



Univerzitet u Sarajevu
Elektrotehnički fakultet
Sarajevo

Projektni zadatak – II dio

Predmetni asistent: Mr. dipl. ing. Neira Novalić
Ime i prezime studenta: Nedim Bečić
Broj indeksa: 19274

1. McCabe metrika i kontrolni graf

Prvi zadatak je zahtijevao računanje McCabe metrike – ciklomatske kompleksnosti, te crtanje kontrolnog grafa za prikaz svih puteva unutar odabrane metode za analizu.

Odabrana metoda: CitajLozinku

Svrha ove metode u ranije kreiranoj konzolnoj aplikaciji je da sakrije korisnički unos odabrane lozinke prilikom kreiranja i prijavljivanja u aplikaciju, doprinoseći sigurnosti i očuvanju integriteta korisničkih podataka.

CitajLozinku je metoda bez parametara čija povratna vrijednost predstavlja string, preciznije sakrivenu korisničku šifru. Ukoliko prilikom kreiranja lozinke korisnik unosi karaktere "password123", prikaz na ekranu će biti "*****".

Ova metoda je odabrana zbog svoje složenosti, sadrži WHILE petlju i više IF iskaza, što je čini pogodnom za analizu putem McCabe metrike, ali i brojnih ostalih tehnika za *white box* testiranje.

```
public string CitajLozinku()
{
    string lozinka = string.Empty;
    ConsoleKey key;
    do
    {
        var keyInfo = _inputReader.ReadKey(intercept: true);
        key = keyInfo.Key;

        if (key == ConsoleKey.Backspace)
        {
            if (lozinka.Length > 0)
            {
                Console.Write("\b \b");
                lozinka = lozinka[0..^1];
            }
        }
        else if (!char.IsControl(keyInfo.KeyChar))
        {
            Console.Write("*");
            lozinka += keyInfo.KeyChar;
        }
    } while (key != ConsoleKey.Enter);

    Console.WriteLine();
    return lozinka;
}
```

Slika 1. Metoda CitajLozinku

Ručno izračunata McCabe metrika:

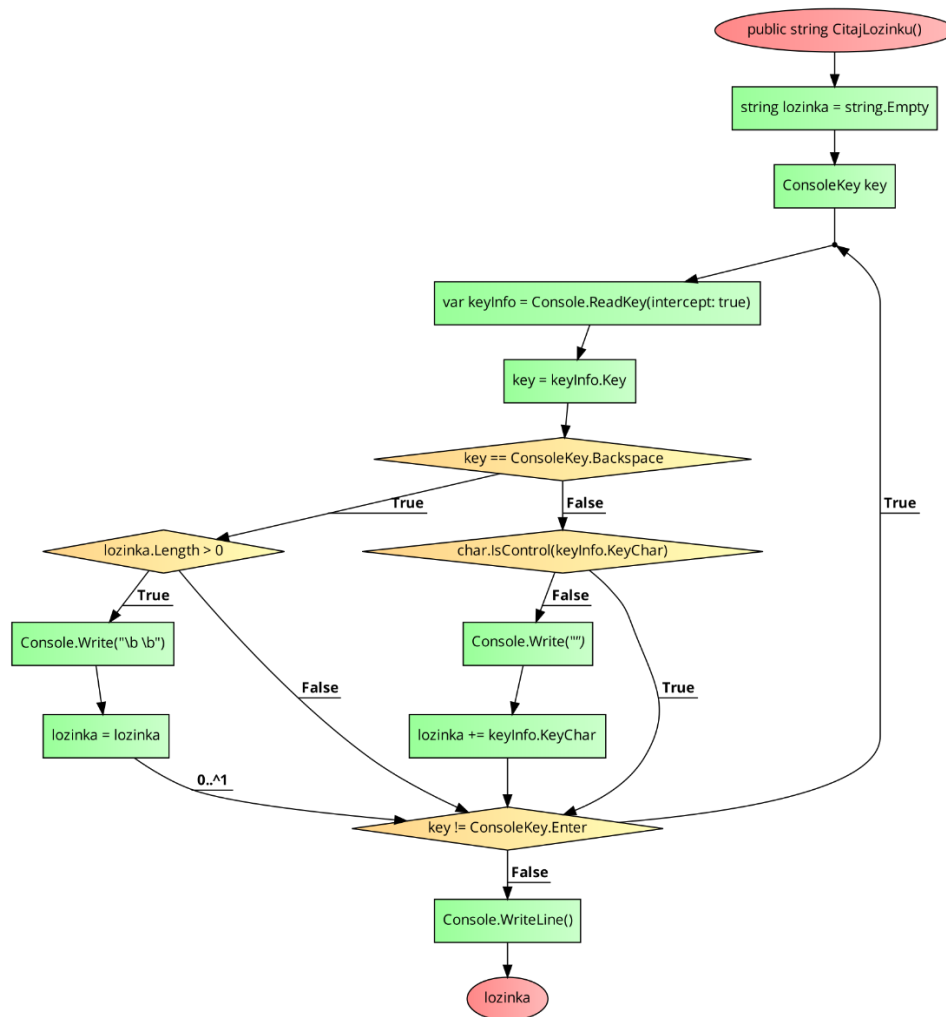
$$M = V(G) = E - N + 2P$$

Gdje je:

- E: Broj grana (edges) u kontrolnom grafu – za metodu CitajLozinku iznosi 19
- N: Broj čvorova (nodes) u kontrolnom grafu – za metodu CitajLozinku 16
- P: Broj povezanih komponenti (u ovom slučaju 1).

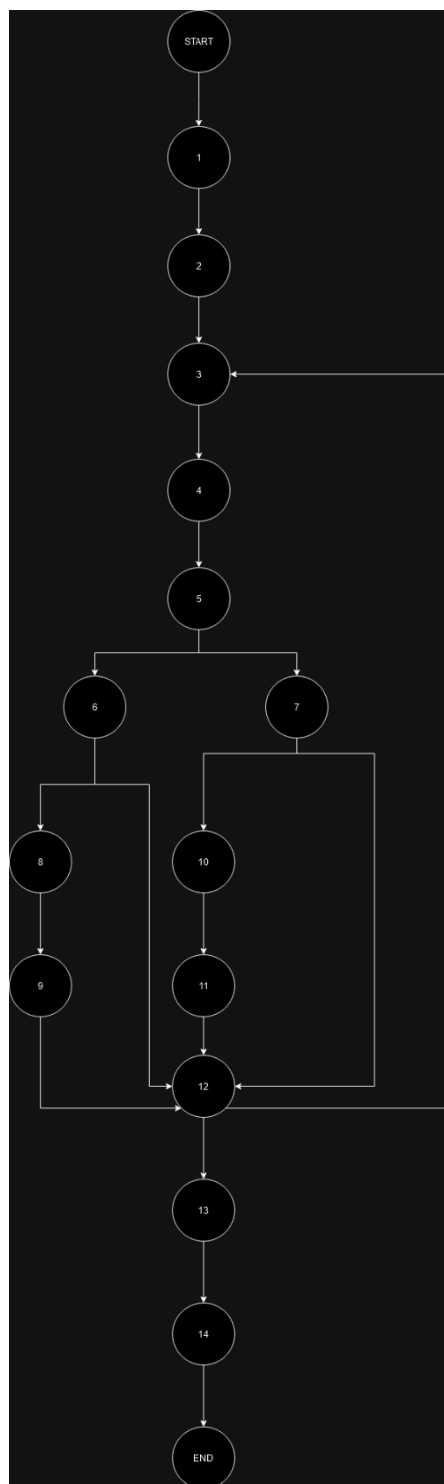
Vrijednost McCabe metrike:

$$M = V(G) = E - N + 2P = 19 - 16 + 2 = 5$$









Slika 2. Automatski generisani control flow diagram

Prikaz kontrolnog grafa



Slika 3. Kontrolni graf za metodu CitajLozinku

Rezultati iz Code Metrics alata

 korisnici : Dictionary<string, string>		93	0
 CitajLozinku() : string		59	5
 RegistrujKorisnika(string, string) : bool		77	2
 AutentifikujKorisnika(string, string) : bool		88	2
 HashLozinka(string) : string		77	1

Slika 4. Prikaz indeksa održavanja i ciklomatske složenosti za metodu CitajLozinku

Diskusija

Na osnovu prethodno izvršene analize, došli smo do saznanja da ciklomatska kompleksnost metode CitajLozinku iznosi 5, a indeks održavanja iznosi 59. McCabe metrika u vrijednosti 5 ukazuje da je ova metoda umjereno složena, a indeks održavanja u vrijednosti 59 ukazuje da je metoda srednje složena, međutim dodatni code tuning zajedno sa refaktoringom bi imao znatan utjecaj na složenosti i održivost.

2. White box testiranje

Odabrane metode za vršenje *white box* testiranja su:

- testiranje potpunog obuhvata iskaza/linija (statement/line coverage)
- testiranje potpunog obuhvata grana/odluka (branch/decision coverage)
- testiranje potpunog obuhvata uslova (conditional coverage)

Redni broj puta	Čvorovi koje put obuhvata
1.	START → 1 → 2 → 3 → 4 → 5 → 6 → 8 → 9 → 12 → 13 → 14 → END
2.	START → 1 → 2 → 3 → 4 → 5 → 6 → 12 → 13 → 14 → END
3.	START → 1 → 2 → 3 → 4 → 5 → 7 → 10 → 11 → 12 → 13 → 14 → END
4.	START → 1 → 2 → 3 → 4 → 5 → 7 → 12 → 13 → 14 → END
5.	START → 1 → 2 → 3 → 4 → 5 → (6/7) → ... → 12 → 3 → 4 → 5 → (6/7) -> ... → 12 → 13 → 14 → END

Tabela 1. Prikaz svih puteva kroz graf

Postizanje potpunog obuhvata iskaza/linija (statement/line coverage)

Da bismo postigli potpuni obuhvat iskaza/linija potrebno je kreirati testove koji će obuhvatiti sljedeće puteve kroz graf:

1. **START** → 1 → 2 → 3 → 4 → 5 → 6 → 8 → 9 → 12 → 13 → 14 → **END**
2. **START** → 1 → 2 → 3 → 4 → 5 → 7 → 10 → 11 → 12 → 13 → 14 → **END**

```
public void CitajLozinku_InputWithBackspace_HandlesBackspaceCorrectly()
{
    _mockInputReader.SetupSequence(reader => reader.ReadKey(true))
        .Returns(new ConsoleKeyInfo('p', ConsoleKey.P, false, false, false))
        .Returns(new ConsoleKeyInfo('a', ConsoleKey.A, false, false, false))
        .Returns(new ConsoleKeyInfo('s', ConsoleKey.S, false, false, false))
        .Returns(new ConsoleKeyInfo('s', ConsoleKey.S, false, false, false))
        .Returns(new ConsoleKeyInfo('b', ConsoleKey.Backspace, false, false, false))
        .Returns(new ConsoleKeyInfo('\b', ConsoleKey.Backspace, false, false, false))
        .Returns(new ConsoleKeyInfo('w', ConsoleKey.W, false, false, false))
        .Returns(new ConsoleKeyInfo('o', ConsoleKey.O, false, false, false))
        .Returns(new ConsoleKeyInfo('r', ConsoleKey.R, false, false, false))
        .Returns(new ConsoleKeyInfo('d', ConsoleKey.D, false, false, false))
        .Returns(new ConsoleKeyInfo('\r', ConsoleKey.Enter, false, false, false));

    string result = _authService.CitajLozinku();

    Assert.AreEqual("paword", result, "Backspace nije pravilno obrađen.");
}
```

Slika 5. Test za put broj 1

```
public void CitajLozinku_ValidInput_ReturnsCorrectPassword()
{
    _mockInputReader.SetupSequence(reader => reader.ReadKey(true))
        .Returns(new ConsoleKeyInfo('p', ConsoleKey.P, false, false, false))
        .Returns(new ConsoleKeyInfo('a', ConsoleKey.A, false, false, false))
        .Returns(new ConsoleKeyInfo('s', ConsoleKey.S, false, false, false))
        .Returns(new ConsoleKeyInfo('s', ConsoleKey.S, false, false, false))
        .Returns(new ConsoleKeyInfo('w', ConsoleKey.W, false, false, false))
        .Returns(new ConsoleKeyInfo('o', ConsoleKey.O, false, false, false))
        .Returns(new ConsoleKeyInfo('r', ConsoleKey.R, false, false, false))
        .Returns(new ConsoleKeyInfo('d', ConsoleKey.D, false, false, false))
        .Returns(new ConsoleKeyInfo('1', ConsoleKey.D1, false, false, false))
        .Returns(new ConsoleKeyInfo('2', ConsoleKey.D2, false, false, false))
        .Returns(new ConsoleKeyInfo('3', ConsoleKey.D3, false, false, false))
        .Returns(new ConsoleKeyInfo('\r', ConsoleKey.Enter, false, false, false));

    string result = _authService.CitajLozinku();

    Assert.AreEqual("password123", result, "Lozinka nije pravilno pročitana.");
}
```

Slika 6. Test za put broj 2

Pokrivenost koda:

Services.AuthenticationService	27	0	27	68	100%	<div></div>	8	8	100%	<div></div>
--------------------------------	----	---	----	----	------	-------------	---	---	------	-------------

Slika 7. Rezultat pokrivenosti iskaza/linija koda iz Code Metrics alata

Postizanje potpunog obuhvata grana/odluka (branch/decision coverage)

Da bismo postigli potpuni obuhvat grana/odluka potrebno je kreirati testove koji će obuhvatiti sljedeće puteve kroz graf:

1. **START** → 1 → 2 → 3 → 4 → 5 → 6 → 8 → 9 → 12 → 13 → 14 → **END**
2. **START** → 1 → 2 → 3 → 4 → 5 → 6 → 12 → 13 → 14 → **END**
3. **START** → 1 → 2 → 3 → 4 → 5 → 7 → 10 → 11 → 12 → 13 → 14 → **END**
4. **START** → 1 → 2 → 3 → 4 → 5 → 7 → 12 → 13 → 14 → **END**

```
public void CitajLozinku_BackspaceWithNonEmptyPassword_ShouldReturnCorrect()
{
    _mockInputReader.SetupSequence(x => x.ReadKey(true))
        .Returns(new ConsoleKeyInfo('a', ConsoleKey.A, false, false, false))
        .Returns(new ConsoleKeyInfo('b', ConsoleKey.B, false, false, false))
        .Returns(new ConsoleKeyInfo('\b', ConsoleKey.Backspace, false, false, false))
        .Returns(new ConsoleKeyInfo('\r', ConsoleKey.Enter, false, false, false));

    string result = _authService.CitajLozinku();

    Assert.AreEqual("a", result);
}
```

Slika 8. Test za put broj 1

```
public void CitajLozinku_BackspaceWithEmptyPassword_ShouldReturnCorrect()
{
    _mockInputReader.SetupSequence(x => x.ReadKey(true))
        .Returns(new ConsoleKeyInfo('\b', ConsoleKey.Backspace, false, false, false))
        .Returns(new ConsoleKeyInfo('\r', ConsoleKey.Enter, false, false, false));

    string result = _authService.CitajLozinku();

    Assert.AreEqual("", result);
}
```

Slika 9. Test za put broj 2

```

public void CitajLozinku_EnterOnlyPassword_ShouldReturnCorrect()
{
    _mockInputReader.SetupSequence(x => x.ReadKey(true))
        .Returns(new ConsoleKeyInfo('a', ConsoleKey.A, false, false, false))
        .Returns(new ConsoleKeyInfo('b', ConsoleKey.B, false, false, false))
        .Returns(new ConsoleKeyInfo('\r', ConsoleKey.Enter, false, false, false));

    string result = _authService.CitajLozinku();

    Assert.AreEqual("ab", result);
}

```

Slika 10. Test za put broj 3

```

public void CitajLozinku_ControlCharactersIgnored_ShouldReturnCorrect()
{
    _mockInputReader.SetupSequence(x => x.ReadKey(true))
        .Returns(new ConsoleKeyInfo('a', ConsoleKey.A, false, false, false))
        .Returns(new ConsoleKeyInfo('\t', ConsoleKey.Tab, false, false, false))
        .Returns(new ConsoleKeyInfo('b', ConsoleKey.B, false, false, false))
        .Returns(new ConsoleKeyInfo('\r', ConsoleKey.Enter, false, false, false));

    string result = _authService.CitajLozinku();

    Assert.AreEqual("ab", result);
}

```

Slika 11. Test za put broj 4

Pokrivenost koda:

Services.AuthenticationService	27	0	27	68	100%	<div></div>	8	8	100%	<div></div>
--------------------------------	----	---	----	----	------	-------------	---	---	------	-------------

Slika 12. Rezultat pokrivenosti grana/odluka iz Code Metrics alata

Postizanje potpunog obuhvata uslova (conditional coverage)

Da bismo postigli potpuni obuhvat uslova potrebno je kreirati testove koji će obuhvatiti sljedeće puteve kroz graf:

1. **START** → 1 → 2 → 3 → 4 → 5 → 6 → 8 → 9 → 12 → 13 → 14 → **END**
2. **START** → 1 → 2 → 3 → 4 → 5 → 6 → 12 → 13 → 14 → **END**
3. **START** → 1 → 2 → 3 → 4 → 5 → 7 → 10 → 11 → 12 → 13 → 14 → **END**
4. **START** → 1 → 2 → 3 → 4 → 5 → 7 → 12 → 13 → 14 → **END**


```

public void CitajLozinku_BackspaceWithNonEmptyPassword_ShouldReturnCorrectPassword()
{
    _mockInputReader.SetupSequence(x => x.ReadKey(true))
        .Returns(new ConsoleKeyInfo('a', ConsoleKey.A, false, false, false))
        .Returns(new ConsoleKeyInfo('\b', ConsoleKey.Backspace, false, false, false))
        .Returns(new ConsoleKeyInfo('\r', ConsoleKey.Enter, false, false, false));

    string result = _authService.CitajLozinku();

    Assert.AreEqual("", result);
}

```

Slika 13. Test za put broj 1

```

public void CitajLozinku_BackspaceWithEmptyPassword_ShouldReturnCorrectPassword()
{
    _mockInputReader.SetupSequence(x => x.ReadKey(true))
        .Returns(new ConsoleKeyInfo('\b', ConsoleKey.Backspace, false, false, false))
        .Returns(new ConsoleKeyInfo('\r', ConsoleKey.Enter, false, false, false));

    string result = _authService.CitajLozinku();

    Assert.AreEqual("", result);
}

```

Slika 14. Test za put broj 2

```

public void CitajLozinku_NonBackspaceNonControlChar_ShouldReturnCorrectPassword()
{
    _mockInputReader.SetupSequence(x => x.ReadKey(true))
        .Returns(new ConsoleKeyInfo('a', ConsoleKey.A, false, false, false))
        .Returns(new ConsoleKeyInfo('\r', ConsoleKey.Enter, false, false, false));

    string result = _authService.CitajLozinku();

    Assert.AreEqual("a", result);
}

```

Slika 15. Test za put broj 3

```

public void CitajLozinku_NonBackspaceControlChar_ShouldReturnCorrectPassword()
{
    _mockInputReader.SetupSequence(x => x.ReadKey(true))
        .Returns(new ConsoleKeyInfo('\t', ConsoleKey.Tab, false, false, false))
        .Returns(new ConsoleKeyInfo('\r', ConsoleKey.Enter, false, false, false));

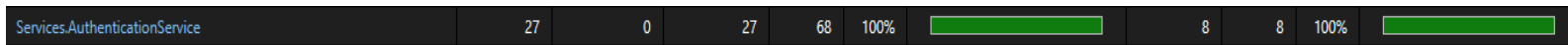
    string result = _authService.CitajLozinku();

    Assert.AreEqual("", result);
}

```

Slika 16. Test za put broj 4

Pokrivenost koda:



Slika 17. Rezultat pokrivenosti uslova iz Code Metrics alata

3. Code tuning

U nastavku ćemo vršiti code tuning nad metodom CitajLozinku s ciljem poboljšanja performansi, specifičnije vremena izvršavanja, memorijske kompleksnosti i povećanja čitljivosti.

Ovu analizu započinjemo kreiranjem testnog slučaja koji će vršiti veliki broj pokretanja metode CitajLozinku, s ciljem boljeg razumijevanja dešavanja u slučaju da metoda nije efikasno napisana.

```
public string CitajLozinku()
{
    string lozinka = string.Empty;
    ConsoleKey key;
    do
    {
        var keyInfo = _inputReader.ReadKey(intercept: true);
        key = keyInfo.Key;

        if (key == ConsoleKey.Backspace)
        {
            if (lozinka.Length > 0)
            {
                Console.Write("\b \b");
                lozinka = lozinka[0..^1];
            }
        }
        else if (!char.IsControl(keyInfo.KeyChar))
        {
            Console.Write("*");
            lozinka += keyInfo.KeyChar;
        }
    } while (key != ConsoleKey.Enter);

    Console.WriteLine();
    return lozinka;
}
```

Slika 18. Metoda CitajLozinku

```

[TestMethod]
public void CitajLozinku_LongPasswordWithManyOperations()
{
    var keySequence = new Queue<ConsoleKeyInfo>();
    var expectedResult = new StringBuilder();
    int _iterationCount = 300000;

    for (int i = 0; i < _iterationCount; i++)
    {
        // Dodavanje regularnih karaktera
        keySequence.Enqueue(new ConsoleKeyInfo('a', ConsoleKey.A, false, false, false));
        expectedResult.Append('a');

        // Backspace poslije svakih 100 karaktera
        if (i % 100 == 0)
        {
            for (int j = 0; j < 20; j++)
            {
                keySequence.Enqueue(new ConsoleKeyInfo('\b', ConsoleKey.Backspace, false, false, false));
                if (expectedResult.Length > 0)
                    expectedResult.Length--;
            }
        }

        // Control karakteri poslije svakih 50 karaktera
        if (i % 50 == 0)
        {
            keySequence.Enqueue(new ConsoleKeyInfo('\t', ConsoleKey.Tab, false, false, false));
            keySequence.Enqueue(new ConsoleKeyInfo('\u0013', ConsoleKey.Pause, false, false, false));
        }

        // Specijalni karakteri poslije svakih 200 karaktera
        if (i % 200 == 0)
        {
            var specialChars = "!@#$%^&*()_+";
            foreach (char c in specialChars)
            {
                keySequence.Enqueue(new ConsoleKeyInfo(c, ConsoleKey.A, false, false, false));
                expectedResult.Append(c);
            }
        }
    }

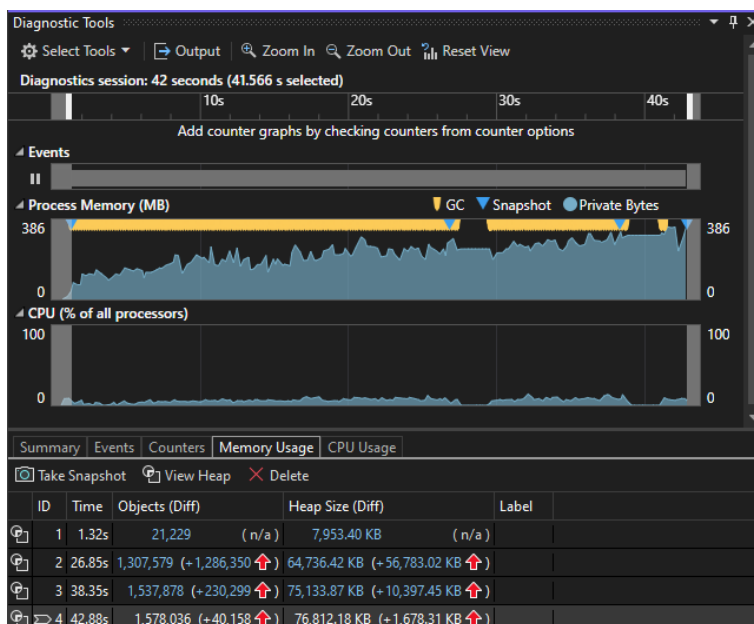
    // Enter za kraj
    keySequence.Enqueue(new ConsoleKeyInfo('\r', ConsoleKey.Enter, false, false, false));

    _mockInputReader
        .Setup(x => x.ReadKey(It.IsAny<bool>()))
        .Returns(() => keySequence.Count > 0 ? keySequence.Dequeue() : new ConsoleKeyInfo('\r', ConsoleKey.Enter, false, false, false));

    string result = _authService.CitajLozinku();
    Assert.AreEqual(expectedResult.ToString(), result);
}

```

Slika 19. Testni slučaj koji vrši mnogobrojno pozivanje metode CitajLozinku



Slika 20. Prikaz iz alata Diagnostic Tools za prethodni testni slučaj

Diskusija

Ukupna dijagnostička sesija je trajala približno 43 sekunde. Na slici se također mogu vidjeti 4 snapshota koja pokazuju stanje prilikom izvršavanja koda. Na osnovu snapshota u različitim stadijima izvršavanja koda možemo vidjeti da se broj objekata, ali i iskorištene memorije povećao sa 21,229 na 1,578,036 i 7,954.40 KB na 76,812.17 KB respektivno, što ukazuje na znatnu akumulaciju podataka. CPU opterećenje je relativno nisko, sa povremenim skokovima aktivnosti, što ne predstavlja pretjerano zahtjevan zadatak za procesor.

PRIMJENA CODE TUNING-a

U nastavku ćemo primijeniti 3 tehnike code tuning-a, s ciljem poboljšanja performansi izvornog koda:

- tehnika 1: optimizacija pomoću sentinel vrijednosti
- tehnika 2: uređivanje iskaza po frekvenciji
- tehnika 3: zamjena string tipa podatka sa StringBuilder-om

Tehnika 1 - optimizacija pomoću sentinel vrijednosti

```
public string CitajLozinku()
{
    string lozinka = string.Empty;
    ConsoleKey key;

    const ConsoleKey SentinelKey = ConsoleKey.Enter;

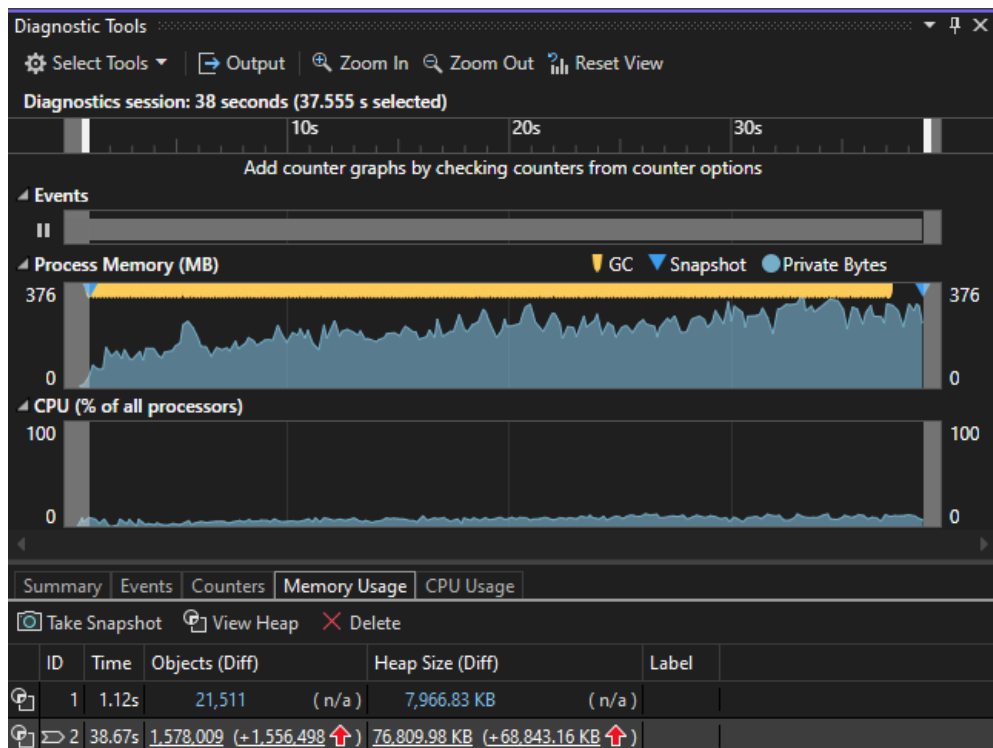
    do
    {
        var keyInfo = _inputReader.ReadKey(intercept: true);
        key = keyInfo.Key;

        if (key == SentinelKey)
        {
            break;
        }
        if (key == ConsoleKey.Backspace && lozinka.Length > 0)
        {
            Console.Write("\b \b");
            lozinka = lozinka[0..^1];
        }
        else if (!char.IsControl(keyInfo.KeyChar))
        {
            Console.Write("*");
            lozinka += keyInfo.KeyChar;
        }
    } while (key != ConsoleKey.Enter);

    Console.WriteLine();
    return lozinka;
}
```

≤ 37,551ms elapsed

Slika 21. Code Tuning korištenjem sentinel vrijednosti



Slika 22. Prikaz iz alata Diagnostic Tools nakon prve tehnike Code Tuning-a

Diskusija

Ukupna dijagnostička sesija je trajala približno 39 sekundi. Na slici se također mogu vidjeti 2 snapshota koja pokazuju stanje prilikom izvršavanja koda. Na osnovu snapshota u različitim stadijima izvršavanja koda možemo vidjeti da se broj objekata, ali i iskorištene memorije povećao sa 21,511 na 1,578,009 i 7,966.83 KB na 76,809.98 KB respektivno, što ukazuje na znatnu akumulaciju podataka. U poređenju sa rezultatima prije izvršenog Code Tuning-a, možemo vidjeti da je došlo do minimalnog smanjenja iskorištenih objekata, kao i memorije, međutim, vrijeme izvršavanja je smanjeno za približno 4 sekunde što ukazuje na poboljšanje efikasnosti metode. Također, nakon dodavanja Sentinel vrijednosti, dolazi do povećanja McCabe metrike sa 5 na 6. Pored toga, indeks održavanja mijenja vrijednost, sa 59 na 56.

Tehnika 2 - Uređivanje iskaza po frekvenciji

```
public string CitajLozinku()
{
    string lozinka = string.Empty;
    ConsoleKey key;

    const ConsoleKey SentinelKey = ConsoleKey.Enter;

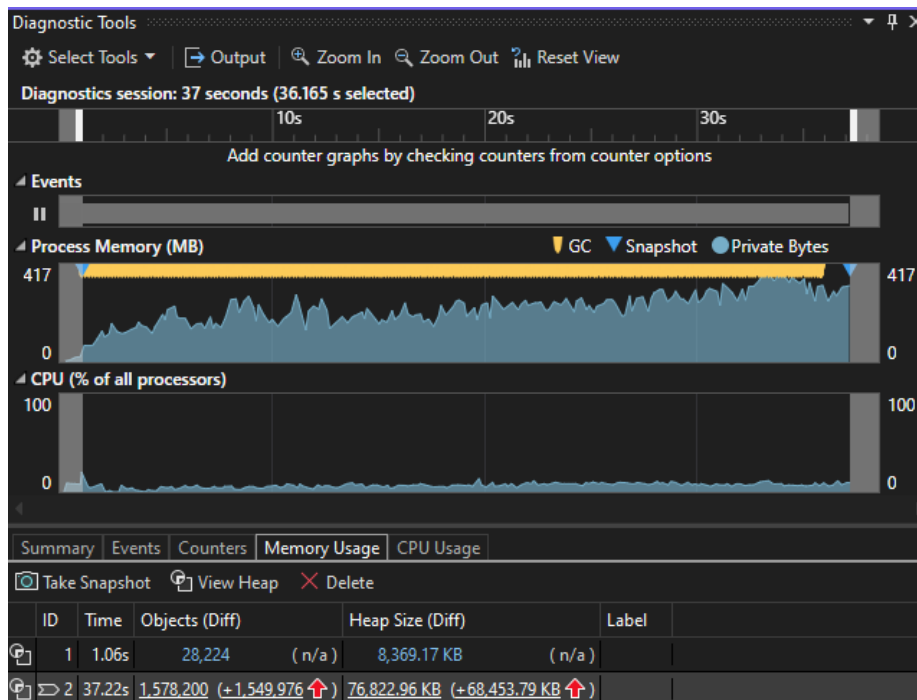
    do
    {
        var keyInfo = _inputReader.ReadKey(intercept: true);
        key = keyInfo.Key;

        if (key == SentinelKey)
        {
            break;
        }
        else if (!char.IsControl(keyInfo.KeyChar))
        {
            Console.Write("*");
            lozinka += keyInfo.KeyChar;
        }
        else if (key == ConsoleKey.Backspace && lozinka.Length > 0)
        {
            Console.Write("\b \b");
            lozinka = lozinka[0..^1];
        }
    } while (key != ConsoleKey.Enter);

    Console.WriteLine();
    return lozinka;
}
```

≤ 36,165ms elapsed

Slika 24. Code Tuning korištenjem uređivanja iskaza po frekvenciji



Slika 25. Prikaz iz alata Diagnostic Tools nakon druge tehnike Code Tuning-a

Diskusija

Ukupna dijagnostička sesija je trajala približno 37 sekundi. Na slici se također mogu vidjeti 2 snapshota koja pokazuju stanje prilikom izvršavanja koda. Na osnovu snapshota u različitim stadijima izvršavanja koda možemo vidjeti da se broj objekata, ali i iskorištene memorije povećao sa 28,224 na 1,578,200 i 8,469.17 KB na 76,822.96 KB respektivno, što ukazuje na znatnu akumulaciju podataka. U poređenju sa rezultatima nakon uvođenja Sentinel vrijednosti, možemo vidjeti da je došlo do minimalnog povećanja iskorištenih objekata, kao i memorije, međutim, vrijeme izvršavanja je smanjeno za približno 2 sekunde, što ukazuje na poboljšanje efikasnosti metode. Nakon uređivanja/promjene rasporeda iskaza, preciznije if iskaza, ne dolazi do promjene McCabe metrike i indeksa održavanja.

Tehnika 3 - Zamjena string tipa podatka sa StringBuilder-om

```
public string CitajLozinku()
{
    var lozinka = new StringBuilder();
    ConsoleKey key;

    const ConsoleKey SentinelKey = ConsoleKey.Enter;

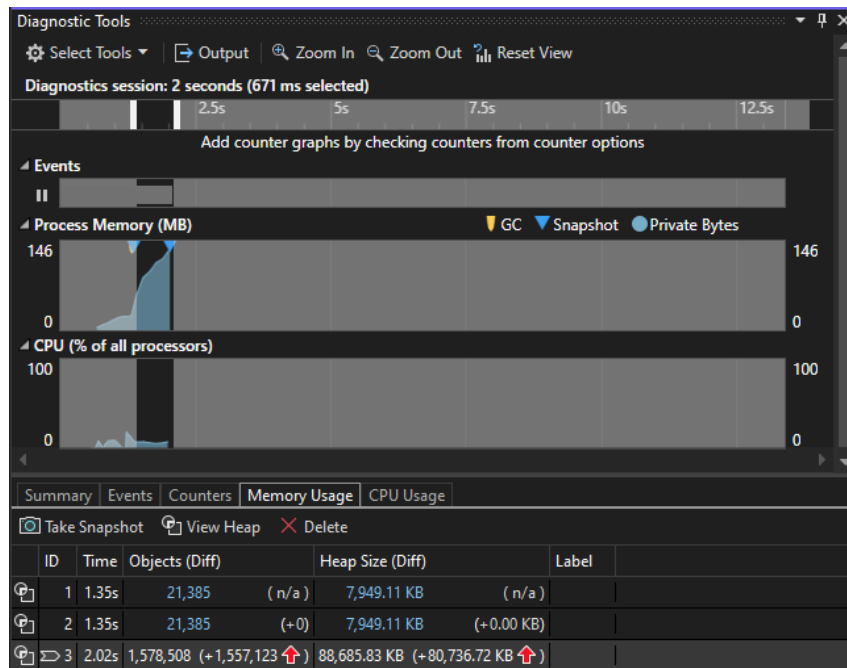
    do
    {
        var keyInfo = _inputReader.ReadKey(intercept: true);
        key = keyInfo.Key;

        if (key == SentinelKey)
        {
            break;
        }

        if (!char.IsControl(keyInfo.KeyChar))
        {
            Console.Write("*");
            lozinka.Append(keyInfo.KeyChar);
        }
        else if (key == ConsoleKey.Backspace && lozinka.Length > 0)
        {
            Console.Write("\b \b");
            lozinka.Length--;
        }
    } while (key != ConsoleKey.Enter);

    Console.WriteLine();
    return lozinka.ToString();
}
≤ 671ms elapsed
```











Slika 26. Code Tuning zamjenom tipa podatka







Slika 27. Prikaz iz alata Diagnostic Tools nakon treće tehnike Code Tuning-a

Diskusija

Ukupna dijagnostička sesija je trajala približno 2 sekunde. Na slici se također mogu vidjeti 2 snapshota koja nam pokazuju stanje prilikom izvršavanja koda. Na osnovu snapshota u različitim stadijima izvršavanja koda možemo vidjeti da se broj objekata, ali i iskorištene memorije povećao sa 21,385 na 1,578,508 i 7,949.11 KB na 88,685.83 KB respektivno, što ukazuje na znatnu akumulaciju podataka. U poređenju sa rezultatima nakon promjene rasporeda iskaza u metodi, možemo vidjeti da je došlo do minimalnog povećanja iskorištenih objekata, ali se iskorištenost memorije znatno povećala. Vrijeme izvršavanja se smanjilo eksponencijalno, sa 37 sekundi na 2 sekunde, što ukazuje da je najveći problem u izvornoj metodi bilo korištenje immutable tipa podatka, string-a. Nakon promjene tipa podatka lozinke, sa tipa string na tip StringBuilder, ne dolazi do promjene McCabe metrike i indeksa održavanja.

 korisnici : Dictionary<string, string>		93	0
 CitajLozinku() : string		59	5
 RegistrujKorisnika(string, string) : bool		77	2
 AutentifikujKorisnika(string, string) : bool		88	2
 HashLozinka(string) : string		77	1

Slika 28. Prikaz iz alata Code Metrics prije Code Tuning-a

 korisnici : Dictionary<string, string>		93	0
 _inputReader : InputReader		100	0
 AuthenticationService(InputReader)		96	1
 CitajLozinku() : string		56	6

Slika 29. Prikaz iz alata Code Metrics nakon Code Tuning-a

4. Refaktoring na osnovu checkliste

Idući dio analize se fokusira na refaktoring koda, s ciljem poboljšanja efikasnosti, lakoće održavanja, kao i smanjenja ciklomatske kompleksnosti.

Odabrane tehnike za ovu analizu uključuju:

- “replace a magic number with a named constant”
- “move a routine’s code inline” + “move a complex boolean expression into a well-named boolean function”
- “use break or return instead of a loop control variable”
- “extract a routine”
- “rename a variable with a clearer or more informative name”

Tehnika 1 - replace a magic number with a named constant

```
private const char MASK_CHAR = '*';
private const string BACKSPACE_SEQUENCE = "\b \b";_
4 references | 1/1 passing
public string CitajLozinku()
{
    var lozinka = new StringBuilder();
    ConsoleKey key;

    const ConsoleKey SentinelKey = ConsoleKey.Enter;

    do
    {
        var keyInfo = _inputReader.ReadKey(intercept: true);
        key = keyInfo.Key;

        if (key == SentinelKey)
        {
            break;
        }
        else if (!char.IsControl(keyInfo.KeyChar))
        {
            Console.Write(MASK_CHAR);
            lozinka.Append(keyInfo.KeyChar);
        }
        else if (key == ConsoleKey.Backspace && lozinka.Length > 0)
        {
            Console.Write(BACKSPACE_SEQUENCE);
            lozinka.Length--;
        }
    } while (key != ConsoleKey.Enter);

    Console.WriteLine();
    return lozinka.ToString();
}
```

Slika 30. Prikaz metode CitajLozinku nakon optimizacije tehnikom 1

AuthenticationService	71	7
korisnici : Dictionary<string, string>	93	0
_inputReader : InputReader	100	0
AuthenticationService(InputReader)	96	1
MASK_CHAR : char	93	0
BACKSPACE_SEQUENCE : string	93	0
CitajLozinku() : string	56	6

Slika 31. Prikaz iz alata Code Metrics nakon optimizacije tehnikom 1

Tehnika 2 - move a routine's code inline + move a complex boolean expression into a well-named boolean function

```
private const char MASK_CHAR = '*';
private const string BACKSPACE_SEQUENCE = "\b \b";
4 references | 1/1 passing
public string CitajLozinku()
{
    var lozinka = new StringBuilder();
    ConsoleKey key;

    do
    {
        var keyInfo = _inputReader.ReadKey(intercept: true);
        key = keyInfo.Key;

        if (IsEnterKey(key))
        {
            return lozinka.ToString();
        }
        else if (!char.IsControl(keyInfo.KeyChar))
        {
            Console.Write(MASK_CHAR);
            lozinka.Append(keyInfo.KeyChar);
        }
        else if (IsBackspaceWithText(key, lozinka))
        {
            Console.Write(BACKSPACE_SEQUENCE);
            lozinka.Length--;
        }
    } while (key != ConsoleKey.Enter);

    Console.WriteLine();
    return lozinka.ToString();
}

1 reference
private bool IsEnterKey(ConsoleKey key) => key == ConsoleKey.Enter;

1 reference
private bool IsBackspaceWithText(ConsoleKey key, StringBuilder lozinka) => key == ConsoleKey.Backspace && lozinka.Length > 0;
```

Slika 32. Prikaz metode CitajLozinku nakon optimizacije tehnikom 2

AuthenticationService	77	9
korisnici : Dictionary<string, string>	93	0
_inputReader : IInputReader	100	0
AuthenticationService(IInputReader)	96	1
MASK_CHAR : char	93	0
BACKSPACE_SEQUENCE : string	93	0
CitajLozinku() : string	58	5
IsEnterKey(ConsoleKey) : bool	95	1
IsBackspaceWithText(ConsoleKey, StringBuilder) : bool	89	2

Slika 33. Prikaz iz alata Code Metrics nakon optimizacije tehnikom 2

Tehnika 3 - use break or return instead of a loop control variable

```
private const char MASK_CHAR = '*';
private const string BACKSPACE_SEQUENCE = "\b \b";
4 references | 1/1 passing
public string CitajLozinku()
{
    var lozinka = new StringBuilder();
    ConsoleKey key;

    while(true)
    {
        var keyInfo = _inputReader.ReadKey(intercept: true);
        key = keyInfo.Key;

        if (IsEnterKey(key))
        {
            break;
        }
        else if (!char.IsControl(keyInfo.KeyChar))
        {
            Console.Write(MASK_CHAR);
            lozinka.Append(keyInfo.KeyChar);
        }
        else if (IsBackspaceWithText(key, lozinka))
        {
            Console.Write(BACKSPACE_SEQUENCE);
            lozinka.Length--;
        }
    }

    Console.WriteLine();
    return lozinka.ToString();
}

1 reference
private bool IsEnterKey(ConsoleKey key) => key == ConsoleKey.Enter;

1 reference
private bool IsBackspaceWithText(ConsoleKey key, StringBuilder lozinka) => key == ConsoleKey.Backspace && lozinka.Length > 0;
```

Slika 34. Prikaz metode *CitajLozinku* nakon optimizacije tehnikom 3

AuthenticationService	77	9
korisnici : Dictionary<string, string>	93	0
_inputReader : IInputReader	100	0
AuthenticationService(IInputReader)	96	1
MASK_CHAR : char	93	0
BACKSPACE_SEQUENCE : string	93	0
CitajLozinku() : string	58	5
IsEnterKey(ConsoleKey) : bool	95	1
IsBackspaceWithText(ConsoleKey, StringBuilder) : bool	89	2

Slika 35. Prikaz iz alata Code Metrics nakon optimizacije tehnikom 3

Tehnika 4 - extract a routine

```
private const char MASK_CHAR = '*';
private const string BACKSPACE_SEQUENCE = "\b \b";
4 references | 1/1 passing
public string CitajLozinku()
{
    var lozinka = new StringBuilder();

    while(true)
    {
        var keyInfo = _inputReader.ReadKey(intercept: true);

        if (IsEnterKey(keyInfo.Key))
            break;

        ProcessKeyPress(keyInfo, lozinka);
    }

    Console.WriteLine();
    return lozinka.ToString();
}

1 reference
private void ProcessKeyPress(ConsoleKeyInfo keyInfo, StringBuilder lozinka)
{
    if (!char.IsControl(keyInfo.KeyChar))
    {
        HandleRegularCharacter(keyInfo.KeyChar, lozinka);
    }
    else if (IsBackspaceWithText(keyInfo.Key, lozinka))
    {
        HandleBackspace(lozinka);
    }
}

1 reference
private void HandleRegularCharacter(char keyChar, StringBuilder lozinka)
{
    Console.Write(MASK_CHAR);
    lozinka.Append(keyChar);
}

1 reference
private void HandleBackspace(StringBuilder lozinka)
{
    Console.Write(BACKSPACE_SEQUENCE);
    lozinka.Length--;
}

1 reference
private bool IsEnterKey(ConsoleKey key) => key == ConsoleKey.Enter;
```

Slika 36. Prikaz metode CitajLozinku nakon optimizacije tehnikom 4

AuthenticationService	■	82	12
korisnici : Dictionary<string, string>	■	93	0
_inputReader : IInputReader	■	100	0
AuthenticationService(IInputReader)	■	96	1
MASK_CHAR : char	■	93	0
BACKSPACE_SEQUENCE : string	■	93	0
CitajLozinku() : string	■	68	3
ProcessKeyPress(ConsoleKeyInfo, StringBuilder) : void	■	75	3
HandleRegularCharacter(char, StringBuilder) : void	■	86	1
HandleBackspace(StringBuilder) : void	■	86	1
IsEnterKey(ConsoleKey) : bool	■	95	1
IsBackspaceWithText(ConsoleKey, StringBuilder) : bool	■	89	2

Slika 37. Prikaz iz alata Code Metrics nakon optimizacije tehnikom 4

Tehnika 5 - rename a variable with a clearer or more informative name

```
public string HidePasswordInput()
```

Slika 38. Promjena imena metode

AuthenticationService	■	82	12
korisnici : Dictionary<string, string>	■	93	0
_inputReader : IInputReader	■	100	0
AuthenticationService(IInputReader)	■	96	1
MASK_CHAR : char	■	93	0
BACKSPACE_SEQUENCE : string	■	93	0
CitajLozinku() : string	■	68	3
ProcessKeyPress(ConsoleKeyInfo, StringBuilder) : void	■	75	3
HandleRegularCharacter(char, StringBuilder) : void	■	86	1
HandleBackspace(StringBuilder) : void	■	86	1
IsEnterKey(ConsoleKey) : bool	■	95	1
IsBackspaceWithText(ConsoleKey, StringBuilder) : bool	■	89	2

Slika 39. Prikaz iz alata Code Metrics nakon optimizacije tehnikom 5

Diskusija

Nakon izvršenih optimizacija koda može se vidjeti znatno poboljšanje indeksa održavanja i McCabe metrike – ciklomatske kompleksnosti, sa 59 na 68 i sa 5 na 3, respektivno. Ovo predstavlja značajno poboljšanje u poređenju sa izvornom metodom. Najveći utjecaj na ovu promjenu je imala tehnika *extract a routine*, koja doprinosi poboljšanoj modularnosti i smanjuje kompleksnost cijele metode. Također, izvorna metoda je sada lakša za održavanje i testiranje, što potvrđuje znatno poboljšani indeks održavanja.