Effektive Sicherheitspraktiken für die Softwareentwicklung

LB184 Luca Colona

Contents

Einleitung

In diesem Portfolio geht es um die Implementierung von Applikationssicherheiten. Dabei habe ich die folgenden Punkte bearbeitet:

Aktuelle Bedrohungen erkennen und erläutern können. Aktuelle Informationen zum Thema (Erkennung und Gegenmassnahmen) beschaffen und mögliche Auswirkungen aufzeigen und erklären können.
Sicherheitslücken und ihre Ursachen in einer Applikation erkennen können. Gegenmassnahmen vorschlagen und implementieren können.
Mechanismen für die Authentifizierung und Autorisierung umsetzen können.
Sicherheitsrelevante Aspekte bei Entwurf, Implementierung und Inbetriebnahme berücksichtigen.
Informationen für Auditing und Logging generieren. Auswertungen und Alarme definieren und implementieren.

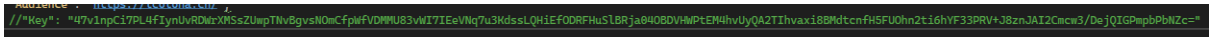
Als Basis für die aufgezeigten Artefakte habe die Insecure App aus dem Modul genommen und erweitert.

Repository

Den Verwendeten Code finden sie hier:

https://github.com/lucacolona/LB_M183_Colona

Der JWT-Key ist auskommentiert, da ich noch den Secret Manager implementiert habe. Zum Testen müssten Sie den Kommentar wieder entfernen.



Handlungsziel 1

Aktuelle Bedrohungen erkennen und erläutern können.

Informationen zu den aktuellen Bedrohungen, Erkennung Gegenmassnahme beschaffe ich über https://owasp.org/www-project-top-ten/ und https://brightsec.com/blog/broken-access-control-attack-examples-and-4-defensive-measures/#regular .Die OWASP Top 10 ist eine Liste mit den aktuellen Top 10 Sicherheitsbedrohungen für Webapplikationen. Auf Platz 1 befindet sich aktuell der Broken Access Control. 2017 war der noch auf Platz 5.

Broken Access Control

Beschreibung:
Access Control stellt sicher, dass der Benutzer nicht ausserhalb der zugewiesenen Berechtigungen handeln kann.
Auswirkungen:
Wenn dieser «Broken», also kaputt ist, führt dies oft zur Offenlegung, Änderung oder Löschung von Daten, auf welche der Benutzer kein Zugriff haben sollte.
Erkennung:

- Durch Logging kann man Überwachen, wer was auf dem System macht und so feststellen, wenn jemand etwas macht, was er nicht dürfte.

Gegenmassnahmen:

- Wenn eine Ressource nicht public ist, soll ein Benutzer standardmässig keinen Zugriff darauf haben.

- Den Access Control Mechanismus global an einem Ort implementieren und dann wiederverwenden, damit man Cross-Origin Resource Sharing verhindern kann.
- Zugriffskontrollen im Modell sollten den Besitz von Datensätzen durchsetzen, anstatt zu akzeptieren, dass der Benutzer jeden Datensatz erstellen, lesen, aktualisieren oder löschen kann.
- Einzigartige geschäftliche Grenzanforderungen der Anwendung sollten durch Domänenmodelle durchgesetzt werden.
- Deaktiviere die Verzeichnisauflistung des Webservers und stelle sicher, dass Dateimetadaten (z. B. .git) und Sicherungsdateien nicht im Web-Root-Verzeichnis vorhanden sind.
- Logge fehlgeschlagene Zugriffskontrollen und benachrichtige Administratoren, wenn es angemessen ist (z. B. bei wiederholten Fehlschlägen).
- Begrenze die Zugriffsraten für APIs und Controller, um den Schaden durch automatisierte Angriffstools zu minimieren.
- Zustandsbehaftete Sitzungskennungen sollten nach dem Logout auf dem Server ungültig gemacht werden. Zustandslose JWT-Tokens sollten stattdessen kurzlebig sein, um das Zeitfenster für einen Angreifer zu minimieren. Für länger lebende JWTs wird dringend empfohlen, den OAuth-Standards zu folgen, um den Zugriff zu widerrufen.

Beurteilung Artefakt

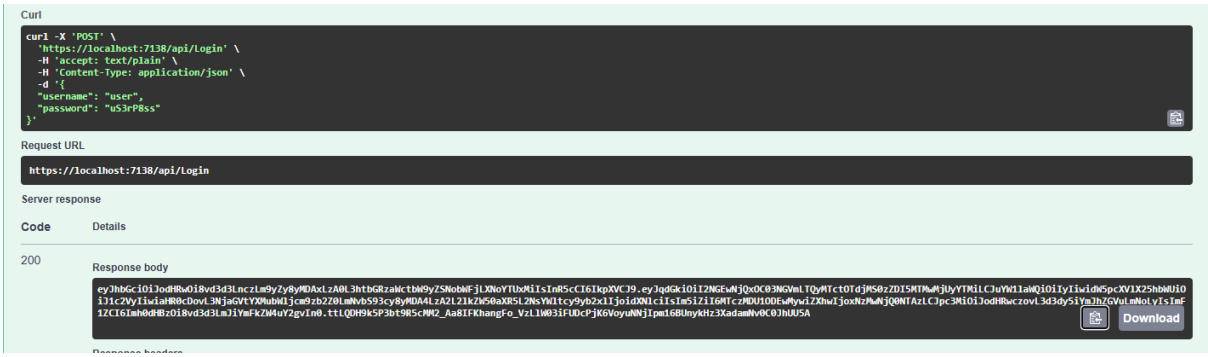
Ich kann damit das HZ abdecken, ich habe die Bedrohung aufgezeigt und ausführlich nach den Vorgaben beschrieben

HZ2 Sicherheitslücke erkennen und beheben

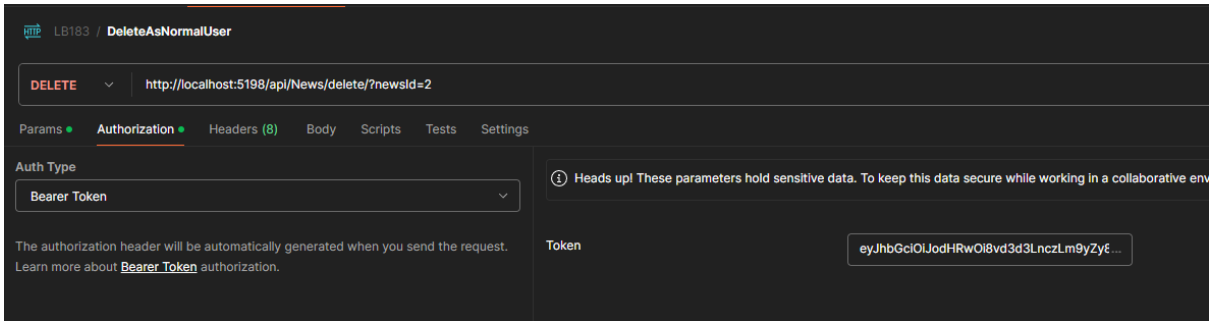
Der Benutzer kann im jetzigen Zustand meiner Applikation Beiträge, die nicht von ihm erstellt wurden, im Frontend nicht löschen, aber er kann einen Request absetzten, mit welchem er einen Beitrag über das Backend trotzdem löschen kann. Das Ganze teste ich mit Postman.

Dieser Beitrag des admins z.B. kann der user nicht übers Frontend entfernen.

Also setzte ich diesen Request mit Postman ab. Dazu hole ich mir zuerst mit Swagger das Bearer Token und führe dann den Request mit Postman aus:



Und wie man sieht, kann der normale Benutzer so einen Admin Beitrag löschen.



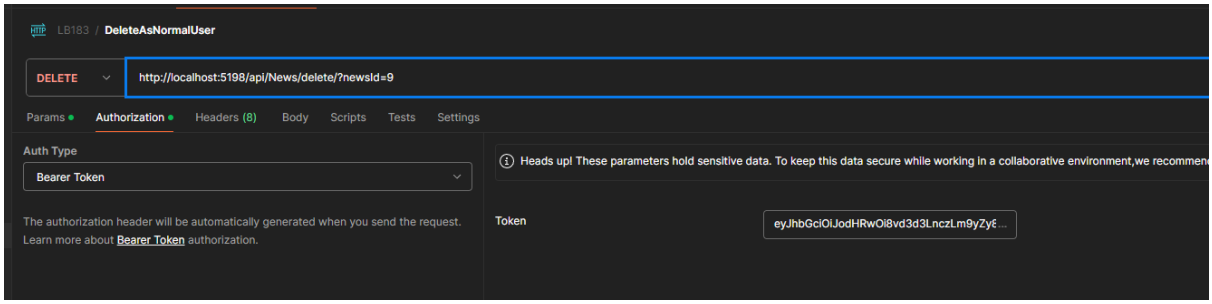
```
[HttpDelete( template: "{id}")]
[ProducesResponseType( statusCode: 200)]
[ProducesResponseType( statusCode: 404)]
0 references
public ActionResult Delete(int newsId)  newsId = 2
{
    var news = _context.News.Find(newsId);  news = {News}
    if (news == null)
    {
        return NotFound(string.Format("News {0} not found", newsId));
    }

    _context.News.Remove(news);
    _context.SaveChanges();  DbContext.SaveChanges() = 1

    return Ok();  ≤ 111ms elapsed
}
```

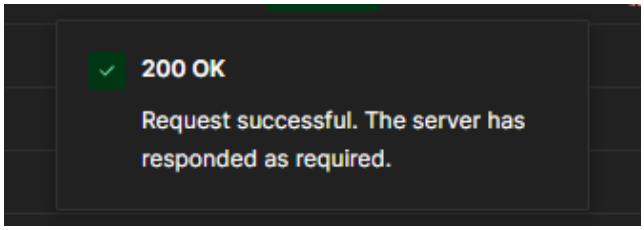
Um den Broken Access Control Fehler zu beheben, muss ich das Backend anpassen, damit dort auch nochmals der Benutzer validiert wird.

Dies mache ich mithilfe des aktuellen authenticated User und überprüfe, ob er berechtigt ist, das delete auszuführen. Wenn nicht, dann gib ich einen Unauthorized Statuscode zurück.

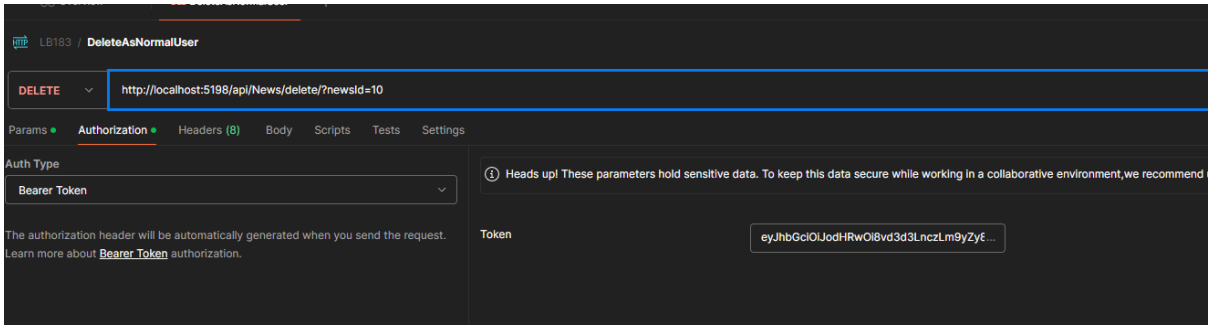
```
public ActionResult Delete(int newsId)
{
    var userClaim = User.Claims.FirstOrDefault();
    var user = _context.Users.FirstOrDefault(user => userClaim != null && user.Username == userClaim.Subject.Name);
    if (user == null)
    {
        return NotFound(string.Format("User {0} not found", userClaim.Subject.Name));
    }

    var news = _context.News.Find(newsId);
    if (news == null)
    {
        return NotFound(string.Format("News {0} not found", newsId));
    }
    if (news.IsAdminNews && !user.IsAdmin || (!news.IsAdminNews && news.AuthorId != user.Id))
    {
        return Unauthorized();
    }

    _context.News.Remove(news);
    _context.SaveChanges();

    return Ok();
}
```

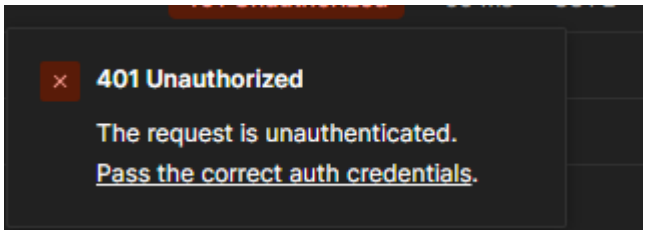
Jetzt führe ich den Request als User mit Postman nochmals aus, und wie man sieht, kann ich meinen eigenen Beitrag löschen:





Aber den Beitrag eines anderen Benutzers nicht mehr:





Somit konnte ich den Broken Access Control beheben.

Beurteilung Artefakt

Ich kann mit dem Artefakt aufzeigen, wie man den Broken Access Control erkennt und eine Gegenmassnahme implementieren kann.

HZ3: Authentifizierung & Autorisierung umsetzen

Ich verwende als Basis die Insecure-App ohne Authetifizierung und Authorisierung.

Autorisierung

Zuerst definiere ich die Rollen.

- Leser
- Autoren (können Beiträge erstellen)
- Admins (Zugriff auf Beiträge von normalen Benutzern

Dazu erstelle ich ein Roles Enum und füge dies der User Entity hinzu und erstelle eine Migration.

```
1 reference
public enum UserRole
{
    Leser,
    Autor,
    Admin
}
```

```
8 references
public class User
{
    [Key]
    4 references
    public int Id { get; set; }
    5 references
    public string Username { get; set; }
    4 references
```

```
/// <inheritdoc />
1 reference
public partial class UserRole : Migration
{
    /// <inheritdoc />
    0 references
    protected override void Up(MigrationBuilder migrationBuilder)
    {
        migrationBuilder.DropColumn(
            name: "IsAdmin",
            table: "Users");

        migrationBuilder.AddColumn<int>(
            name: "UserRole",
            table: "Users",
            type: "int",
            nullable: false,
            defaultValue: 0);

        migrationBuilder.UpdateData(
            table: "Users",
            keyColumn: "Id",
            keyValue: 1,
            column: "UserRole",
            value: 2);

        migrationBuilder.UpdateData(
            table: "Users",
            keyColumn: "Id",
            keyValue: 2,
            column: "UserRole",
            value: 1);
    }
```

Somit habe ich die verschiedenen Rollen und Funktionen definiert und implementiert.

Authentifizierung

JWT-Token

JWT-Tokens verwendet man zur Authentifizierung und werden erstellt, wenn ein Benutzer sich anmeldet.

Ein JWT-Token hat immer ein Issuer, eine Audience und Claims. Die Claims enthalten Infos über den User, das Ablaufdatum des Tokens etc. Wenn das Token abläuft, muss der User ein neues vom Server abrufen. Der Vorteil ist, dass der Benutzer keine Informationen über die Benutzer verwalten muss.

Um die JWT-Authentifizierung einbauen zu können, muss man folgende Sachen beachten:

- Beim Login muss ein Token generiert und zurückgegeben werden.
- Es braucht eine Middleware, welche das JWT verifiziert
- Alle nicht-öffentlichen Endpoints müssen gesichert werden.

JWT-Bearer Nuget installieren:
https://www.nuget.org/packages/Microsoft.AspNetCore.Authentication.JwtBearer/8.0.0/

Erzeugen eines Tokens beim Login:

Ich rufe die CreateToken() Methode bei einem erfolgreichen Login auf

Ablauf:
Zuerst werden der Issuer und die Audience aus der appsettings.json geladen

```
string issuer = _configuration.GetSection(key: "Jwt:Issuer").Value!;
string audience = _configuration.GetSection(key: "Jwt:Audience").Value!;
```

```
"Jwt": {
  "Issuer": "https://lcolona.ch/",
  "Audience": "https://lcolona.ch/",
  "Key": "47v1npCi7PL4fIynUvRDWrXMSsZUwpTNvBgvsNOmCfpWfVDMMU83vWI7IEeVNq7u3KdssLQHiEfODRFHuSlBRja04OBDVHWPtEM4hvUyQA2TIhvaxi8BMdtcnfH5FUOhn2ti6hYF33PRV+J8znJAI2Cmcw3/DejQIGPmpbPbNZc="
```

Dem Beispiel wegen habe ich einfach für beides https://lcolona.ch/ verwendet.

Danach werden die verschiedenen Claims (Inhalt des Tokens) festgelegt.

```
List<Claim> claims = new List<Claim> {
        new Claim(type: JwtRegisteredClaimNames.Jti, value: Guid.NewGuid().ToString()),
        new Claim(type: JwtRegisteredClaimNames.NameId, value: user.Id.ToString()),
        new Claim(type: JwtRegisteredClaimNames.UniqueName, value: user.Username),
        new Claim(type: ClaimTypes.Role, value: nameof(user.UserRole).ToLower())
};
```

Es gibt Claims für: Die Id des Tokens, Id des Users, der sich anmeldet, Benutzername des Users und die Rolle des Users.

Danach wird der SymmetricSecurityKey aus dem Base64 encodierten JWT:Key erstellt (befindet sich auch in den appsettings).

Mit dem Security Key und dem HmacSha512-Algorithmus wird werden die SigningCredentials erstellt. Diese werden verwendet, um den Token zu signieren.

```
string base64Key = _configuration.GetSection( key: "Jwt:Key").Value!;
SymmetricSecurityKey securityKey = new SymmetricSecurityKey(Convert.FromBase64String(base64Key));

SigningCredentials credentials = new SigningCredentials(
        securityKey,
        algorithm: SecurityAlgorithms.HmacSha512Signature);
```

Somit habe ich alle Infos, um dann den Token zu erstellen und an den User zurückzugeben.

```
JwtSecurityToken token = new JwtSecurityToken(
    issuer: issuer,
    audience: audience,
    claims: claims,
    notBefore: DateTime.Now,
    expires: DateTime.Now.AddDays(1),
    signingCredentials: credentials
 );

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

Middleware

Damit sich der Benutzer beim Absetzten eines Requests Authentifizieren kann brauche ich eine Middleware. Diese befindet sich im Program.cs

(Middleware Aufbau erklären)

Sicherung Endpoints

Um die Endpoints zu sichern, kann man einem Controller oder einzelnen Methoden das [Authorize] Attribut hinzufügen. Das heisst, ein Benutzer muss mit einem gültigen Token angemeldet sein, um auf den Endpunkt zugreifen zu können.

Beispiel News Controller:

```
[Route( template: "api/[controller]")]
[Authorize]
[ApiController]

public class NewsController : ControllerBase
{
```

2FA

Um eine zusätzliche Sicherheitsstufe einzubauen, werde ich noch die 2FA von Google implementieren.

Alse erstes installiere ich das Google Authenticator nuget package.

```
M183>dotnet add package GoogleAuthenticator --version 3.2.0
```

Als erstes brauche ich einen Auth Controller, damit der Benutzer die 2FA einrichten kann. Um den Code für die Einrichtung anzuzeigen, brauche ich ein DTO mit der Image Url.

```
1  namespace M183.Controllers.Dto
2  {
       3 references
3      public class Auth2FADto
4      {
           1 reference
5          public string? QrCodeSetupImageUrl { get; set; }
6      }
7  }
```

Um das Secret für die 2FA zu speichern, muss ich die User Entity erweitern.

```
8 references
public class User
{
    [Key]
    4 references
    public int Id { get; set; }
    7 references
    public string Username { get; set; }
    4 references
    public string Password { get; set; }
    6 references
    public UserRole UserRole { get; set; }
    1 reference
    public string? SecretKey2FA { get; set; }

}
```

Für die neue Property erstelle ich eine Migration.

```
ef migrations add SecretKey2FA
```

Mit der Enable2FA Methode kann der User dann die 2FA einrichten. Darin wird ein 10-stelliger Secret Key generiert und auf dem User in der DB abgelegt und damit wird der QR-Coder erstellt und als ImageUrl im Auth2FA Dto zurückgegeben.

```
[HttpPost]
[ProducesResponseType(statusCode: 200)]
[ProducesResponseType(statusCode: 404)]
0 references
public ActionResult<Auth2FADto> Enable2FA()
{
    var user = _context.Users.Find(params keyValues: _userService.GetUserId());
    if (user == null)
    {
        return NotFound(string.Format("User {0} not found", _userService.GetUsername()));
    }
    {
        var secretKey:string = Guid.NewGuid().ToString().Replace(oldValue: "-", newValue: "").Substring(startIndex: 0, length: 10);
        string userUniqueKey = user.Username + secretKey;
        string issuer = _configuration.GetSection(key: "Jwt:Issuer").Value!;
        TwoFactorAuthenticator authenticator = new TwoFactorAuthenticator();
        SetupCode setupInfo = authenticator.GenerateSetupCode(issuer, accountTitleNoSpaces: user.Username, userUniqueKey, secretIsBase32: false, qrPixelsPerModule: 3);

        user.SecretKey2FA = secretKey;
        _context.Update(user);
        _context.SaveChanges();

        Auth2FADto auth2FADto = new Auth2FADto();
        auth2FADto.QrCodeSetupImageUrl = setupInfo.QrCodeSetupImageUrl;

        return Ok(auth2FADto);
    }
}
```

Danach erstelle ich noch das js File für die Enable 2FA Seite („enable2FA.js“ im Repo).

```
aScript Content Files
    0 references
function onEnable2FA() {
    fetch("/api/Auth/", {
        method: "POST",
        headers: {
            'Accept': 'application/json',
            'Content-Type': 'application/json',
            'Authorization': getJwtHeader(getJwtToken())
        }
    })
        .then((response) => {
            return response.json();
        })
        .then((data) => {
            displayQrCode(data);
        })
        .catch((error) => {
            showError(error);
        });
}
```

onEnable2FA() sendet den Post um die 2FA einzurichten. Die Anfrage enthält das JWT-Token zur Authentifizierung. Wenn Erfolgreich → displayQrCode() um QR-Code anzuzeigen, wenn Fehler dann showError() um Error anzuzeigen.

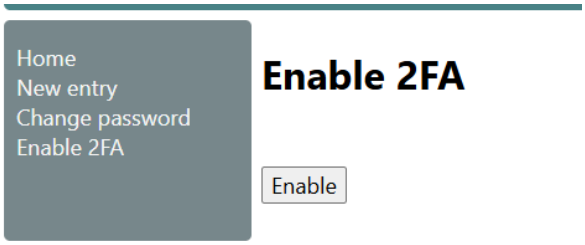
Danach füge ich die Seite noch in der Navigation hinzu.

```html
<nav>
    <ul>
        <li><a href="index.html">Home</a></li>
        <li><a href="index.html?page=createNews">New entry</a></li>
        <li><a href="index.html?page=changePassword">Change password</a></li>
        <li><a href="index.html?page=enable2FA">Enable 2FA</a></li>
    </ul>
</nav>
```

```html
<script>
    if (!isLoggedIn()) {
        createLoginForm();
    }
    else {
        // User is lgged in.
        document.getElementById("loggedInUsername").innerText = getUsername();

        // Read parameter from URL.
        var urlParams = new URLSearchParams(window.location.search);
        var page = urlParams.get('page');

        if (page == 'login') {
            createLoginForm();
        }
        else if (page == 'createNews') {
            createNews();
        }
        else if (page == 'changePassword') {
            createChangePasswordForm();
        }
        else if (page == 'enable2FA') {
            createEnable2FAForm();
        }
        else {
            loadNews();
        }
    }
</script>
```

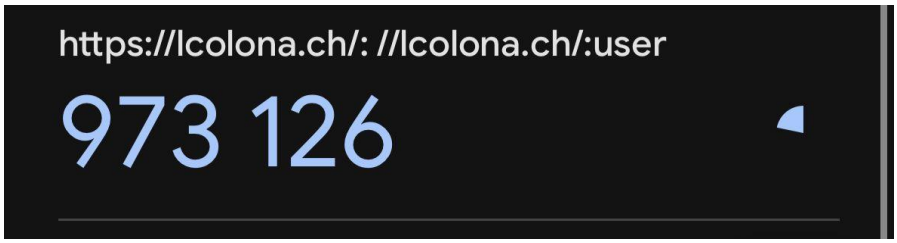
Einrichtung 2FA

Somit kann ein Benutzer, wenn er angemeldet ist, die 2FA einrichten:

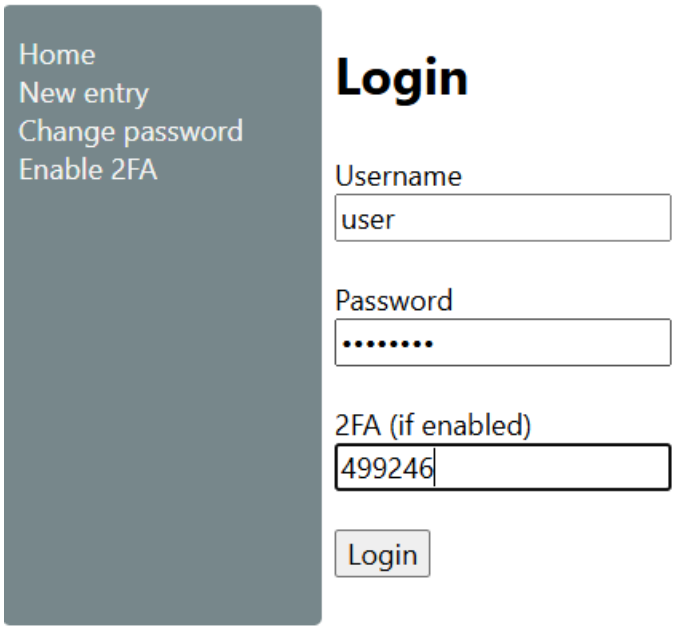


Beim Klicken auf den Button wird der QR-Code generiert.

Diesen Code kann man z.B. mit dem Google Authenticator einscannen und hat dann den Code:



Wenn ich mich jetzt abmelde und wieder anmelde, werde ich nach dem 2FA Code gefragt.



Wenn dieser korrekt ist kann ich mich anmelden, sonst erhalte ich einen Fehler.

Beurteilung Artefakt

Ich konnte die Authentifizierung und Autorisierung implementieren und aufzeigen, wie ich das gemacht habe. Ebenfalls habe ich für zusätzliche Sicherheit noch die 2FA mit Google hinzugefügt.

HZ4: Sicherheitsrelevante Aspekte berücksichtigen

Entwurf

- Identifizierung von potenziellen Bedrohungen und Schwachstellen in der Anwendung.
- Darauf achten, dass Benutzer und Komponenten nur die Berechtigungen haben, die sie für ihre Funktion benötigen. Z.B. mit einem Autorisierungskonzept wie ich es erstellt habe.

Implementierung

- Security sollte konsistent und so klar wie möglich implementiert werden.
- Unnötige Komplexität sollte vermieden werden, da so potenziell mehr Fehler entstehen können.
- Security sollte man an einem zentralen Ort implementieren, damit nicht von Ort zu Ort die Vorgaben, Validierungen etc. anders sind.
- Jede Eingabe sollte geprüft werden.
- Stelle sicher, dass aktuelle und sichere Versionen von Bibliotheken und Frameworks verwendet werden.
- Verhindern, dass detaillierte Fehlermeldungen an den Benutzer gelangen, die sensible Informationen enthalten.
- Code-Reviews und automatische Tests zur Erkennung von Sicherheitslücken durchführen.

Inbetriebnahme

- Sicherstellen, dass Keys nicht veröffentlicht werden, z.B. durch das Verwenden von Key Vaults. Hier das Beispiel mit der Key Storage:

```
PS C:\Users\school\Desktop\Schule\Schule\Berufsschule\Module\M183\AbgabeLb\LB183_Code\M183> dotnet user-secrets init
Set UserSecretsId to '02b83787-1bfb-4431-96c1-7e67357b5069' for MSBuild project 'C:\Users\school\Desktop\Schule\Schule\Berufsschule\Module\M183\AbgabeLb\LB183_Code\M183\M183.csproj'.
PS C:\Users\school\Desktop\Schule\Schule\Berufsschule\Module\M183\AbgabeLb\LB183_Code\M183> dotnet user-secrets set "Jwt:Key" "47v1npCi7PL4fIynUvRDWrXMSsZUwpTNvBgvsNOmCfpWfVDMMU83vWI7IEeVNq7u3KdssLQHiEfODRFHuSlBRja04OBDVHWPtEM4hvUyQA2TIhvaxi8BMdtcnfH5FUOhn2ti6hYF33PRV+J8znJAI2Cmcw3/DejQIGPmpbPbNZc="
Successfully saved Jwt:Key = 47v1npCi7PL4fIynUvRDWrXMSsZUwpTNvBgvsNOmCfpWfVDMMU83vWI7IEeVNq7u3KdssLQHiEfODRFHuSlBRja04OBDVHWPtEM4hvUyQA2TIhvaxi8BMdtcnfH5FUOhn2ti6hYF33PRV+J8znJAI2Cmcw3/DejQIGPmpbPbNZc= to the secret store.
PS C:\Users\school\Desktop\Schule\Schule\Berufsschule\Module\M183\AbgabeLb\LB183_Code\M183>
```

- Implementieren von einem System zur Protokollierung von sicherheitsrelevanten Ereignissen, wie z. B. Anmeldeversuchen
- Alarme einrichten, die auf ungewöhnliche Aktivitäten oder mögliche Sicherheitsvorfälle aufmerksam machen.
- Sicherstellen, dass das System regelmässig mit Sicherheitsupdates auf den sichersten Stand gebracht wird.

Beurteilung Artefakt

Ich habe die Sicherheitsrelevanten Aspekte aufgezeigt.

HZ5 Auditing und Logging

Auditing

Als Erstes habe ich eine News Auditing Tabelle erstellt.

```
namespace M183.Models
{
    0 references
    public class NewsAudit
    {
        [Key]
        [DatabaseGenerated(DatabaseGeneratedOption.Identity)]
        0 references
        public int Id { get; set; }

        [Required]
        0 references
        public int NewsId { get; set; }

        [Required]
        [MaxLength(255)]
        0 references
        public string Action { get; set; }

        [Required]
        0 references
        public int AuthorId { get; set; }
    }
}
```

Ich füge die neue Entity dem DbContext hinzu.

```
16 references
public class NewsAppContext : DbContext
{
    0 references
    public NewsAppContext(DbContextOptions<NewsAppContext> options) : base(options) { }
    7 references
    public DbSet<News> News { get; set; }
    5 references
    public DbSet<User> Users { get; set; }
    0 references
    public DbSet<NewsAudit> NewsAudits { get; set; }
```

Danach erstelle ich dafür eine Migration

```
\M183>dotnet ef migrations add Auditing
```

```
/// <inheritdoc />
public partial class Auditing : Migration
{
    /// <inheritdoc />
    protected override void Up(MigrationBuilder migrationBuilder)
    {
        migrationBuilder.CreateTable(
            name: "NewsAudits",
            columns: table :ColumnsBuilder => new
            {
                Id = table.Column<int>(type: "int", nullable: false) // OperationBuilder<AddColumnOperation>
                    .Annotation(name: "SqlServer:Identity", value: "1, 1"),
                NewsId = table.Column<int>(type: "int", nullable: false),
                Action = table.Column<string>(type: "nvarchar(max)", nullable: false),
                AuthorId = table.Column<int>(type: "int", nullable: false)
            },
            constraints: table =>
            {
                table.PrimaryKey(name: "PK_NewsAudits", columns: x :{Id,NewsId,…} => x.Id);
            });
    }

    /// <inheritdoc />
    0 references
    protected override void Down(MigrationBuilder migrationBuilder)
    {
        migrationBuilder.DropTable(
            name: "NewsAudits");
    }
}
```

Und erweitere die Up & Down Methoden mit Triggern.

```
protected override void Up(MigrationBuilder migrationBuilder)
{
    migrationBuilder.CreateTable(
        name: "NewsAudit",
        columns: table :ColumnsBuilder => new
        {
            Id = table.Column<int>(type: "int", nullable: false) // OperationBuilder<AddColumnOperation>
                .Annotation(name: "SqlServer:Identity", value: "1, 1"),
            NewsId = table.Column<int>(type: "int", nullable: false),
            Action = table.Column<string>(type: "nvarchar(max)", nullable: false),
            AuthorId = table.Column<int>(type: "int", nullable: false)
        },
        constraints: table =>
        {
            table.PrimaryKey(name: "PK_NewsAudit", columns: x :{Id,NewsId,…} => x.Id);
        });

    migrationBuilder.Sql(@"CREATE TRIGGER insert_news ON dbo.News
        AFTER INSERT
        AS DECLARE
          @NewsId INT,
          @AuthorId INT;
        SELECT @NewsId = ins.ID FROM INSERTED ins;
        SELECT @AuthorId = ins.AuthorId FROM INSERTED ins;

        INSERT INTO NewsAudit (NewsId, Action, AuthorId) VALUES (@NewsId, 'Create', @AuthorId);");

    migrationBuilder.Sql(@"CREATE TRIGGER update_news ON dbo.News
        AFTER UPDATE
        AS DECLARE
          @NewsId INT,
          @AuthorId INT;
        SELECT @NewsId = ins.ID FROM INSERTED ins;
        SELECT @AuthorId = ins.AuthorId FROM INSERTED ins;

        INSERT INTO NewsAudit (NewsId, Action, AuthorId) VALUES (@NewsId, 'Update', @AuthorId);");


    migrationBuilder.Sql(@"CREATE TRIGGER delete_news ON dbo.News
        AFTER DELETE
        AS DECLARE
          @NewsId INT,
          @AuthorId INT;
        SELECT @NewsId = del.ID FROM DELETED del;
        SELECT @AuthorId = del.AuthorId FROM DELETED del;

        INSERT INTO NewsAudit (NewsId, Action, AuthorId) VALUES (@NewsId, 'Delete', @AuthorId);");
}

/// <inheritdoc />
0 references
protected override void Down(MigrationBuilder migrationBuilder)
{
    migrationBuilder.DropTable(name: "NewsAudit");
    migrationBuilder.Sql("DROP TRIGGER IF EXISTS insert_news");
    migrationBuilder.Sql("DROP TRIGGER IF EXISTS update_news");
    migrationBuilder.Sql("DROP TRIGGER IF EXISTS delete_news");
}
```

Ertstellt habe ich diese 3 Trigger:

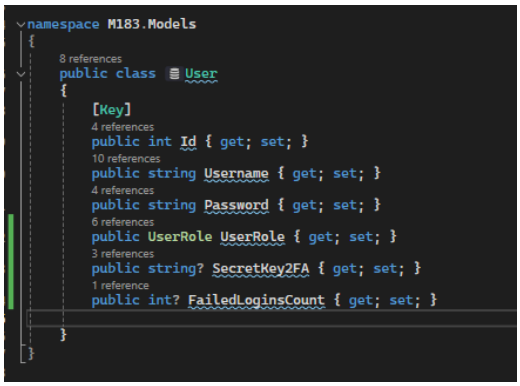
- insert_news
- update_news
- delete_news

Zum Testen erstelle, update und lösche ich ein Eintrag. Das sind dann die Resultate. Auf der DB sieht das dann so aus:

	Id	NewsId	Action	AuthorId
1	1	3	Create	2
2	2	3	Update	2
3	3	3	Delete	2

Logging

Ich will loggen, wie oft ein User sein Passwort falsch eingibt. Bei 10 Mal falsch gibt es einen Error. Ich erweitere als erstes die User Property.



```
namespace M183.Models
{
    8 references
    public class User
    {
        [Key]
        4 references
        public int Id { get; set; }
        10 references
        public string Username { get; set; }
        4 references
        public string Password { get; set; }
        6 references
        public UserRole UserRole { get; set; }
        3 references
        public string? SecretKey2FA { get; set; }
        1 reference
        public int? FailedLoginsCount { get; set; }

    }
}
```

Dann erstellte ich eine Migration.

```
dotnet ef migration add LoginAttemptsCount
```

Hier sind die beiden Methoden für das Loggen:

```
2 references
private void AddFailedLogin(User user)
{
    user.FailedLoginsCount += 1;

    if (user.FailedLoginsCount >= 10)
    {
        _logger.LogCritical(message: $"[{DateTime.Now}] Unauthorized: User {user.Username} login failed {user.FailedLoginsCount} times");
        user.FailedLoginsCount = 0;
    }
    else
    {
        _logger.LogWarning(message: $"[{DateTime.Now}] Unauthorized: User {user.Username} login failed");
    }
    _context.Update(user);
    _context.SaveChanges();
}

1 reference
private void AddFailedLogin(string username)
{
    _logger.LogWarning(message: $"[{DateTime.Now}] Unauthorized: User {username} login failed");
}
```

Wenn ich jetzt das Passwort falsch eingebe, sieht das Log so aus:

```
warn: M183.Controllers.LoginController[0]
      [03/11/2024 20:31:24] Unauthorized: User administrator login failed
```

Und nach 10 Mal dann so:

```
crit: M183.Controllers.LoginController[0]
      [03/11/2024 20:35:45] Unauthorized: User administrator login failed 10 times
```

Beurteilung Artefakt

Ich habe gezeigt wie man das Auditing und Logging implementieren und dann anwenden kann.

Selbsteinschätzung

Kompetenz: Applikationen sicher planen, entwickeln und in Betrieb nehmen.

Ich denke, dass ich die Kompetenz abdecken konnte, aber ich habe das gefühlt, dass wir nicht besonders viel umsetzten, konnten sondern vieles einfach nur Theorie war. Das Entwickeln an der Insecure App war zwar gut, aber ich denke es wäre lehrreicher gewesen, wenn wir mehr Zeit im Modul gehabt hätten und etwas von Grund aufgebaut hätten können. Was ein bisschen vernachlässigt wurde war die Inbetriebnahme, dazu haben wir leider nicht so viel gemacht, ich habe das Gefühl, dass ich da noch am meiste hätte dazulernen können.

Selbstständigkeitserklärung

Selbstständigkeitserklärung

Hiermit erkläre ich, dass ich das **ePortfolio und die Artefakte** selbstständig und ohne fremde Hilfe verfasst und keine anderen Hilfsmittel als die angegebenen verwendet habe.

Insbesondere versichere ich, dass ich alle wörtlichen und sinngemässen Übernahmen aus anderen Werken als solche kenntlich gemacht habe.

Staufen, 03.11.2024 (Ort, Datum)

Luca Colona (Unterschrift)