ipso! Bildung seit 1876

Basel Coin App

ICT-Modul 183 Applikationssicherheit implementieren

Arda Dursun, Bobby Bilali, David Reichert, Mahir Gönen

ipso! Bildung seit 1876

Inhaltsverzeichnis

1 Versionverzeichnis

Version	Autor	Datum	Änderung
1.0	Mahir Gönen	28.02.2024	Erstellung des Dokuments
1.1	Bobby Bilali	28.02.2024	Ergänzung des Dokuments
1.2	Arda Dursun	28.02.2024	Rechtschreibung- und Grammatikkorrektur
1.3	David Reichert	28.02.2024	Rechtschreibung- und Grammatikkorrektur
1.4	Mahir Gönen	29.02.2024	Korrektur für Abgabe

2 Executive Summary

Das Projektziel ist die Entwicklung eines Prototyps für die Applikation, eine Kryptowährung der Stadt Basel. Der Fokus liegt auf der Implementierung sicherheitskritischer Features wie Schutz gegen Injektion-Angriffe, effizientes Session Management, detailliertes Applikationslogging, Validierung aller Eingabefelder, und die Einrichtung von Benutzerrollen für Admins und User. Zudem wird ein automatisches Backup-System implementiert, das alle 24 Stunden generiert wird, um Datenintegrität und Wiederherstellbarkeit zu gewährleisten.

Die Sicherheitsarchitektur beinhaltet modernste Verfahren, darunter die Nutzung von JSON Web Tokens (JWT) mit HS256-Algorithmus für die Login-Implementierung, automatische Nutzersperrung nach dreimaliger fehlerhafter Anmeldung, und Implementierung von Idle- und Absolute-Timeouts für das Session Management. Ein umfassendes Logging-System basierend auf Serilog ermöglicht die präzise Überwachung und Analyse von Benutzeraktivitäten und Systemereignissen.

Input Validierung erfolgt sowohl auf dem Backend als auch auf dem Frontend, um die Sicherheit und Integrität der Daten zu gewährleisten. Eine klare Abgrenzung der Benutzerrollen durch eine spezifische Matrix ermöglicht eine detaillierte Kontrolle über Zugriffsrechte und ausführbare Aktionen innerhalb der Applikation.

Um die Applikation aktuell und sicher zu halten, werden die neuesten Versionen relevanter Bibliotheken und Frameworks verwendet, mit regelmässigen Updates zur Behebung von Sicherheitslücken und zur Performance-Verbesserung.
Dieser Ansatz sichert nicht nur die Applikation gegen eine Vielzahl von Bedrohungen ab, sondern trägt auch dazu bei, das Vertrauen in die Sicherheit und Zuverlässigkeit der Applikation zu stärken.

3 Ausgangslage

Das Team wurde beauftragt, die Applikation zu entwickeln, die in Zukunft, für die von der Stadt Basel ausgegebene Kryptowährung zur Zahlung verwendet werden kann. Der erste Schritt in diesem Projekt ist die Entwicklung eines kleinen Prototyps oder Proof of Concept (POC) für den PC. Dieser Prototyp soll es dem Projektteam der Stadt Basel ermöglichen, einen ersten Eindruck zu gewinnen und konkrete Ideen für die weitere Entwicklung zu sammeln.

Die Sicherheit der Applikation hat dabei oberste Priorität. Um das Vertrauen des Projektteams in die Applikation von Beginn an zu stärken, muss der Prototyp folgende sicherheitsrelevante Merkmale aufweisen: Schutz des Logins (Benutzername und Passwort) gegen Injektion-Angriffe, ein Session Management mit Idle- und absoluten Timeouts, ein Applikationslog, das Datum, Uhrzeit, Ereignistyp und Benutzeraktionen erfasst, sowie die Validierung aller Eingabefelder, um die Sicherheit der Benutzerinformationen zu gewährleisten.

Im Rahmen des POCs sollen zunächst nur zwei Benutzerrollen implementiert werden: Admin und User. Der Admin hat die Möglichkeit, neue Benutzer anzulegen und deren Kontostände abzufragen oder zu ändern. Benutzer können sich einloggen, ihren Kontostand einsehen und sich wieder abmelden.

Zusätzlich zu dieser Applikation ist eine knappe und präzise Dokumentation der Sicherheitsfeatures vorgesehen. Diese Dokumentation soll dem Projektteam der Stadt Basel einen Überblick über die implementierten Sicherheitsmassnahmen bieten und somit das Vertrauen in die Sicherheit der "Basel Coin" Applikation stärken.

4 Login-Implementation (JWT)

Die Implementierung des Logins in der Webapplikation erfolgt durch ein modernes und sicheres Verfahren, das jedem Nutzer ermöglicht, sich einzuloggen und dabei ein JSON Web Token (JWT) zu erzeugen. Dieser JWT ist ein wesentlicher Bestandteil der Sicherheitsarchitektur und folgt einer präzisen Struktur:

- **Algorithmus HS256:** Der Token basiert auf dem HS256-Algorithmus, einem weit verbreiteten Standard für die Erstellung von JWTs, der für seine Sicherheit und Zuverlässigkeit bekannt ist.

- **Payload:** Der Inhalt des Tokens wird sorgfältig ausgewählt und enthält Informationen wie die ID des Nutzers, dessen Benutzernamen, die Rolle innerhalb der Applikation und das Ausstellungsdatum (iat für "issued at").

- **Einzigartige Signatur:** Um den Schutz weiter zu verstärken, wird der JWT mit einer einzigartigen Signatur versehen, die zusätzlich mit einem HMACSHA256-Verfahren verschlüsselt ist. Dieses Verfahren kombiniert den Payload mit einem geheimen Schlüssel, um die Integrität und Authentizität des Tokens zu gewährleisten.

Der JWT wird im Browser-Speicher des Nutzers hinterlegt, was eine sichere und zugleich benutzerfreundliche Lösung darstellt. Er spielt eine zentrale Rolle bei der Autorisierung von Benutzeroperationen, die je nach Rolle des Nutzers variieren können. Zudem ist der Token mit einem Zeitstempel versehen, der seine Gültigkeit auf einen Tag begrenzt. Nach Ablauf dieses Zeitraums wird der Token ungültig, und der Nutzer muss sich erneut anmelden, um weiterhin Zugriff auf die Applikation zu haben. Diese Massnahme dient nicht nur der Sicherheit der Applikation, sondern auch dem Schutz der Nutzerdaten und der Prävention gegen unbefugten Zugriff.

4.1 Nutzersperrung

Die Sicherheitsmassnahme der Nutzersperrung in der Applikation wurde sorgfältig entwickelt, um die Sicherheit der Benutzerkonten zu gewährleisten. Sowohl Admins als auch reguläre Benutzer sind dieser Sicherheitsrichtlinie unterworfen. Wenn ein Benutzer dreimal hintereinander fehlerhafte Anmeldeinformationen eingibt, wird sein Konto automatisch gesperrt. Diese Massnahme dient dazu, unbefugte Zugriffsversuche auf Benutzerkonten zu verhindern und die Sicherheit der Applikation zu erhöhen.

Ein entscheidendes Element dieser Sicherheitsstrategie ist, dass nur ein Nutzer mit **Admin-Rechten** die Fähigkeit besitzt, ein gesperrtes Konto wieder zu **entsperren**. Dies stellt sicher, dass die Überprüfung und Freigabe gesperrter Konten kontrolliert und verantwortungsvoll gehandhabt wird, indem sie in die Hände von vertrauenswürdigen und autorisierten Personen gelegt wird. Durch diese gezielte Kontrolle über die Kontensperrung und -entsperrung wird ein zusätzliches Sicherheitsniveau eingeführt, das die Integrität der Benutzerdaten und die Sicherheit der gesamten Applikation stärkt.

```
if (employee.Password != loginDto.Password)
{
    employee.FailedLoginAttempts++;
    int remainingAttempts = MaxLoginAttempts - employee.FailedLoginAttempts;
    if (employee.FailedLoginAttempts >= MaxLoginAttempts)
    {
        employee.IsLocked = true;
        await _collection.UpdateOneAsync(
            Builders<AccountHolder>.Filter.Eq(emp => emp.Id, employee.Id),
            Builders<AccountHolder>.Update
                .Set(emp => emp.IsLocked, true)
                .Set(emp => emp.FailedLoginAttempts, employee.FailedLoginAttempts)
        );
        return new AuthenticationResult { IsAuthenticated = false, Message = "Benutzerkonto wurde wegen zu vieler fehlgeschlagener Versuche gesperrt.", RemainingAttempts = 0 };
    }
    else
    {
        await _collection.UpdateOneAsync(
            Builders<AccountHolder>.Filter.Eq(emp => emp.Id, employee.Id),
            Builders<AccountHolder>.Update.Set(emp => emp.FailedLoginAttempts, employee.FailedLoginAttempts)
        );
        return new AuthenticationResult { IsAuthenticated = false, Message = $"Falsches Passwort. Verbleibende Versuche: {remainingAttempts}", RemainingAttempts = remainingAttempts };
    }
}
```

4.2 Session Management mit Idle / absolut Timeout

Im Kontext des Projekts "Basel Coin" spielt das Session Management eine zentrale Rolle, insbesondere im Hinblick auf Sicherheitsaspekte. Ein effektives Session Management muss sicherstellen, dass Benutzersitzungen nach einer bestimmten Zeit der Inaktivität automatisch beendet werden (Idle Timeout) sowie nach einer absoluten Maximaldauer der Session (Absolute Timeout). Dieses Vorgehen ist entscheidend, um das Risiko von Session-Hijacking und anderen sicherheitsrelevanten Bedrohungen zu minimieren.

Ein Idle Timeout tritt in Kraft, wenn ein Benutzer für eine vorgegebene Zeitspanne, in diesem Fall **30 Sekunden, inaktiv bleibt**. Nach dieser Zeit wird die Sitzung automatisch beendet, und der Benutzer muss sich erneut anmelden, um Zugriff auf die Applikation zu erhalten. Diese Massnahme schützt vor dem Risiko, dass offene Sitzungen von Unbefugten missbraucht werden könnten.

```
claims.Add(new Claim(JwtRegisteredClaimNames.Iat, DateTime.UtcNow.ToString(), ClaimValueTypes.Integer64));

var creds = new SigningCredentials(_key, SecurityAlgorithms.HmacSha512Signature);

var tokenDescriptor = new SecurityTokenDescriptor
{
    Subject = new ClaimsIdentity(claims),
    Expires = DateTime.UtcNow.Add(_idleTimeout),
    NotBefore = DateTime.UtcNow,
    SigningCredentials = creds
};

var tokenHandler = new JwtSecurityTokenHandler();

var token = tokenHandler.CreateToken(tokenDescriptor);

return tokenHandler.WriteToken(token);
```

Darüber hinaus wird ein Absolute Timeout angewendet, das die Lebensdauer einer Sitzung auf eine maximale Dauer, hier auf **eine Stunde** gesetzt, begrenzt. Nach Ablauf dieser Zeit wird der JWT-Token ungültig, unabhängig von der Benutzeraktivität, was den Benutzer zwingt, sich erneut anzumelden. Diese Vorgehensweise hilft, das langfristige Risiko von kompromittierten Sitzungen zu reduzieren und trägt zur allgemeinen Sicherheit der Applikation bei.

Beide Timeout-Mechanismen sind flexibel gestaltet und können an die spezifischen Anforderungen und Sicherheitsrichtlinien der Applikation **angepasst werden**. Diese Anpassungsfähigkeit ermöglicht es, ein optimales Gleichgewicht zwischen Benutzerfreundlichkeit und Sicherheit zu finden, was für die Akzeptanz und den Schutz der Applikation von grosser Bedeutung ist.

5 Interne Logs

Für das Projekt wurde ein umfassendes Logging-System implementiert, das auf der Log-Bibliothek Serilog basiert. Dieses System ist entscheidend für die Überwachung und Sicherheit der Applikation, indem es verschiedene Arten von Logs erfasst und in einer Logdatei speichert. Es gibt zwei Hauptarten von Logs:

- **Serverlogs:** Diese dokumentieren wesentliche Vorgänge auf dem Server, einschliesslich eingehender Requests und aufgetretener Fehler. Serverlogs sind unerlässlich für das Verständnis des Verhaltens der Applikation unter realen Betriebsbedingungen und ermöglichen es dem Entwicklungsteam, auf potenzielle Probleme proaktiv zu reagieren.

- **Benutzerlogs:** Bei jedem Einloggen eines Benutzers werden spezifische Informationen erfasst, darunter die Uhrzeit des Logins und Benutzerinformationen. Diese Logs spielen eine wichtige Rolle beim Nachvollziehen von Benutzeraktivitäten und sind besonders wertvoll für die Sicherheitsanalyse, da sie helfen, unautorisierte Zugriffsversuche oder verdächtige Aktivitäten schnell zu identifizieren.

Durch die Nutzung von Serilog können detaillierte und strukturierte Logs erstellt werden, die nicht nur für die laufende Überwachung, sondern auch für die spätere Analyse von unschätzbarem Wert sind. Die Fähigkeit, Ereignisse genau zu protokollieren und schnell auf sie zugreifen zu können, ist ein kritischer Aspekt der Sicherheitsstrategie der Applikation, der dazu beiträgt, das System robust und sicher zu halten.

```
"Logging": {
  "LogLevel": {
    "Default": "Information",
    "Microsoft.AspNetCore": "Warning"
  }
},
"Serilog": {
  "Using": [ "Serilog.Sinks.File" ],
  "MinimumLevel": {
    "Default": "Information",
    "Override": {
      "Microsoft": "Warning",
      "System": "Error"
    }
  },
  "WriteTo": [
    {
      "Name": "File",
      "Args": {
        "path": "Logs/log-.txt",
        "rollingInterval": "Day",
        "outputTemplate": "{Timestamp:yyyy-MM-dd HH:mm:ss.fff zzz} [{Level:u3}] {Message:lj}{NewLine}{Exception}"
      }
    }
  ],
  "Enrich": [ "FromLogContext" ]
```

6 Input Validierung

Die Input Validierung ist ein kritischer Aspekt der Applikation, um die Sicherheit und Integrität der Daten zu gewährleisten. Dies wird durch eine zweistufige Validierungsstrategie erreicht:

Backend-Validierung: Die Validierung auf der Backend-Seite erfolgt durch Überprüfung der Datenobjektübertragungen (**DTOs**) und **Modelle**. Diese Schicht der Validierung stellt sicher, dass alle eingehenden Daten den erwarteten Formaten und Bedingungen entsprechen. Bei Nichtübereinstimmung der Daten werden diese Objekte abgefangen und nicht weiterverarbeitet, sondern direkt als Fehler an den Benutzer zurückgemeldet. Diese Massnahme verhindert, dass falsche oder manipulative Eingaben in die Datenbank gelangen oder die Datenbankstruktur verletzen.

Frontend-Validierung: Zusätzlich zur Backend-Validierung wird im Frontend, also im Bereich, der vom Benutzer genutzt wird, eine Validierung durchgeführt. Diese erfolgt mithilfe von regulären Ausdrücken (Regex) und umfasst Richtlinien wie die Länge des Benutzernamens und weitere Kriterien, die sicherstellen, dass die Eingaben bereits auf der Client-Seite den Anforderungen entsprechen.

Diese umfassende Validierungsstrategie spielt eine zentrale Rolle bei der Sicherung der Applikation gegen Eingabemanipulationen und trägt dazu bei, die Applikation und die darin gespeicherten Daten zu schützen. Indem sowohl auf der Client- als auch auf der Server-Seite rigoros validiert wird, wird ein hohes Mass an Sicherheit gewährleistet, dass für die Integrität und Zuverlässigkeit der Applikation unerlässlich ist.

In unserem Ansatz haben wir beschlossen, das Escaping direkt auf die Benutzernamen anzuwenden, die in die Datenbankanfragen eingehen. Dies geschieht durch die Nutzung der internen Sicherheitsmechanismen des BSON-Frameworks, welches von MongoDB verwendet wird. BSON, oder Binary JSON, bietet eine sichere Methode, um Daten zu serialisieren, einschließlich der Benutzereingaben, und schützt so vor Injection-Angriffen.

7 Benutzerrollen - Autorisierung

Für das Projekt wurden lediglich zwei Benutzerrollen vorgeschlagen: Admin und User. Um die Berechtigungen und Möglichkeiten dieser beiden Rollen klar zu definieren, wurde eine Matrix erstellt. Diese Matrix dient dazu, übersichtlich darzustellen, welche Aktionen von welcher Rolle durchgeführt werden können. Diese klare Abgrenzung der Rollen und ihrer Berechtigungen ist essentiell für die Sicherheit und Funktionalität der Applikation, da sie sicherstellt, dass nur autorisierte Benutzer bestimmte Aktionen ausführen können. Username ist Unique, also einzigartig

Rolle / **Operation**	**User**	**Admin**
Eigenen Kontostand abfragen	X	X
Einloggen	X	X
Benutzer anlegen		X
Kontostände abfragen		X
Benutzer entsperren		X

8 Backup

Die Implementierung eines automatischen Backup-Systems, das alle 24 Stunden generiert wird, stellt eine wichtige Sicherheitsmassnahme für jedes IT-System oder jede Anwendung dar. Diese Strategie bietet mehrere Vorteile im Hinblick auf die Sicherheit und Wiederherstellungsfähigkeit:

- **Datenwiederherstellung:** Im Falle eines Datenverlustes durch Hardwarefehler, menschliche Fehler, Softwarebugs oder Cyberangriffe ermöglicht das regelmässige Backup eine Wiederherstellung der Daten bis zum letzten Sicherungszeitpunkt. Dies minimiert den Datenverlust und stellt die Kontinuität des Betriebs sicher.

- **Schutz vor Manipulation:** Bei Verdacht auf Datenmanipulation, sei es durch interne oder externe Akteure, ermöglicht das Backup die Wiederherstellung einer unveränderten Version der Daten. Dies ist besonders wichtig, um die Integrität von sensiblen Informationen zu gewährleisten.

- **Anpassbare Backup-Zeiten:** Die Fähigkeit, die Häufigkeit der Backups anzupassen, bietet Flexibilität, um den spezifischen Anforderungen und Risikoprofilen verschiedener Anwendungen oder Daten zu entsprechen. Für besonders kritische Systeme könnten häufigere Backups notwendig sein, während bei weniger kritischen Daten längere Intervalle ausreichend sein können. Im Proof of Concept wird alle 24 Stunden ein Backup generiert.

- **Mehrschichtige Sicherheit:** Das automatische Backup dient als zusätzliche Sicherheitsebene (Layer) in einem umfassenden Sicherheitskonzept. Die Kombination aus vorbeugenden Massnahmen (wie Firewalls und Antivirus-Software) und reaktiven Massnahmen (wie Backups für die Wiederherstellung) bildet ein robusteres Sicherheitssystem.

9 Bibliotheken, Frameworks

Die Verwendung der aktuellen Versionen von Bibliotheken und Frameworks in Softwareprojekten ist eine gute Praxis, um die Sicherheit und Stabilität der Anwendungen zu gewährleisten. Aktuellste Versionen enthalten oft Patches für Sicherheitslücken (CVEs - Common Vulnerabilities and Exposures), Verbesserungen der Leistung und neue Funktionen. Es ist wichtig, regelmässig nach Updates zu suchen und Abhängigkeiten zu aktualisieren, um von diesen Verbesserungen zu profitieren. Unten sind alle Bibliotheken und Frameworks, die in diesem Projekt verwendet wurden, aufgelistet. Die Versionen sind aktuell (Stand: 28.02.2024).

Bibliothek/Framework	Version
AutoMapper.Extensions.Microsoft.DependencyInjection	12.0.1
MongoDB.Bson	2.23.1
MongoDB.Driver	2.23.1
Serilog	3.1.1
Serilog.AspNetCore	7.0.0
Swashbuckle.AspNetCore	6.5.0
Microsoft.AspNetCore.Authentication.JwtBearer	8.0.1
Microsoft.AspNetCore.OpenApi	8.0.1

10 Quellenverzeichnis

https://owasp.org
https://www.cve.org