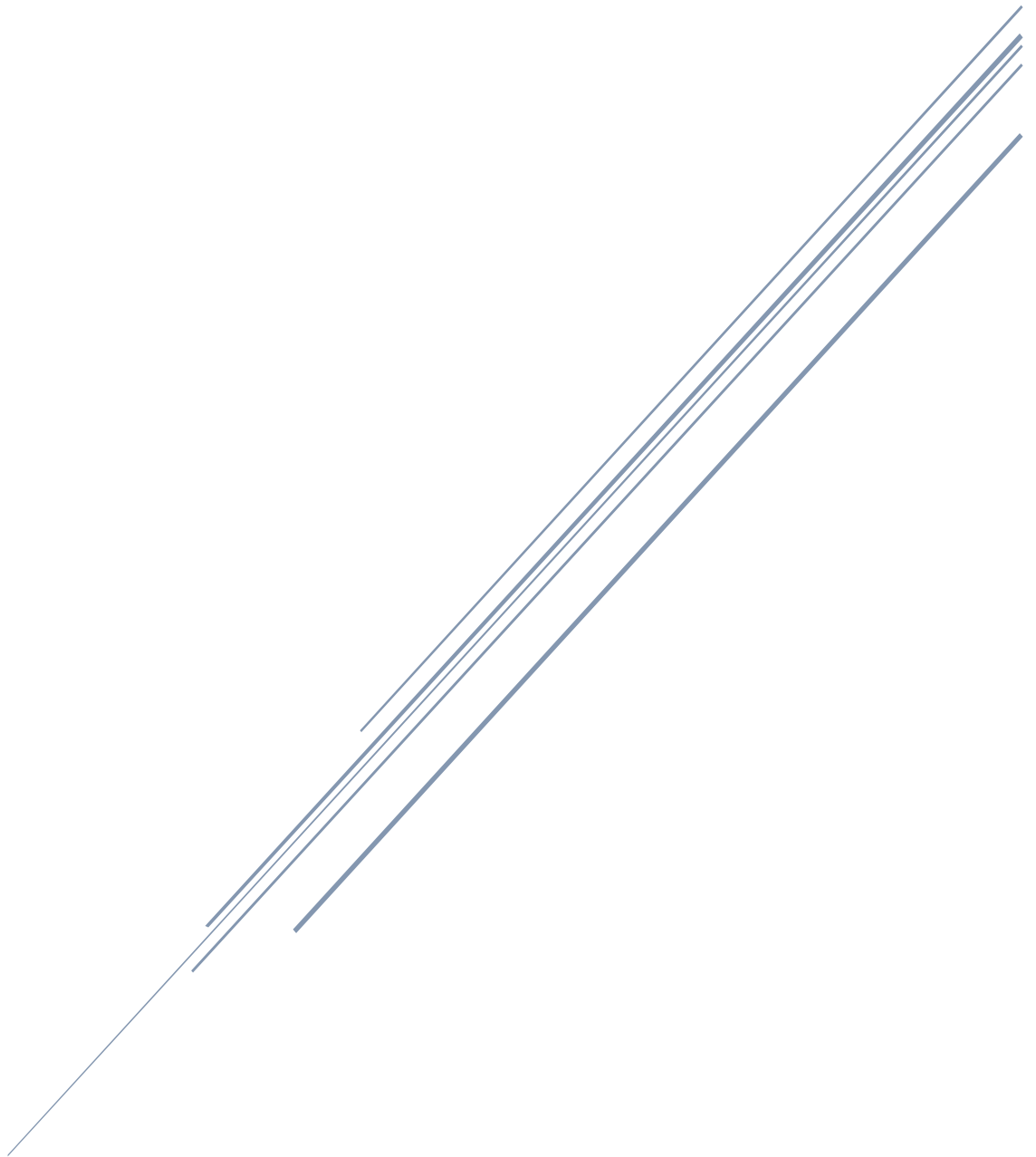


BASEL COIN APP (POC)

Projektdokumentation



Fokko Vos & Robin Ruf
IBZ 2024

Inhaltsverzeichnis

1	EINLEITUNG	3
2	LOGIN MIT INJECTION-PROTECTION.....	3
2.1	BESCHREIBUNG	3
2.2	CODE	3
3	SESSION MANAGEMENT	4
3.1	BESCHREIBUNG	4
3.2	CODE	4
4	APPLIKATIONSLOG	5
4.1	BESCHREIBUNG	5
4.2	CODE	5
5	INPUT VALIDIERUNG	6
5.1	BESCHREIBUNG	6
5.2	CODE	6
6	2FA.....	6
6.1	BESCHREIBUNG	6
7	ABBILDUNGSVERZEICHNIS	7

1 Einleitung

Basel Stadt startet eine eigene Kryptowährung, mit welcher man innerhalb der Region bezahlen können soll. Dafür wird eine Applikation benötigt. Unser Team hat den Prototyp (POC) dafür erstellt. Bei einer solchen App steht die Sicherheit an erster Stelle, weshalb wir in dieser kurzen Dokumentation auf die wichtigsten, bereits im Prototyp eingebauten, Sicherheitsfeatures etwas näher eingehen.

2 Login mit Injection-Protection

https://owasp.org/Top10/A03_2021-Injection/

2.1 Beschreibung

Durch Injections können beispielsweise Benutzerdaten aus der Datenbank ausgelesen, verändert oder sogar ganze Tabellen gelöscht werden, indem schadhafter Code durch die Input-Felder einer Webseite an die Datenbank geschickt werden.

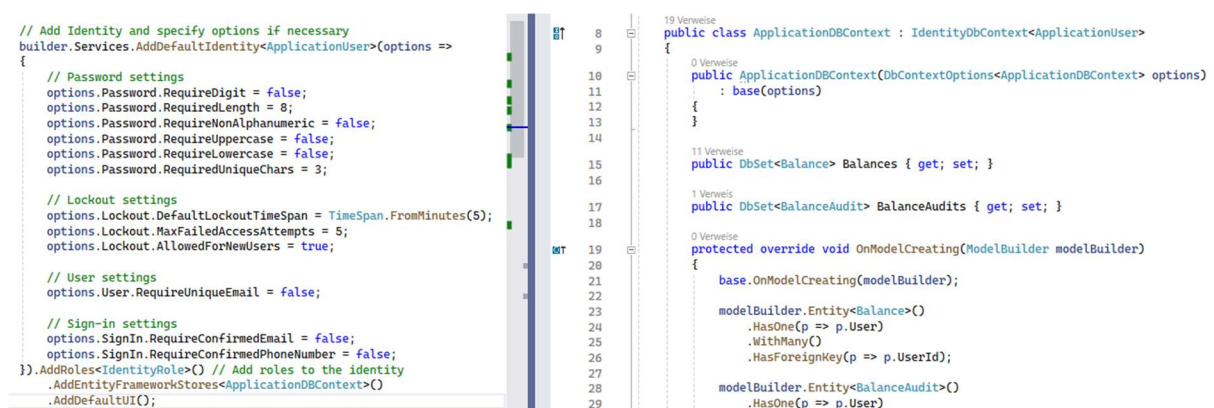
Um Injections zu verhindern, werden die Eingaben des Benutzers als Parameter dem Query übergeben, anstatt eine einfache String-Interpolation zu machen. Somit wird die ganze Eingabe als Text angesehen, egal, welche Zeichen verwendet wurden.

Ausserdem ist die Admin-Route IP-Protected. Nur im lokalen Netzwerk kann auf diese zugegriffen werden. Selbst wenn eine Drittperson die vorherigen Hürden bewältigt, hat er keinen Zugriff auf die Admin-Route, da er sich nicht im internen Netzwerk befindet.

2.2 Code

Durch die Verwendung von Entity Framework Core werden viele sicherheitsrelevante Features automatisch umgesetzt. Die Querys werden nicht durch String-Interpolation, sondern automatisch durch Objects als Parameter zusammengefügt. Somit wird jeder Input als Text angesehen, egal, welche Characters verwendet wurden.

Hier wird auch das Passwort überprüft. Dies wurde durch den NIST-Standard inspiriert. Das Passwort muss eine Länge von mindestens 8 Zeichen, mindestens einen Nummer und einen Grossbuchstaben haben.



```
// Add Identity and specify options if necessary
builder.Services.AddDefaultIdentity<ApplicationUser>(options =>
{
    // Password settings
    options.Password.RequireDigit = false;
    options.Password.RequiredLength = 8;
    options.Password.RequireNonAlphanumeric = false;
    options.Password.RequireUppercase = false;
    options.Password.RequireLowercase = false;
    options.Password.RequiredUniqueChars = 3;

    // Lockout settings
    options.Lockout.DefaultLockoutTimeSpan = TimeSpan.FromMinutes(5);
    options.Lockout.MaxFailedAccessAttempts = 5;
    options.Lockout.AllowedForNewUsers = true;

    // User settings
    options.User.RequireUniqueEmail = false;

    // Sign-in settings
    options.SignIn.RequireConfirmedEmail = false;
    options.SignIn.RequireConfirmedPhoneNumber = false;
}).AddRoles<IdentityRole>() // Add roles to the identity
.AddEntityFrameworkStores<ApplicationDbContext>()
.AddDefaultUI();

19 Verweise
public class ApplicationDbContext : IdentityDbContext<ApplicationUser>
{
    0 Verweise
    public ApplicationDbContext(DbContextOptions<ApplicationDbContext> options)
        : base(options)
    {
    }

    11 Verweise
    public DbSet<Balance> Balances { get; set; }

    1 Verweise
    public DbSet<BalanceAudit> BalanceAudits { get; set; }

    0 Verweise
    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        base.OnModelCreating(modelBuilder);

        modelBuilder.Entity<Balance>()
            .HasOne(p => p.User)
            .WithMany()
            .HasForeignKey(p => p.UserId);

        modelBuilder.Entity<BalanceAudit>()
            .HasOne(p => p.User)
            .WithMany()
            .HasForeignKey(p => p.UserId);
    }
}
```

1 - Injection Protection

3 Session Management

https://cheatsheetseries.owasp.org/cheatsheets/Session_Management_Cheat_Sheet.html

3.1 Beschreibung

Ein gutes Session Management ist essenziell für jede Finanzapp. Zwei wichtige Aspekte wurden umgesetzt: Idle und Absolut Timeout. Durch den Idle Timeout, den man bereits aus E-Banking-Anwendungen kennt, wird definiert, wie lange eine Session aktiv bleibt ohne, dass der Benutzer eine Aktivität vornimmt. Der Absolut Timeout auf der anderen Seite definiert, nach welcher Zeit die Session beendet wird, unabhängig von der Aktivität des Benutzers.

3.2 Code

Durch JavaScript wird der Idle Timeout definiert. Nach einer Inaktivität von 5 Minuten wird der Benutzer automatisch ausgeloggt. Wird eine Page geladen, die Maus bewegt oder eine Taste gedrückt, so wird der Timer wieder auf 5 Minuten zurückgesetzt. Ist der Benutzer 5 Minuten am Stück inaktiv, wird er automatisch ausgeloggt.

Der Absolut Timeout ist auf 20 Minuten definiert. Die Session wird nach dieser Zeit automatisch beendet. Danach wird der Benutzer gezwungen, sich erneut einzuloggen. Hier werden dann auch die Cookies gelöscht.

```
<script>
var inactivityTime = function () {
  var time;
  window.onload = resetTimer;
  // DOM Events
  document.onmousemove = resetTimer;
  document.onkeypress = resetTimer;

  function logout() {
    // Send a POST request to the logout action
    fetch('/Identity/Account/Logout', {
      method: 'POST',
      headers: {
        'Content-Type': 'application/x-www-form-urlencoded',
        'RequestVerificationToken': "@GetAntiXsrfRequestToken()"
      },
      credentials: 'same-origin'
    }).then(response => {
      if (response.ok) {
        window.location.href = '/';
      } else {
        console.error('Logout failed');
      }
    }).catch(error => {
      console.error('Error:', error);
    });
  }

  function resetTimer() {
    clearTimeout(time); // 5 minutes
    time = setTimeout(logout, 300000);
  }
};

inactivityTime();
</script>
```

3 - Idle Timeout

```
builder.Services.ConfigureApplicationCookie(options =>
{
  options.SlidingExpiration = true;
  options.ExpireTimeSpan = TimeSpan.FromMinutes(20);
});
```

2 - Absolut Timeout

4 Applikationslog

4.1 Beschreibung

Ein Applikationslog speichert vorher definierte Aktionen, welche in der Applikation vorgenommen werden, beispielsweise in einer Datei oder in einer Tabelle der Datenbank ab. Dazu gehören diverse Informationen, damit immer genau nachvollziehbar ist, welcher Benutzer welche Aktion durchgeführt hat.

Die Basel Coin App führt einen Applikationslog in einer Tabelle der Datenbank. Dort werden Datum und Uhrzeit, Event Typ, der Benutzer und die vorgenommene Aktion hinterlegt. Alle Balance-Änderungen werden dort verzeichnet.

4.2 Code

Der ChangeTracker erkennt vorgenommene Änderungen an der Balance eines Benutzers. Daraufhin wird ein Audit-Eintrag mit allen relevanten Informationen erstellt und in die Datenbank eingefügt.

```
1 Verweis
private List<BalanceAudit> OnBeforeSaveChanges(string userId)
{
    ChangeTracker.DetectChanges();
    var auditEntries = new List<BalanceAudit>();

    foreach (var entry in ChangeTracker.Entries<Balance>())
    {
        if (entry.State == EntityState.Added || entry.State == EntityState.Modified || entry.State == EntityState.Deleted)
        {
            var current = (decimal)(entry.CurrentValues["Amount"] ?? 0);
            var before = entry.State == EntityState.Added ? 0 : (decimal)(entry.OriginalValues["Amount"] ?? 0);

            var amount = current - before;

            var auditEntry = new BalanceAudit
            {
                UserId = userId,
                BalanceId = entry.State == EntityState.Added ? 0 : entry.State == EntityState.Deleted ? null : entry.Entity.Id,
                Amount = amount,
                BalanceBefore = before,
                BalanceAfter = current,
                Action = entry.State == EntityState.Added ? "Created" :
                    entry.State == EntityState.Modified ? "Updated" :
                    entry.State == EntityState.Deleted ? "Deleted" : "Unknown",
                Date = DateTime.Now
            };
            auditEntries.Add(auditEntry);
        }
    }

    return auditEntries;
}
```

4 - Applikationslog

5 Input Validierung

https://cheatsheetseries.owasp.org/cheatsheets/Input_Validation_Cheat_Sheet.html

5.1 Beschreibung

Die Input Validierung ist eines der Grundkonzepte in der Applikationssicherheit. Dadurch wird jede Eingabe, die von einem Benutzer vorgenommen wird, auf seine Richtigkeit überprüft, bevor das System weiter mit dem Wert arbeitet. So kann beispielsweise schadhafte Eingaben vorgebeugt werden.

Die Basel Coin Applikation regelt die Input Validierung über Data Annotations. Diese bestimmen, welche Felder Pflicht sind und wie lange das Passwort beispielsweise sein muss. Des Weiteren wird ein Request Verification Token verwendet. Dieser wird bei jedem Seitenzugriff generiert. Dieser Wert wird beim Absenden mitgesendet und wenn die Response zurückgeschickt wird, wird dieser Wert nochmals mitgeschickt. Stimmt dieser nicht überein, wird ein Fehler ausgelöst und die Aktion abgebrochen. Das schützt den Benutzer davor, dass sein Input von Dritten abgeändert werden kann.

5.2 Code

Mithilfe des Entity Frameworks wird die Input Validierung durch Data Annotations umgesetzt. Diverse Felder werden als «Required» definiert. Auch die Struktur der Eingabe (Regex durch EF Core) wird geprüft, ob eine E-Mail-Adresse auch die Struktur einer solchen aufweist. Zusätzlich wird eine Mindest- und Maximallänge des Benutzernamens vorausgesetzt sowie ausschliesslich alphanumerische Zeichen.

```
1 Verweis
public class UserCreateRequest
{
    [Required(ErrorMessage = "UserName is required")]
    [StringLength(50, ErrorMessage = "Username must be between 3 and 50 characters", MinimumLength = 3)]
    [RegularExpression(@"^[a-zA-Z0-9]*$", ErrorMessage = "The username can only contain letters and numbers")]
    4 Verweise
    public string UserName { get; set; } = default!;

    [Required(ErrorMessage = "Password is required")]
    [MinLength(8, ErrorMessage = "Password must be at least 8 characters")]
    4 Verweise
    public string Password { get; set; } = default!;

    [Required(ErrorMessage = "Email is required")]
    [EmailAddress(ErrorMessage = "Invalid email address")]
    4 Verweise
    public string Email { get; set; } = default!;

    [Required(ErrorMessage = "Role is required")]
    4 Verweise
    public string Role { get; set; } = default!;
}
```

5 - Input Validierung

6 2FA

https://cheatsheetseries.owasp.org/cheatsheets/Multifactor_Authentication_Cheat_Sheet.html

6.1 Beschreibung

Die Zwei-Faktor-Authentifizierung ist ein weiterer starker Sicherheitsaspekt. Der Benutzer hat die Möglichkeit dies für sein Konto einzurichten – was dringend empfohlen wird. Dann genügt es nicht die E-Mail-Passwort-Kombination zu kennen, sondern ein Zugriff auf die damit verknüpfte Authentifizierungsapp muss auch vorhanden sein, um in den Account zu gelangen.

7 Abbildungsverzeichnis

1 - INJECTION PROTECTION	3
2 - ABSOLUT TIMEOUT	4
3 - IDLE TIMEOUT.....	4
4 - APPLIKATIONSLOG	5
5 - INPUT VALIDIERUNG	6