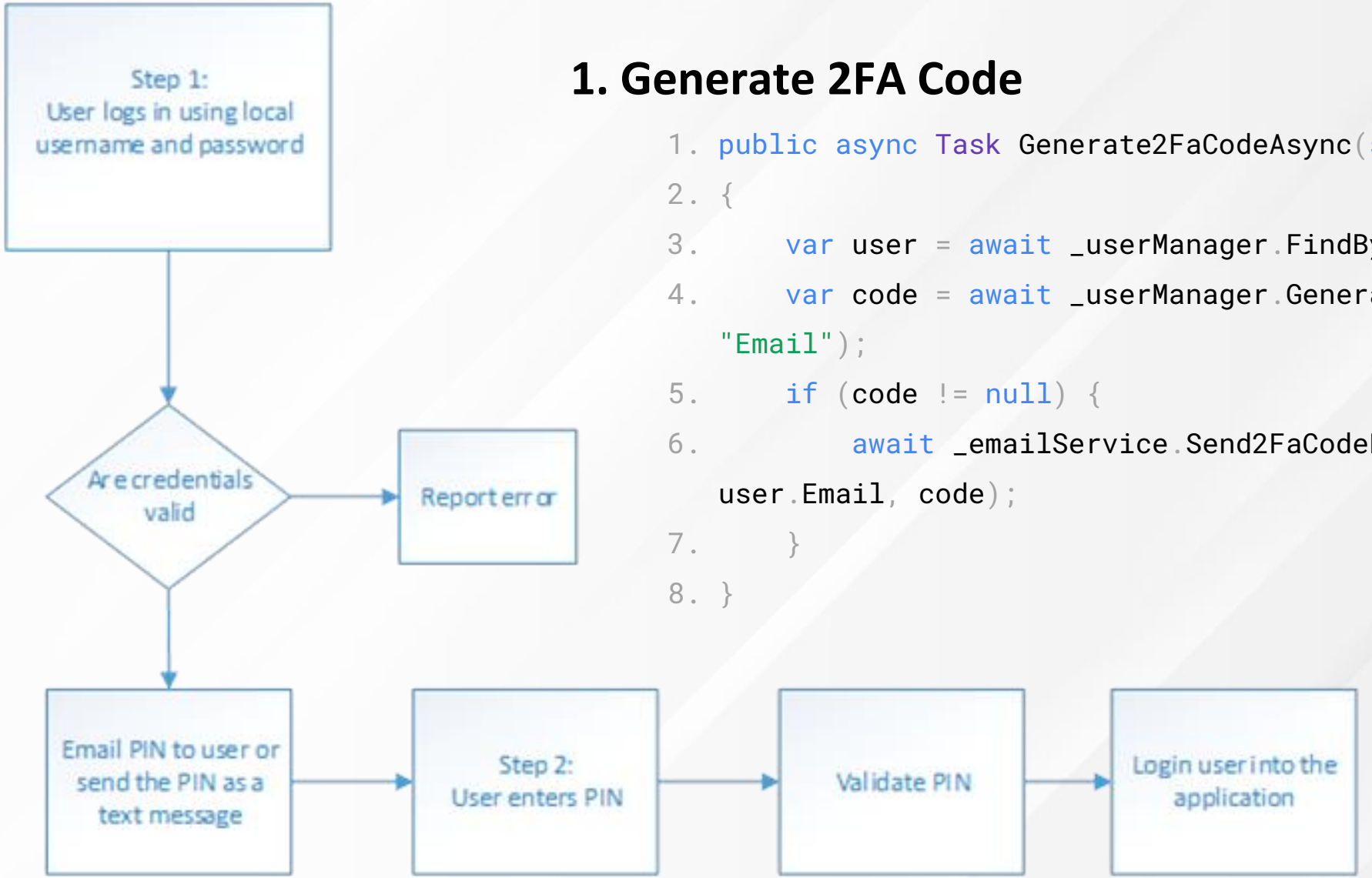
```
1.  public async Task<string> TokenExternalLoginAsync(string provider, string token)
2.  {
3.      var (name, email, subject) = await VerifyGoogleToken(token);

4.      var user = await _userManager.FindByEmailAsync(email);
5.      if (user == null)
6.      {
7.          user = new AppUser { Name = name, UserName = email, Email = email, EmailConfirmed = true };
8.          await _userManager.CreateAsync(user);
9.          await _userManager.AddToRoleAsync(user, "User");
10.     }

11.     var info = new UserLoginInfo(provider, subject, provider);
12.     await _userManager.AddLoginAsync(user, info);

13.     var roles = await _userManager.GetRolesAsync(user);
14.     var role = roles.FirstOrDefault();
15.     return _tokenService.GenerateToken(user, role!);
16. }
```

```
1. private async Task<(string name, string email, string subject)> VerifyGoogleToken(string token)
2. {
3.     try
4.     {
5.         var clientId = _configuration.GetSection("Authentication:Google:ClientId").Value!;
6.         var validationSettings = new GoogleJsonWebSignature.ValidationSettings
7.         {
8.             Audience = new string[] { clientId }
9.         };

10.        var payload = await GoogleJsonWebSignature.ValidateAsync(token, validationSettings);

11.        return (payload.Name, payload.Email, payload.Subject);
12.    }
13.    catch (Exception ex)
14.    {
15.        throw new DomainException("Authentication failed", 401);
16.    }
17.}
```

External Identity Provider



Two-factor Authentication (2FA)



1. Generate 2FA Code

```
1. public async Task Generate2FaCodeAsync(string email)
2. {
3.     var user = await _userManager.FindByEmailAsync(email);
4.     var code = await _userManager.GenerateTwoFactorTokenAsync(user,
   "Email");
5.     if (code != null) {
6.         await _emailService.Send2FaCodeEmailAsync(user.Name,
   user.Email, code);
7.     }
8. }
```

Two-factor Authentication (2FA)

2. Verify 2FA Code

```
1. public async Task Verify2FaCodeAsync(string email, string code)
2. {
3.     var user = await _userManager.FindByEmailAsync(email);
4.     var isValidToken = await _userManager.VerifyTwoFactorTokenAsync(user, "Email", code);
5.     if (isValidToken)
6.     {
7.         var is2FaEnabled = await _userManager.GetTwoFactorEnabledAsync(user);
8.         if (!is2FaEnabled)
9.         {
10.            await _userManager.SetTwoFactorEnabledAsync(user, true);
11.        }
12.    }
13.    else
14.    {
15.        throw new DomainException("Invalid two-factor authentication code");
16.    }
17.}
```

Questions?