Na podstawie specyfikacji **7.5.10 Card lock/unlock operation** przygotujmy test jednostkowy dla komponentu odpowiedzialnego za interpretację i wykonanie komendy **Card Lock/Unlock**, przesyłanej jako blok danych.

Zakładamy klasę `CardLockHandler`, która interpretuje strukturę komendy i zmienia stan karty.

🔧 Założenia:

- Komenda zawiera flagę `SET_PWD` i hasło `0xCAFE`.
- `PWD_LEN` wynosi `2`.
- Hasło ma być zapisane i karta powinna zostać zablokowana.
- Karta ma właściwość `IsLocked`.

✅ Uproszczona klasa Mock

To atrapa, która symuluje działanie prawdziwej karty.

```
public class MockCard
{
    private byte[] _storedPassword = Array.Empty<byte>();
    public byte[] StoredPassword => _storedPassword.ToArray();
    public int PasswordLength => _storedPassword.Length;

    public bool IsLocked { get; private set; } = false;

    public void SetPassword(byte[] pwd)
    {
        _storedPassword = pwd.ToArray();
        IsLocked = true;
    }

    public bool TryUnlock(byte[] pwd)
    {
        if (_storedPassword.SequenceEqual(pwd))
        {
            IsLocked = false;
            return true;
```

```
        }

        return false;
    }

    public bool ClearPasswordIfCorrect(byte[] pwd)
    {
        if (_storedPassword.SequenceEqual(pwd))
        {
            _storedPassword = Array.Empty<byte>();
            IsLocked = false;
            return true;
        }

        return false;
    }
}
```

🧪 3. Przykładowy test: poprawne odblokowanie

```
[Fact]
public void TryUnlock_WithCorrectPassword_ChangesStateToUnlocked()
{
    var card = new MockCard();
    card.SetPassword(new byte[] { 0xAA, 0xBB });

    var success = card.TryUnlock(new byte[] { 0xAA, 0xBB });

    Assert.True(success);
    Assert.Equal(CardLockState.Unlocked, card.LockState);
}
```

🧪 4. Przykładowy test: błędne hasło

```
[Fact]
public void TryUnlock_WithWrongPassword_ChangesStateToError()
{
```

```
    var card = new MockCard();
    card.SetPassword(new byte[] { 0xAA, 0xBB });

    var success = card.TryUnlock(new byte[] { 0x00, 0x00 });

    Assert.False(success);
    Assert.Equal(CardLockState.Error, card.LockState);
}
```

🧰 `CardLockHandler` (wersja bazowa)

```
public class CardLockHandler
{
    private readonly MockCard _card;

    public CardLockHandler(MockCard card)
    {
        _card = card;
    }

    public void HandleCommand(byte[] block)
    {
        byte flags = block[0];
        byte pwdLen = block[1];
        byte[] pwd = block.Skip(2).Take(pwdLen).ToArray();

        // SET_PWD
        if ((flags & 0b00000001) != 0)
        {
            _card.SetPassword(pwd);
        }

        // CLR_PWD
        if ((flags & 0b00000010) != 0)
        {
            _card.ClearPasswordIfCorrect(pwd);
        }

        // LOCK_UNLOCK
        if ((flags & 0b00000100) != 0)
        {
            _card.TryUnlock(pwd);
        }
```

```
        // ERASE (jeśli używamy)
        if ((flags & 0b00001000) != 0)
        {
            _card.RequestErase();
        }
    }
}
```

Bit	Flaga	Znaczenie
0	`SET_PWD`	Ustawienie hasła
1	`CLR_PWD`	Usunięcie hasła (jeśli poprawne)
2	`LOCK_UNLOCK`	Próba odblokowania kartą
3	`ERASE`	Żądanie skasowania danych

🧪 1. Test: Odblokowanie karty poprawnym hasłem

```
[Fact]
public void UnlockCommand_WithCorrectPassword_UnlocksCard()
{
    // Arrange
    var card = new MockCard();
    card.SetPassword(new byte[] { 0xAB, 0xCD });
    var handler = new CardLockHandler(card);

    byte[] commandBlock = new byte[]
    {
        0b00000100, // Byte 0: LOCK_UNLOCK
        0x02,       // Byte 1: PWD_LEN = 2
        0xAB, 0xCD  // Byte 2–3: correct password
    };

    // Act
    handler.HandleCommand(commandBlock);

    // Assert
    Assert.False(card.IsLocked);
    Assert.True(card.WasUnlocked);
}
```

💾 2. Test: zapis danych na odblokowanej karcie

```
[Fact]
public void WriteData_WhenCardIsUnlocked_DataIsStored()
{
    // Arrange
    var card = new MockCard();
    card.SetPassword(new byte[] { 0x01, 0x02 });

    // Odblokuj kartę poprawnym hasłem
    card.TryUnlock(new byte[] { 0x01, 0x02 });

    // Act
    card.WriteData("SECRET");

    // Assert
    Assert.Equal("SECRET", card.StoredData);
}
```

🧱 3. Uzupełniona klasa `MockCard` o zapis

```
public class MockCard
{
    private byte[] _storedPassword = Array.Empty<byte>();
    public int PasswordLength => _storedPassword.Length;

    public bool IsLocked { get; private set; } = true;
    public bool WasUnlocked { get; private set; } = false;

    public string StoredData { get; private set; } = string.Empty;

    public void SetPassword(byte[] pwd)
    {
        _storedPassword = pwd;
        IsLocked = true;
        WasUnlocked = false;
    }

    public bool TryUnlock(byte[] pwd)
    {
```

```
        if (_storedPassword.SequenceEqual(pwd))
        {
            IsLocked = false;
            WasUnlocked = true;
            return true;
        }

        return false;
    }

    public void WriteData(string data)
    {
        if (IsLocked)
        {
            throw new InvalidOperationException("Card is locked.");
        }

        StoredData = data;
    }
}
```

🧰 `LockCommandBuilder` — Budowniczy

Zastosowanie **Wzorca Budowniczego (Builder Pattern)** ułatwi konstruowanie ramki komendy `Lock/Unlock`, pozwalając jasno i czytelnie budować różne warianty (SET_PWD, UNLOCK, CLR_PWD itd.).

```
public class LockCommandBuilder
{
    private byte _flags = 0;
    private readonly List<byte> _password = new();

    public LockCommandBuilder SetPassword(params byte[] pwd)
    {
        _flags |= 0b00000001; // SET_PWD
        _password.Clear();
        _password.AddRange(pwd);
        return this;
    }

    public LockCommandBuilder ClearPassword(params byte[] pwd)
    {
```

```
        _flags |= 0b00000010; // CLR_PWD
        _password.Clear();
        _password.AddRange(pwd);
        return this;
    }

    public LockCommandBuilder LockOrUnlock(params byte[] pwd)
    {
        _flags |= 0b00000100; // LOCK_UNLOCK
        _password.Clear();
        _password.AddRange(pwd);
        return this;
    }

    public LockCommandBuilder Erase()
    {
        _flags |= 0b00001000; // ERASE
        return this;
    }

    public byte[] Build()
    {
        var result = new List<byte> { _flags, (byte)_password.Count };
        result.AddRange(_password);
        return result.ToArray();
    }
}
```

🧪 Przykład użycia w teście

```
[Fact]
public void UnlockCommand_WithCorrectPassword_UnlocksCard()
{
    // Arrange
    var card = new MockCard();
    card.SetPassword(new byte[] { 0xAA, 0xBB });

    var command = new LockCommandBuilder()
        .LockOrUnlock(0xAA, 0xBB)
        .Build();

    var handler = new CardLockHandler(card);
```

```
    // Act
    handler.HandleCommand(command);

    // Assert
    Assert.False(card.IsLocked);
}
```

🧪 Inny przykład: SET_PWD

```
[Fact]
public void SetPasswordCommand_SavesPasswordAndLocksCard()
{
    // Arrange
    var card = new MockCard();
    var command = new LockCommandBuilder()
        .SetPassword(0x11, 0x22)
        .Build();

    var handler = new CardLockHandler(card);

    // Act
    handler.HandleCommand(command);

    // Assert
    Assert.True(card.IsLocked);
    Assert.Equal(new byte[] { 0x11, 0x22 }, card.StoredPassword);
}
```

🎯 Korzyści

- **Czytelność** – nie trzeba ręcznie obliczać bajtów ramki.
- **Bezpieczeństwo** – centralne miejsce dla poprawnego kodowania flag.
- **Elastyczność** – łatwo dodać nowe opcje (`Erase` , `Combined` , itp.).

🔧 Zaktualizowany `LockCommandBuilder`

Rozszerzmy `LockCommandBuilder`, aby umożliwić **kombinowanie flag** (np. `SET_PWD` + `LOCK_UNLOCK`) i dodawanie hasła **wspólnego dla wszystkich trybów** (co jest zgodne z formatem, gdzie hasło i jego długość są podawane raz niezależnie od liczby flag).

```
public class LockCommandBuilder
{
    private byte _flags = 0;
    private readonly List<byte> _password = new();

    public LockCommandBuilder WithSetPassword()
    {
        _flags |= 0b00000001; // SET_PWD
        return this;
    }

    public LockCommandBuilder WithClearPassword()
    {
        _flags |= 0b00000010; // CLR_PWD
        return this;
    }

    public LockCommandBuilder WithLockOrUnlock()
    {
        _flags |= 0b00000100; // LOCK_UNLOCK
        return this;
    }

    public LockCommandBuilder WithErase()
    {
        _flags |= 0b00001000; // ERASE
        return this;
    }

    public LockCommandBuilder WithPassword(params byte[] pwd)
    {
        _password.Clear();
        _password.AddRange(pwd);
        return this;
    }

    public byte[] Build()
    {
        var result = new List<byte> { _flags, (byte)_password.Count };
```

```
        result.AddRange(_password);
        return result.ToArray();
    }
}
```

🧪 Przykład testu: `SET_PWD` + `LOCK_UNLOCK` z hasłem

```
[Fact]
public void SetPasswordAndLockCard_CombinedFlags_LocksCard()
{
    // Arrange
    var card = new MockCard();
    var command = new LockCommandBuilder()
        .WithSetPassword()
        .WithLockOrUnlock()
        .WithPassword(0x55, 0x66)
        .Build();

    var handler = new CardLockHandler(card);

    // Act
    handler.HandleCommand(command);

    // Assert
    Assert.True(card.IsLocked);
    Assert.Equal(new byte[] { 0x55, 0x66 }, card.StoredPassword);
}
```

🧪 Przykład: `CLR_PWD` + `LOCK_UNLOCK` (czyli odblokuj i wyczyść)

```
[Fact]
public void ClearPasswordAfterUnlock_CombinedCommand_ClearsPassword()
{
    // Arrange
    var card = new MockCard();
    card.SetPassword(new byte[] { 0x99, 0x88 });
```

```
    var command = new LockCommandBuilder()
        .WithLockOrUnlock()
        .WithClearPassword()
        .WithPassword(0x99, 0x88)
        .Build();

    var handler = new CardLockHandler(card);

    // Act
    handler.HandleCommand(command);

    // Assert
    Assert.False(card.IsLocked);
    Assert.Equal(0, card.PasswordLength);
}
Przykład: `CLR_PWD` + `LOCK_UNLOCK` (czyli odblokuj i wyczyść)
```

🔧 Możesz też dodać metodę `WithAllFlags()` np. do testów ekstremalnych:

```
public LockCommandBuilder WithAllFlags()
{
    _flags = 0x0F; // 00001111
    return this;
}
```

🔧 Refaktoring z użyciem Chain of Responsibility

- Każdy handler obsługuje **jedną konkretną flagę**,
  np. `SET_PWD_Handler`, `LOCK_UNLOCK_Handler`.
- Jeśli flaga nie jest ustawiona, przekazuje dalej.
- Można łatwo dołączyć nowe typy operacji.

🧰 Interfejs `ILockCommandHandler`

```
public interface ILockCommandHandler
{
    void SetNext(ILockCommandHandler next);
    void Handle(byte flags, byte[] password, MockCard card);
}
```

🧰 Klasa bazowa `LockCommandHandlerBase`

```
public abstract class LockCommandHandlerBase : ILockCommandHandler
{
    private ILockCommandHandler _next;

    public void SetNext(ILockCommandHandler next)
    {
        _next = next;
    }

    public void Handle(byte flags, byte[] password, MockCard card)
    {
        if (!Process(flags, password, card) && _next != null)
        {
            _next.Handle(flags, password, card);
        }
    }

    protected abstract bool Process(byte flags, byte[] password, MockCard
card);
}
```

🧩 Konkretne handlery

`SetPasswordHandler`

```
public class SetPasswordHandler : LockCommandHandlerBase
{
    protected override bool Process(byte flags, byte[] password, MockCard
card)
    {
```

```
        if ((flags & 0b00000001) != 0)
        {
            card.SetPassword(password);
            return true;
        }
        return false;
    }
}
```

`UnlockHandler`

```
public class UnlockHandler : LockCommandHandlerBase
{
    protected override bool Process(byte flags, byte[] password, MockCard
card)
    {
        if ((flags & 0b00000100) != 0)
        {
            card.TryUnlock(password);
            return true;
        }
        return false;
    }
}
```

`ClearPasswordHandler`

```
public class ClearPasswordHandler : LockCommandHandlerBase
{
    protected override bool Process(byte flags, byte[] password, MockCard
card)
    {
        if ((flags & 0b00000010) != 0)
        {
            card.ClearPasswordIfCorrect(password);
            return true;
        }
        return false;
    }
}
```

🧠 Główna klasa `CardLockHandler`

```
public class CardLockHandler
{
    private readonly ILockCommandHandler _pipeline;

    public CardLockHandler(MockCard card)
    {
        // Zbuduj łańcuch odpowiedzialności
        var setPwd = new SetPasswordHandler();
        var unlock = new UnlockHandler();
        var clearPwd = new ClearPasswordHandler();

        setPwd.SetNext(unlock);
        unlock.SetNext(clearPwd);

        _pipeline = setPwd;
    }

    public void HandleCommand(byte[] block)
    {
        byte flags = block[0];
        int pwdLen = block[1];
        byte[] pwd = block.Skip(2).Take(pwdLen).ToArray();

        _pipeline.Handle(flags, pwd, card);
    }
}
```

🔧 `MockCard` – dodaj metodę do usuwania hasła

```
public void ClearPasswordIfCorrect(byte[] pwd)
{
    if (_storedPassword.SequenceEqual(pwd))
    {
        _storedPassword = Array.Empty<byte>();
        IsLocked = false;
        WasUnlocked = true;
    }
```

```
}
```

✅ Zalety refaktoryzacji

Cecha	Korzyść
Brak `if` ów w głównym handlerze	Czystszy kod
Łatwe dodanie nowych funkcji	np. `ERASE`, `LOGGING`
Możliwość testowania każdego handlera osobno	Single Responsibility

🧪 1. Test dla `SetPasswordHandler`

```
public class SetPasswordHandlerTests
{
    [Fact]
    public void Handle_WithSetPwdFlag_SetsPasswordAndLocksCard()
    {
        // Arrange
        var handler = new SetPasswordHandler();
        var card = new MockCard();
        byte flags = 0b00000001; // SET_PWD
        byte[] password = { 0x12, 0x34 };

        // Act
        handler.Handle(flags, password, card);

        // Assert
        Assert.True(card.IsLocked);
        Assert.Equal(password, card.StoredPassword);
    }

    [Fact]
    public void Handle_WithoutSetPwdFlag_DoesNotChangeCard()
    {
        var handler = new SetPasswordHandler();
        var card = new MockCard();
        byte flags = 0b00000100; // LOCK_UNLOCK only
        byte[] password = { 0x12, 0x34 };

        handler.Handle(flags, password, card);
```

```
        Assert.Empty(card.StoredPassword);
    }
}
```

🧪 2. Test dla `UnlockHandler`

```
public class UnlockHandlerTests
{
    [Fact]
    public void Handle_WithCorrectPassword_UnlocksCard()
    {
        var handler = new UnlockHandler();
        var card = new MockCard();
        card.SetPassword(new byte[] { 0xCA, 0xFE });

        byte flags = 0b00000100; // LOCK_UNLOCK
        byte[] password = { 0xCA, 0xFE };

        handler.Handle(flags, password, card);

        Assert.False(card.IsLocked);
        Assert.True(card.WasUnlocked);
    }

    [Fact]
    public void Handle_WithWrongPassword_DoesNotUnlockCard()
    {
        var handler = new UnlockHandler();
        var card = new MockCard();
        card.SetPassword(new byte[] { 0xAB, 0xCD });

        byte flags = 0b00000100;
        byte[] password = { 0x00, 0x11 };

        handler.Handle(flags, password, card);

        Assert.True(card.IsLocked);
        Assert.False(card.WasUnlocked);
    }
}
```

🧪 3. Test dla `ClearPasswordHandler`

```
public class ClearPasswordHandlerTests
{
    [Fact]
    public void Handle_WithCorrectPassword_ClearsPassword()
    {
        var handler = new ClearPasswordHandler();
        var card = new MockCard();
        card.SetPassword(new byte[] { 0xAA, 0xBB });

        byte flags = 0b00000010; // CLR_PWD
        byte[] password = { 0xAA, 0xBB };

        handler.Handle(flags, password, card);

        Assert.False(card.IsLocked);
        Assert.Equal(0, card.PasswordLength);
    }

    [Fact]
    public void Handle_WithWrongPassword_DoesNotClearPassword()
    {
        var handler = new ClearPasswordHandler();
        var card = new MockCard();
        card.SetPassword(new byte[] { 0xAA, 0xBB });

        byte flags = 0b00000010;
        byte[] password = { 0x11, 0x22 };

        handler.Handle(flags, password, card);

        Assert.True(card.IsLocked);
        Assert.Equal(2, card.PasswordLength);
    }
}
```

✅ Zalety refaktoryzacji

- Każdy handler można testować niezależnie.

🧭 Stany logiczne wynikające ze specyfikacji

Stan	Opis
`Uninitialized`	Karta nie ma ustawionego hasła (`PWD_LEN == 0`)
`Locked`	Hasło jest ustawione, karta jest zablokowana i wymaga odblokowania
`Unlocked`	Karta została odblokowana poprawnym hasłem
`Cleared`	Hasło zostało poprawnie usunięte (komenda `CLR_PWD`)
`ErasePending`	Oczekuje na wykonanie komendy `ERASE`
`EraseDone` (opc.)	Dane skasowane w wyniku `ERASE`
`Error` (opc.)	Próba odblokowania z błędnym hasłem itp. (zależnie od implementacji)

🧠 Mapa przejść (uproszczona)

```
Uninitialized
   │
SET_PWD (hasło)
   ↓
Locked ── UNLOCK (hasło OK) ──→ Unlocked
   │                                │
   └── CLR_PWD (hasło OK) ──────────┘
   │
   └── ERASE ──→ EraseDone
```

✅ 1. Zmieniamy klasę `MockCard` na używającą `enum`:

```
public enum CardLockState
{
    Uninitialized,  // brak hasła
    Locked,         // hasło ustawione, karta zablokowana
    Unlocked,       // odblokowana poprawnym hasłem
    Cleared,        // hasło usunięte
    ErasePending,   // żądanie ERASE
```

```
    EraseDone,      // dane skasowane
    Error           // np. zły PIN
}
```

```
public class MockCard
{
    private byte[] _storedPassword = Array.Empty<byte>();
    public byte[] StoredPassword => _storedPassword.ToArray();
    public int PasswordLength => _storedPassword.Length;
    public CardLockState LockState { get; private set; } =
CardLockState.Uninitialized;

    public void SetPassword(byte[] pwd)
    {
        _storedPassword = pwd.ToArray();
        LockState = CardLockState.Locked;
    }

    public bool TryUnlock(byte[] pwd)
    {
        if (_storedPassword.SequenceEqual(pwd))
        {
            LockState = CardLockState.Unlocked;
            return true;
        }

        LockState = CardLockState.Error;
        return false;
    }

    public void ClearPasswordIfCorrect(byte[] pwd)
    {
        if (_storedPassword.SequenceEqual(pwd))
        {
            _storedPassword = Array.Empty<byte>();
            LockState = CardLockState.Cleared;
        }
        else
        {
            LockState = CardLockState.Error;
        }
    }

    public void RequestErase()
    {
```

```
        LockState = CardLockState.ErasePending;
    }

    public void PerformErase()
    {
        _storedPassword = Array.Empty<byte>();
        LockState = CardLockState.EraseDone;
    }
}
```

🧪 3. Przykładowy test: poprawne odblokowanie

```
[Fact]
public void TryUnlock_WithCorrectPassword_ChangesStateToUnlocked()
{
    var card = new MockCard();
    card.SetPassword(new byte[] { 0xAA, 0xBB });

    var success = card.TryUnlock(new byte[] { 0xAA, 0xBB });

    Assert.True(success);
    Assert.Equal(CardLockState.Unlocked, card.LockState);
}
```

🧪 4. Przykładowy test: błędne hasło

```
[Fact]
public void TryUnlock_WithWrongPassword_ChangesStateToError()
{
    var card = new MockCard();
    card.SetPassword(new byte[] { 0xAA, 0xBB });

    var success = card.TryUnlock(new byte[] { 0x00, 0x00 });

    Assert.False(success);
    Assert.Equal(CardLockState.Error, card.LockState);
}
```

🧪 5. Przykładowy test pipeline: `SET_PWD` + `LOCK_UNLOCK`

```
[Fact]
public void Pipeline_SetPasswordAndLock_CardIsLocked()
{
    var card = new MockCard();
    var handler = new CardLockHandler(card);

    var command = new LockCommandBuilder()
        .WithSetPassword()
        .WithLockOrUnlock()
        .WithPassword(0x12, 0x34)
        .Build();

    handler.HandleCommand(command);

    Assert.Equal(CardLockState.Locked, card.LockState);
    Assert.Equal(new byte[] { 0x12, 0x34 }, card.StoredPassword);
}
```

🧪 6. Przykład `CLR_PWD`

```
[Fact]
public void Pipeline_ClearPasswordWithCorrectPassword_TransitionsToCleared()
{
    var card = new MockCard();
    card.SetPassword(new byte[] { 0xCA, 0xFE });

    var handler = new CardLockHandler(card);

    var command = new LockCommandBuilder()
        .WithLockOrUnlock()
        .WithClearPassword()
        .WithPassword(0xCA, 0xFE)
        .Build();

    handler.HandleCommand(command);

    Assert.Equal(CardLockState.Cleared, card.LockState);
    Assert.Empty(card.StoredPassword);
}
```

✅ 2. Licznik nieudanych prób (`FailedAttempts`)

```
public int FailedAttempts { get; private set; } = 0;
public int MaxAttempts { get; set; } = 3;

public bool TryUnlock(byte[] pwd)
{
    if (_storedPassword.SequenceEqual(pwd))
    {
        LockState = CardLockState.Unlocked;
        FailedAttempts = 0;
        return true;
    }

    FailedAttempts++;
    if (FailedAttempts >= MaxAttempts)
    {
        LockState = CardLockState.Error;
    }
    else
    {
        LockState = CardLockState.Locked;
    }

    return false;
}
```

🔧 `AuditLogHandler` – rejestr operacji

Interfejs pomocniczy

```
public interface IAuditLogger
{
    void Log(string message);
}
```

Prosty logger do testów

```
public class TestAuditLogger : IAuditLogger
{
```

```
    public List<string> Entries { get; } = new();
    public void Log(string message) => Entries.Add(message);
}
```

- Możesz użyć standardowego `ILogger`

Audit handler

```
public class AuditLogHandler : LockCommandHandlerBase
{
    private readonly IAuditLogger _logger;

    public AuditLogHandler(IAuditLogger logger)
    {
        _logger = logger;
    }

    protected override bool Process(byte flags, byte[] password, MockCard
card)
    {
        var ops = new List<string>();
        if ((flags & 0b00000001) != 0) ops.Add("SET_PWD");
        if ((flags & 0b00000010) != 0) ops.Add("CLR_PWD");
        if ((flags & 0b00000100) != 0) ops.Add("LOCK_UNLOCK");
        if ((flags & 0b00001000) != 0) ops.Add("ERASE");

        _logger.Log($"Flags: {string.Join(",", ops)}, PWD_LEN:
{password.Length}");

        return false; // zawsze przepuszczamy dalej
    }
}
```

🔄 Aktualizacja `CardLockHandler`

```
public class CardLockHandler
{
    private readonly ILockCommandHandler _pipeline;

    public CardLockHandler(MockCard card, IAuditLogger logger = null)
    {
        var audit = new AuditLogHandler(logger ?? new TestAuditLogger());
        var setPwd = new SetPasswordHandler();
```

```
        var unlock = new UnlockHandler();
        var clearPwd = new ClearPasswordHandler();
        var erase = new EraseHandler();

        audit.SetNext(setPwd);
        setPwd.SetNext(unlock);
        unlock.SetNext(clearPwd);
        clearPwd.SetNext(erase);

        _pipeline = audit;
    }

    public void HandleCommand(byte[] commandBlock)
    {
        byte flags = commandBlock[0];
        byte pwdLen = commandBlock[1];
        byte[] pwd = commandBlock.Skip(2).Take(pwdLen).ToArray();

        _pipeline.Handle(flags, pwd, card);
    }
}
```

🧠 Zastosowanie maszyny stanów skończonych *(Finite State Machine)* do obsługi karty

Dodaj bibliotekę:

```
dotnet add package Stateless
```

⚙️ Konfiguracja maszyny

```
private StateMachine<CardState, CardTrigger> _fsm;

public void ConfigureStateMachine()
{
    _fsm = new StateMachine<CardState, CardTrigger>(() => LockState, s =>
LockState = s);

    _fsm.Configure(CardState.Uninitialized)
        .Permit(CardTrigger.SetPassword, CardState.Locked);

    _fsm.Configure(CardState.Locked)
        .PermitIf(CardTrigger.Unlock, CardState.Unlocked, () =>
```

```
IsPasswordCorrect)
        .PermitIf(CardTrigger.ClearPassword, CardState.Cleared, () =>
IsPasswordCorrect)
        .Permit(CardTrigger.RequestErase, CardState.ErasePending)
        .PermitIf(CardTrigger.FailedUnlock, CardState.Error, () =>
FailedAttempts >= MaxAttempts);

    _fsm.Configure(CardState.Unlocked)
        .Permit(CardTrigger.ClearPassword, CardState.Cleared);

    _fsm.Configure(CardState.ErasePending)
        .Permit(CardTrigger.ConfirmErase, CardState.EraseDone);
}
```

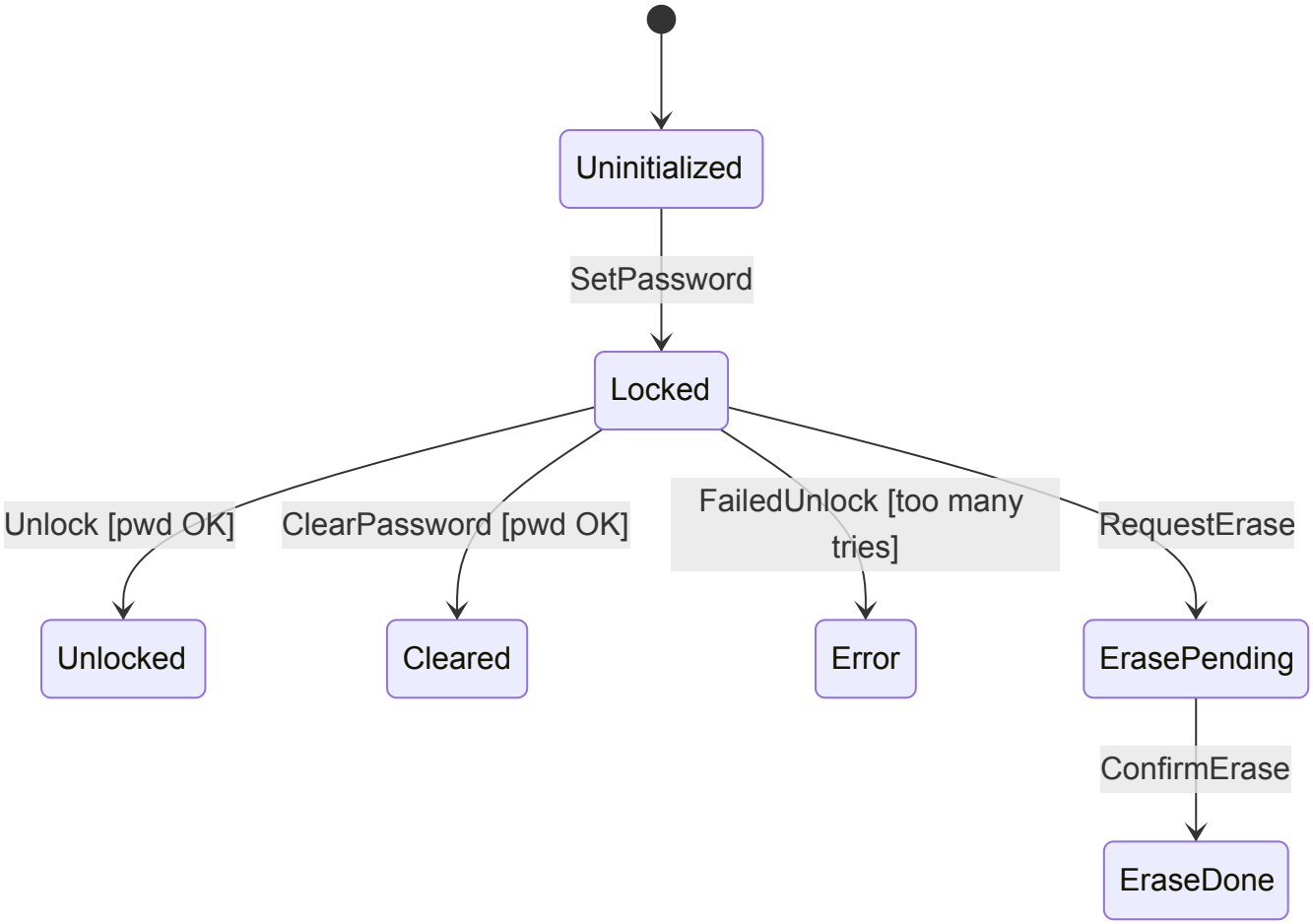
🔧 Działanie w `MockCard`

Zamiast logiki rozproszonej w `TryUnlock`, `ClearPasswordIfCorrect`, `SetPassword`, zastosujemy maszynę stanów skończonych fsm:

```
public void TryUnlock(byte[] pwd)
{
    if (_storedPassword.SequenceEqual(pwd))
    {
        _fsm.Fire(CardTrigger.Unlock);
        FailedAttempts = 0;
    }
    else
    {
        FailedAttempts++;
        _fsm.Fire(CardTrigger.FailedUnlock);
    }
}
```

🧩 Wizualizacja maszyny stanów w formacie Mermaid

```
public string Graph => MermaidGraph.Format(_stateMachine.GetInfo());
```



✅ Przykład zastosowania

```
public class MockCardWithStateMachine
{
    private readonly StateMachine<CardState, CardTrigger> _fsm;

    public string Graph => MermaidGraph.Format(_fsm.GetInfo());

    public MockCardWithStateMachine()
    {
        _fsm = new StateMachine<CardState, CardTrigger>
(CardState.Uninitialized);

        _fsm.Configure(CardState.Uninitialized)
            .Permit(CardTrigger.SetPassword, CardState.Locked);

        _fsm.Configure(CardState.Locked)
            .Permit(CardTrigger.Unlock, CardState.Unlocked)
            .Permit(CardTrigger.ClearPassword, CardState.Cleared)
```

```
            .Permit(CardTrigger.RequestErase, CardState.ErasePending);

        _fsm.Configure(CardState.ErasePending)
            .Permit(CardTrigger.ConfirmErase, CardState.EraseDone);
    }
}
```

✅ Co zyskujesz?

- przejścia są **łatwe do wizualizacji** (Stateless może generować diagramy)
- możesz dodać **akcje wejścia/wyjścia** (`OnEntry` , `OnExit`)
- masz **twardą kontrolę stanów**: brak nielegalnych przejść