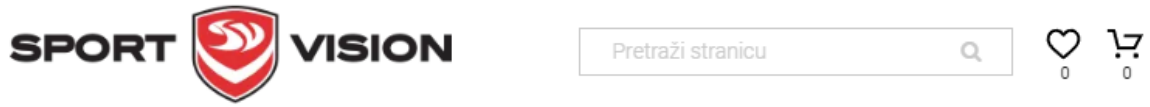
SVEUČILIŠTE U RIJECI
FAKULTET INFORMATIKE I DIGITALNIH TEHNOLOGIJA

SEMINAR U SKLOPU ZAVRŠNOG ISPITA
WEB SCRAPING APLIKACIJA

Kolegij: Programske Paradigme i Jezici
Autor: Leo Vukoje
Nositeljica kolegija: prof. dr. sc. Ana Meštrović
Asistent: Karlo Babić

U sklopu završnog ispita za praktični dio sam odabrao temu aplikacije Web scraping. Na temelju te teme izradio sam aplikaciju koja od korisnika na početku zahtijeva unos pojma pretrage koji će se pretražiti na web trgovini Sportvision. Korisnikov unos u konzolu jednak je unosu u polje „Pretraži stranicu“.



Nakon što korisnik unese željeni pojam, skripta izvršava upit prema web trgovini onoliko puta koliko postoji stranica sa rezultatima tog pojma. Kada izvrši sva dohvaćanja i rezultate pretraživanja premapira u objekte klase *Product* unutar aplikacije, korisniku se daje upit o tome kako želi pretražiti svoje proizvode. Nude mu se 4 opcije: uzlazno sortiranje po cijenama, silazno sortiranje po cijenama, uzlazno sortiranje po popustu te silazno sortiranje po popustu.

```
Pretraga proizvoda na sportvision.hr web trgovini: patike
Found total of 2 pages

Going through page 1
Going through page 2

1) Sortiraj po cijeni - od najnize
2) Sortiraj po cijeni - od najvise
3) Sortiraj po popustu - od najnizeg
4) Sortiraj po popustu - od najviseg
0) Kraj koristenja


Odabir:
```

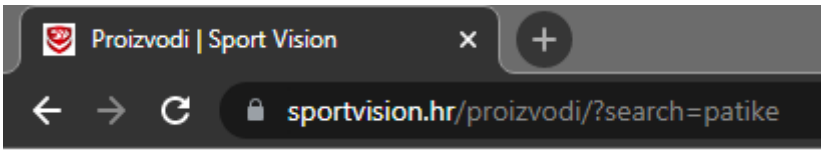
Dodatna značajka je ta da se korisnika pita koliko proizvoda želi da mu se ispiše, tako ako odabere opciju „1) Sortiraj po cijeni – od najnize“ te na dodatni upit unese da želi 5 proizvoda, ispisati će mu se prvih 5 najjeftinijih rezultata pretraživanja.

```
Odabir: 1
Koliko proizvoda za prikazati: 5
Ime: New Balance TENISICE NB
Cijena: 26,48e
Popust: 50%
URL: https://www.sportvision.hr/tenisice/465766-new-balance-tenisice-nb

Ime: adidas TENISICE GRAND COURT
Cijena: 28,56e
Popust: 20%
URL: https://www.sportvision.hr/tenisice/678807-adidas-tenisice-grand-court

Ime: New Balance TENISICE W 500
Cijena: 33,11e
Popust: 50%
URL: https://www.sportvision.hr/tenisice/516896-new-balance-tenisice-w-500

Ime: adidas TENISICE EQ21 RUN EL K
Cijena: 35,21e
Popust: 30%
URL: https://www.sportvision.hr/tenisice/761016-adidas-tenisice-eq21-run-el-k

Ime: New Balance TENISICE K997
Cijena: 35,76e
Popust: 40%
URL: https://www.sportvision.hr/tenisice/710247-new-balance-tenisice-k997

1) Sortiraj po cijeni - od najnize
2) Sortiraj po cijeni - od najvise
3) Sortiraj po popustu - od najnizeg
4) Sortiraj po popustu - od najviseg
0) Kraj koristenja


Odabir:
```

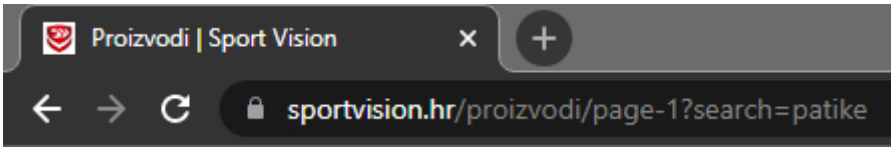
Aplikacija se izvodi sve dok korisnik ne unese „0“ kao znak za kraj korištenja aplikacije, na svaki unos koji nije naveden u izborniku, korisnik dobiva poruku o pogrešnom unosu.

Tok logike i razlika među sintaksama jezika

U aplikaciji 1. korak je traženje unosa od korisnika za pojam pretrage na web trgovini od čega se zatim pravi početni URL koji će se pretražiti. Na web trgovini kada se unese neki pojam pretrage URL dobiva nastavak „proizvodi/?search=“ i onda korisnikov pojam u kojemu su riječi spojene znakom „+“.



S obzirom na to da svi rezultati ne mogu stati na jednu stranicu, oni su paginirani pa je potrebno da nekako dohvatimo rezultate sa svake stranice. Tada odlaskom na neku drugu stranicu možemo uočiti da URL nakon nastavka „proizvodi“ dobije i još jedan nastavak vezan za taj koja stranica se pretražuje.



Te informacije su dovoljne da bi se kreirala funkcija koja će složiti URL za dohvaćanje rezultata nekog pojma pretrage sa neke stranice.

```
let compoundUrl(url:string, pageNumber: int, input: string) =
    let words = input.Split ' '

    let mutable urlOutput = url + "proizvodi/page-" + string(pageNumber) + "/?search="
    for word in words do
        urlOutput <- urlOutput + word + "+"

    urlOutput.Remove(urlOutput.Length - 1)
```

Slika 1. CompoundUrl metoda u F#

```
2 references | Leo Vukoje, 9 hours ago | 1 author, 1 change
private static string CompoundUrl(string url, int pageNumber, string input)
{
    var words = input.Split(" ");

    var urlOutput = url + "proizvodi/page-" + pageNumber.ToString() + "/?search=";
    foreach(var word in words)
    {
        urlOutput += word + "+";
    }

    return urlOutput.Remove(urlOutput.Length - 1);
}
```

Slika 2. CompoundUrl metoda u C#

Metode rade i za dohvaćanje prve stranice jer ako se proslijedi „page-0“ u URL također dobijemo rezultate sa prve stranice kao i bez tog nastavka.

Nakon slaganja URL-a trebamo dohvatiti HTML sa tog linka. Za to koristimo „HttpClient“ klasu iz biblioteke „System.Net.Http“.

```
let getHtml(url: string) = async {
    use client: HttpClient = new HttpClient()
    let! (response: string) = client.GetStringAsync(url) |> Async.AwaitTask
    return response
}
```

Slika 3. GetHtml metoda u F#

```
2 references | Leo Vukoje, 9 hours ago | 1 author, 1 change
private static async Task<string> GetHtml(string url)
{
    using var client = new HttpClient();
    return await client.GetStringAsync(url);
}
```

Slika 4. GetHtml metoda u C#

I u C# i u F# varijablu „client“ instanciramo sa istim principom, a to je da se varijabla izbriše nakon što nam više ne treba. U F# za to postoji ključna riječ „use“, a u C# „using“. Metoda „GetStringAsync“ dohvaća HTML sa proslijeđenog URL-a te si ga sprema u obliku stringa. Ostale metode koje su se mogle iskoristiti u tu svrhu su za spremanje u „IO stream“, „byte array“ te u obliku „HttpResponseMessage“. S obzirom na to da je metoda asinkrona, a bez nje ne možemo dalje izvoditi kôd, koristimo ključnu riječ „await“ da sačekamo rezultat pa s tim i cijela funkcija postaje asinkrona.

Sljedeći korak uključuje korištenje vanjske biblioteke koja mi je bila potrebna za rad sa učitanim HTML-om, a to je HtmlAgiltyPack.

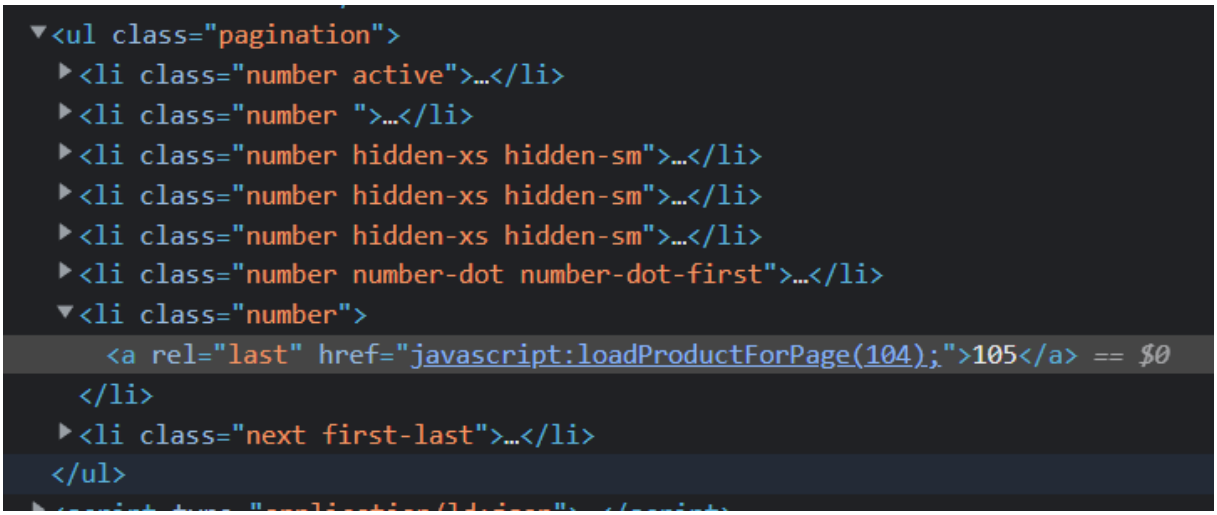
```
let doc = HtmlDocument()
doc.LoadHtml(html)
```

Slika 6. Učitavanje HTML-a - F#

```
var doc = new HtmlDocument();
doc.LoadHtml(html);
```

Slika 5. Učitavanje HTML-a C#

Nakon što smo učitali HTML u varijablu „doc“ možemo krenuti sa prolaskom kroz HTML i traženja što nam treba. Prva stvar koju moramo odrediti je broj stranica na koje su raspodijeljeni rezultati pretraživanja.



Slika 7. HTML oznaka paginacije za pojam "tenisice"

Iz dijela HTML-a sa slike koji prikazuje traku paginacije, vidimo da imamo oznaku „li“ sa CSS klasom „number“ te da nam ona govori o broju posljednje stranice sa rezultatima. Sličnost možemo primijetiti između te oznake i druge „li“ oznake tj. između njihovih klasa, no glavna razlika je u tome što druga oznaka ima jedan razmak poslije imena klase te nam zato ta klasa ne stvara probleme.

```
try
{
    var paginationTag = doc.DocumentNode.SelectNodes("//ul[contains(@class, 'pagination')]")[0];
    var numberTag = paginationTag.Descendants()
        .Single(node => node.Name == "li" &&
        node.Attributes.Any(attr => attr.Name == "class" && attr.Value == "number"));
    int.TryParse(numberTag.InnerText, out numberOfPages);
}
catch (Exception ex)
{
    Console.WriteLine(ex.Message);
}
```

Slika 8. Dohvaćanje broja stranica u C#

```
try
    let paginationTag = doc.DocumentNode.SelectNodes("//ul[contains(@class, 'pagination')]")[0]
    let numberTag = paginationTag.Descendants().Single(fun node -> node.Name = "li" && node.Attr
    numberOfPages <- int numberTag.InnerText
with
    | :? System.InvalidOperationException as ex ->
    printfn "Error: %s" ex.Message
```

Slika 9. Dohvaćanje broja stranica u F#

```
ributes.Any(fun attr -> attr.Name = "class" && attr.Value = "number"))
```

Slika 10. Slika 9. - 2. dio

Dohvaćanje paginacije sam morao staviti unutar „try catch/with“ bloka jer postoji mogućnost da će svi rezultati stati na jednu stranicu te neće biti potrebe za paginacijom.

U kôdu možemo primijetiti tok kojim dohvaćamo broj stranica; u prvu varijablu „paginationTag“ spremamo cijelu „ul“ oznaku iz koje kasnije prvo vadimo sve podređene oznake sa metodom Descendants, a onda primjenjujemo lambda funkciju na njima. Pomoću lambda funkcije dohvaćamo onu oznaku koja ima ime „li“ i koja ima atribut sa imenom „class“ i vrijednošću „number“. Kada nađemo takvu klasu, njen „InnerText“ pokušamo pretvoriti u tip vrijednosti „int“ te ga pospremamo u ranije definiranu varijablu „numberOfPages“.

Što se tiče sintakse u ovom dijelu koda vidimo velike sličnosti zbog korištenja iste vanjske biblioteke u oba jezika što nije čudno jer su oba jezika iz .NET domene te mi to daje mogućnost korištenja istih Nuget paketa u oba jezika. Značajniju razliku u sintaksi između F# i većine ostalih programskih jezika vidimo u operatoru za uspoređivanje vrijednosti. Dok smo navikli u poznatim jezicima koristiti dva znaka jednakosti za uspoređivanje, a jedan za dodjeljivanje vrijednosti varijabli, F# koristi jedan znak jednakosti za uspoređivanje. Dodjeljivanje vrijednosti varijablama F# radi na 2 načina, prvi kada se varijabla inicijalizira koristi također jedan znak jednakosti, no ako želimo kasnije mijenjati vrijednost varijable moramo ju prilikom inicijalizacije označiti ključnom riječi „mutable“ a kasnije joj dodijeliti vrijednost sa „strelicom“ koja ide iz vrijednosti u varijablu.

```
let mutable numberOfPages: int = 1
```

```
numberOfPages <- int numberTag.InnerText
```

Nakon dohvaćanja broja stranice krećemo sa dohvaćanjem proizvoda iz rezultata pretraživanja. Prvo dohvaćamo prvu stranicu te onda ako ih ima više ulazimo u „for“ petlju da ih dohvatimo sve jednu po jednu. Eventualno, ovaj proces sam mogao riješiti rekurzijom prosljeđivanje sljedećeg broja stranice u sljedeću iteraciju, no mislim da bi to moglo otići previše u dubinu s obzirom na to da neki rezultati imaju preko 100 stranica, a i sam proces dohvaćanja HTML-a već ovisi o brzini interneta pa bi sve skupa bilo presporo za izvođenje.

```
let mutable productTags = doc.DocumentNode.SelectNodes("//div[contains(@class,'item') and contains(@class, 'product-item')]")

for tag in productTags do
    products <- Array.append products (getProductFromHtml(tag))

if numberOfPages > 1 then
    for i in 1..numberOfPages-1 do
        printfn "%s" ("Going through page " + string(i+1))
        url <- compoundUrl("https://www.sportvision.hr/", i, input)
        html <- getHtml(url) |> Async.RunSynchronously
        (doc.LoadHtml(html))
        let newProducts = doc.DocumentNode.SelectNodes("//div[contains(@class,'item') and contains(@class, 'product-item')]")

        for tag in newProducts do
            products <- Array.append products (getProductFromHtml(tag))
```

Slika 11. Dohvaćanje proizvoda F#

```
var productTags = doc.DocumentNode.SelectNodes("//div[contains(@class,'item') and contains(@class, 'product-item')]");

foreach(var tag in productTags)
{
    products.Add(GetProductFromHtml(tag));
}

if(numberOfPages > 1)
{
    for(var i = 1; i < numberOfPages; i++)
    {
        Console.WriteLine("Going through page " + (i + 1).ToString());
        url = CompoundUrl("https://www.sportvision.hr/", i, input);
        html = GetHtml(url).Result;
        doc.LoadHtml(html);
        var newProducts = doc.DocumentNode.SelectNodes("//div[contains(@class,'item') and contains(@class, 'product-item')]");

        foreach(var tag in newProducts)
        {
            products.Add(GetProductFromHtml(tag));
        }
    }
}
```

Slika 12. Dohvaćanje proizvoda C#

Prilikom dohvaćanja proizvoda sa ostalih stranica ponavljamo korake opisane dosad, slažemo URL koji ćemo poslat, dohvaćamo HTML sa tog URL-a i učitavamo ga u varijablu. Sljedeći korak nakon toga je filtriranje onih HTML oznaka koje u sebi imaju klase „item“ i „product-item“ jer se one odnose na jedan proizvod u popisu.

```
<div class="wrapper-grid-view item product-item ease col-xs-6 col-sm-4 col-md-3 col-lg-3  l
ist-class grid-view    list-class have-similar-product-slider" data-productposition="21"
data-productid="835489" data-product-item-id="1201A477-001" data-productname="ASICS TENISICE
LYTE CLASSIC " data-productcode="1201A477-001" data-productcategoryid="1" data-
productcategory data-productcat="TENISICE" data-productcatbread="Proizvodi > OBUĆA > TENISIC
E" data-productbrand="ASICS" data-productprice="63,60" data-productdiscount="20" data-
productprevprice="79,50">…</div>
<div class="wrapper-grid-view item product-item ease col-xs-6 col-sm-4 col-md-3 col-lg-3  l
ist-class grid-view    list-class" data-productposition="22" data-productid="810254" data-
product-item-id="149055-BKW" data-productname="SKECHERS TENISICE ARCH FIT " data-
productcode="149055-BKW" data-productcategoryid="1" data-productcategory data-productcat="TE
NISICE" data-productcatbread="Proizvodi > OBUĆA > TENISICE" data-productbrand="SKECHERS"
data-productprice="67,73" data-productdiscount="30" data-productprevprice="96,75">…</div>
 == $0
<div class="wrapper-grid-view item product-item ease col-xs-6 col-sm-4 col-md-3 col-lg-3  l
ist-class grid-view    list-class" data-productposition="23" data-productid="792795" data-
product-item-id="232290-NVY" data-productname="SKECHERS TENISICE SQUAD " data-productcode="2
32290-NVY" data-productcategoryid="1" data-productcategory data-productcat="TENISICE" data-
productcatbread="Proizvodi > OBUĆA > TENISICE" data-productbrand="SKECHERS" data-
```

Nakon što filtriramo sve takve „div“ elemente, svaki od njih prosljeđujemo u funkciju koja će izdvojiti podatke bitne za model proizvoda u aplikaciji.

```fsharp
type Product(name: string, price: float, discount: int, url: string) =
    let mutable _name = name
    let mutable _price = price
    let mutable _discount = discount
    let mutable _url = url

    member this.Name with get() = _name and set(value) = _name <- value
    member this.Price with get() = _price and set(value) = _price <- value
    member this.Discount with get() = _discount and set(value) = _discount <- value
    member this.Url with get() = _url and set(value) = _url <- value
```

Slika 13. Model proizvoda u F#

```
10 references | Leo Vukoje, 10 hours ago | 1 author, 1 change
class Product
{
    2 references | Leo Vukoje, 10 hours ago | 1 author, 1 change
    public string? Name { get; set; }
    4 references | Leo Vukoje, 10 hours ago | 1 author, 1 change
    public decimal? Price { get; set; }
    4 references | Leo Vukoje, 10 hours ago | 1 author, 1 change
    public int? Discount { get; set; }
    2 references | Leo Vukoje, 10 hours ago | 1 author, 1 change
    public string? Url { get; set; }

    1 reference | Leo Vukoje, 10 hours ago | 1 author, 1 change
    public Product(string? name, decimal price, int discount, string? url)
    {
        Name = name;
        Price = price;
        Discount = discount;
        Url = url;
    }
}
```

Slika 14. Model proizvoda u C#

Vidimo da u oba jezika za „property-je“ moramo definirati njihove gettere i settere. S obzirom na veću podršku za C# taj je proces pojednostavljen u odnosu na kako je bilo prije, ali u F# vidimo kako izgleda detaljno definiranje tih funkcija. Naravno istu stvar možemo napraviti i u C# u slučaju da nam ne treba konvencionalni get i set već neka drukčija logika. U C# također možemo postaviti parametre vidljivosti na te funkcije, tako da npr. get može biti public, dok set može biti samo unutar funkcije i brojne druge kombinacije.

```
2 references | Leo Vukoje, 10 hours ago | 1 author, 1 change
private static Product GetProductFromHtml(HtmlNode html)
{
    var titleTag = html.Descendants().FirstOrDefault(node => node.Name == "div" && node.Attributes.Any(attr => a
    var link = titleTag?.Descendants().FirstOrDefault(node => node.Name == "a");
    var name = link?.GetAttributeValue("title", "");
    var url = link?.GetAttributeValue("href", "");

    var pricesWrapperTag = html.Descendants().FirstOrDefault(node => node.Name == "div" && node.Attributes.Any(a
    var currentPriceTag = pricesWrapperTag?.Descendants().FirstOrDefault(node => node.Name == "div" && node.Attr
    var trimmedCurrentPriceTag = currentPriceTag?.InnerText.Trim();
    if (!decimal.TryParse(trimmedCurrentPriceTag![..(trimmedCurrentPriceTag!.Length - 1)], out decimal price))
    {
        Console.WriteLine("Failed to get price for " + name);
        price = 0;
    }

    var discount = 0;

    try
    {
        var discountDiv = pricesWrapperTag?.Descendants().Single(node => node.Name == "div" && node.Attributes.A
        var discountTrimmed = discountDiv?.InnerText.Trim();
        _ = int.TryParse(discountTrimmed?.Substring(discountTrimmed.Length - 3, 2), out discount);
    }
    catch (Exception)
    {
        discount = 0;
    }

    return new Product(name, price, discount, url);
}
```

Slika 15. Mapiranje HTML-a u Product - C#

```
let getProductFromHtml(html: HtmlNode) =
        let titleTag = html.Descendants().FirstOrDefault(fun node -> node.Name = "div" && node.Att
        let link = titleTag.Descendants().FirstOrDefault(fun node -> node.Name = "a")
        let name: string = link.GetAttributeValue("title", "")
        let url: string = link.GetAttributeValue("href", "")

        let pricesWrapperTag = html.Descendants().FirstOrDefault(fun node -> node.Name = "div" &&
        let currentPriceTag = pricesWrapperTag.Descendants().FirstOrDefault(fun node -> node.Name
        let price = float (currentPriceTag.InnerText.Trim().Substring(0, 5).Replace(",", "."))

        let mutable discount: int = 0

        try
            let discountDiv = pricesWrapperTag.Descendants().Single(fun node -> node.Name = "div"
            let discountTrimmed = discountDiv.InnerText.Trim()
            discount <- int (discountTrimmed.Substring(discountTrimmed.Length - 3, 2))
        with
            | :? System.InvalidOperationException as ex ->
            discount <- 0

        Array.create 1 (Product(name, price, discount, url))
```

Slika 16. Mapiranje HTML-a u Product - F#

U ovim metodama vidimo kako tražimo HTML oznake za naslov, URL proizvoda, cijenu i popust ako postoji. Sve te izdvojene informacije se na kraju vraćaju u obliku novog objekta klase Product sa svim informacijama.

Kada je definirana logika za dohvaćanje proizvoda i njihovo spremanje, treba definirati interakciju s korisnikom. U ovom slučaju to je napravljeno kroz tekstualni izbornik u konzoli gdje korisnik prvo upisuje pojam pretrage, zatim čeka da se svi rezultati dohvate i onda te rezultate prikazuje pomoću bilo koje od ponuđenih opcija.

```
let mutable menuChoice = mainMenu()
printfn "%s" "\n"

while menuChoice <> "0" do

    printf "%s" "Koliko proizvoda za prikazati: "
    let n = int(Console.ReadLine())


    if menuChoice = "1" then
        shellSortByPrice products
        printProducts(products, n)
    else if menuChoice = "2" then
        shellSortByPrice products
        printProducts((Array.rev products), n)
    else if menuChoice = "3" then
        shellSortByDiscount products
        printProducts(products, n)
    else if menuChoice = "4" then
        shellSortByDiscount products
        printProducts((Array.rev products), n)
    else
        printfn "%s" "Krivi unos!"

    menuChoice <- mainMenu()
    printfn "%s" "\n"
```

Slika 17. Izbornik F#

```fsharp
let mainMenu() =
    printfn "%s" "1) Sortiraj po cijeni - od najnize"
    printfn "%s" "2) Sortiraj po cijeni - od najvise"
    printfn "%s" "3) Sortiraj po popustu - od najnizeg"
    printfn "%s" "4) Sortiraj po popustu - od najviseg"
    printfn "%s" "0) Kraj koristenja"
    printfn "%s" "\n"

    printf "%s" "Odabir: "
    Console.ReadLine()
```

Slika 18. Ispis izbornika F#

Ovdje jasno možemo uočiti način u kojem se u F# vraćaju vrijednosti iz funkcija, dakle ne postoji konvencionalna ključna riječ „return“ već se samo navede varijabla bez ičega i to se smatra povratnom vrijednosti funkcije.

```
2 references | Leo Vukoje, 11 hours ago | 1 author, 1 change
private static string MainMenu()
{
    Console.WriteLine("1) Sortiraj po cijeni - od najnize");
    Console.WriteLine("2) Sortiraj po cijeni - od najvise");
    Console.WriteLine("3) Sortiraj po popustu - od najnizeg");
    Console.WriteLine("4) Sortiraj po popustu - od najviseg");
    Console.WriteLine("0) Kraj koristenja");
    Console.WriteLine("\n");

    Console.Write("Odabir: ");
    return Console.ReadLine()!;
}
```

Slika 19. Ispis izbornika C#

```
Console.WriteLine();
var menuChoice = MainMenu();

while(menuChoice != "0")
{
    Console.Write("Koliko proizvoda za prikazati: ");
    _ = int.TryParse(Console.ReadLine(), out int n);

    if(menuChoice == "1")
    {
        ShellSortByPrice(products);
        PrintProducts(products, n);
    } else if(menuChoice == "2")
    {
        ShellSortByPrice(products);
        PrintProducts(products.Reverse().ToList(), n);
    } else if(menuChoice == "3")
    {
        ShellSortByDiscount(products);
        PrintProducts(products, n);
    } else if(menuChoice == "4")
    {
        ShellSortByDiscount(products);
        PrintProducts(products.Reverse().ToList(), n);
    } else
    {
        Console.WriteLine("Krivi unos!");
    }

    menuChoice = MainMenu();
    Console.WriteLine();
}
```

Slika 20. Izbornik C#

Po imenima metoda vidimo da za sortiranje postoje napisani *shell sort* algoritmi koji sortiraju objekte samo po drugoj vrijednosti u objektu (cijena ili popust)

```
2 references | Leo Vukoje, 11 hours ago | 1 author, 1 change
private static void ShellSortByPrice(IList<Product> products)
{
    int n = products.Count;
    int gap = n / 2;

    while (gap > 0)
    {
        for (int i = gap; i < n; i++)
        {
            Product temp = products[i];
            int j = i;

            while (j >= gap && products[j - gap].Price > temp.Price)
            {
                products[j] = products[j - gap];
                j -= gap;
            }

            products[j] = temp;
        }

        gap /= 2;
    }
}
```

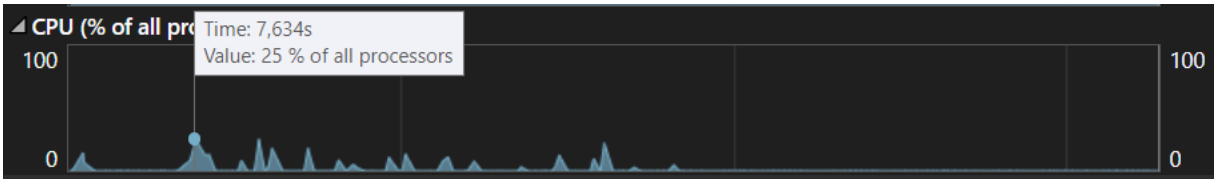
Slika 21. Sortiranje po cijeni C#

```
let shellSortByDiscount (arr:Product[]) =
    let mutable gap = arr.Length / 2
    while gap > 0 do
        for i in gap..arr.Length-1 do
            let mutable temp = arr.[i]
            let mutable j = i
            while j >= gap && arr.[j-gap].Discount > temp.Discount do
                arr.[j] <- arr.[j-gap]
                j <- j - gap
            arr.[j] <- temp
        gap <- gap / 2
```

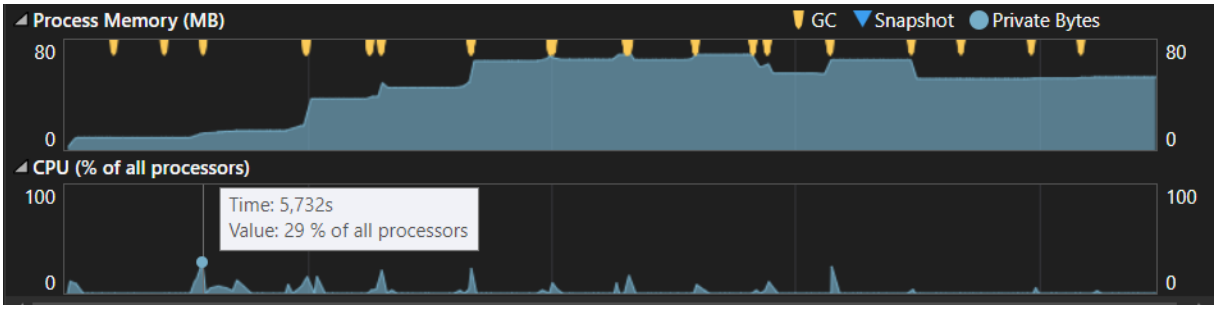
Slika 22. Sortiranje po popustu F#

Pokretanje programa C# skripte u Debug načinu i omogućavanjem dijagnostičkih alata mogao sam provjeriti performanse i zahtjeve aplikacije te sam tako vidio da aplikacije pri jednom korištenju nije potrošila više od 80MB memorije što ju čini vrlo laganom.

Ista stvar vrijedi i za potrošnju procesora, gdje je najveći „spike“ dosegnuo 25% procesora



Pri pokretanju F# skripte dobiju se slični rezultati, RAM također ne prelazi 80MB, a potrošnja procesora je bila najviše 29% u jednom trenutku.



Točno usporediti brzine izvođenja je dosta teško s obzirom na to da uvelike ovisi o brzini dohvaćanja HTML-a i još k tome da se ti zahtjevi mogu slati i po više puta s obzirom na broj stranica.

Upute za pokretanje:

Projekti su svaki zasebno buildani unutar svojih direktorija tako da za pokretanje C# skripte idite na putanju **Web-scraping-project\CSharp\bin\Release\net6.0** i tamo pokrenite **CSharp.exe**.

Za pokretanje F# skripte, na putanji **Web-scraping-project\FSharp\bin\Release\net7.0** pokrenite **FShrap.exe**

Prijedlozi unosa pojma pretrage za testiranje:

„sportske patike“ – 1 stranica sa 3 rezultata

„patike“ – 2 stranice rezultata

„polo majica“ – 8 stranica rezultata

„tenisice“ – 105 stranica rezultata

Ako slučajno neki od tih executable-a ne radi, u Web-scraping-project postoji .sln datoteka koju možete otvoriti u Visual Studio alatu, te tamo pokrenuti projekte zasebno