Ajdin Šuta 19163

White box testiranje – izvještaj

Metoda SelectionSort(List<Course> Index) u CourseController-u:

```
public List<Course> SelectionSort(List<Course> Index)
{
    for (int i = 0; i < Index.Count - 1; i++)
    {
        int minIndex = i;
        for (int j = i + 1; j < Index.Count; j++)
        {
            if (Index[j].Semester < Index[minIndex].Semester)
            {
                minIndex = j;
            }
        }
        if (minIndex != i)
        {
            (Index[minIndex], Index[i]) = (Index[i], Index[minIndex]);
        }
    }
    return Index;
}
```

- Line Coverage

  TC1:

```
[TestMethod]
public void SelectionSort_ShouldSortCoursesBySemester()
{
    // Arrange
    var courses = new List<Course>
    {
        new Course { ID = 1, Name = "Course 1", Semester = 3 },
        new Course { ID = 2, Name = "Course 2", Semester = 1 },
        new Course { ID = 3, Name = "Course 3", Semester = 2 }
    };
    var expectedSortedCourses = new List<Course>
    {
        new Course { ID = 2, Name = "Course 2", Semester = 1 },
        new Course { ID = 3, Name = "Course 3", Semester = 2 },
        new Course { ID = 1, Name = "Course 1", Semester = 3 }
    };

    // Act
    var sortedCourses = _controller.SelectionSort(courses);

    // Assert
    CollectionAssert.AreEqual(expectedSortedCourses, sortedCourses);
}
```

Dovoljan je jedan testni slučaj za postizanje 100% linijske pokrivenosti jer će prolaziti kroz listu, dešavat će se sortiranje elemenata u listi koje zahtjevaju provjere.

- Branch Coverage

  TC1:

```
[TestMethod]
public void test()
{

    List<Course> courses = new List<Course>
    {
        new Course { Semester = 3 },
        new Course { Semester = 1 },
        new Course { Semester = 4 },
        new Course { Semester = 2 }
    };

    List<Course> expected = new List<Course>
    {
        new Course { Semester = 1 },
        new Course { Semester = 2 },
        new Course { Semester = 3 },
        new Course { Semester = 4 }
    };

    // Act
    List<Course> result = _courseController.SelectionSort(courses);

    // Assert
    CollectionAssert.AreEqual(expected, result, Comparer<Course>.Create((x,
y) => x.Semester.CompareTo(y.Semester)));
}
```

- Condition coverage

```
TC1:
[TestMethod]
public void TestSelectionSort()
{
    // Arrange
    var course1 = new Course { Semester = 3 };
    var course2 = new Course { Semester = 1 };
    var course3 = new Course { Semester = 2 };
    var courses = new List<Course> { course1, course2, course3 };

    // Act
    var sortedCourses = _courseController.SelectionSort(courses);
```

```
            // Assert
            Assert.AreEqual(course2, sortedCourses[0]);
            Assert.AreEqual(course3, sortedCourses[1]);
            Assert.AreEqual(course1, sortedCourses[2]);
        }
```

- Loop coverage

  TC1:

  Prazna lista – Vraćena prazna lista

```
        // TC1: Empty List - An empty list is returned
        List<Course> emptyList = new List<Course>();
        List<Course> sortedEmptyList = SelectionSort(emptyList);
        Assert.IsEmpty(sortedEmptyList);
```

  TC2:

  Lista sa jednim elementom – Vraćena ista lista jer je već sortirana

```
            // TC2: List with one element - The same list is returned because it is
already sorted
    List<Course> oneElementList = new List<Course> { new Course() };
    List<Course> sortedOneElementList = SelectionSort(oneElementList);
    Assert.AreEqual(oneElementList, sortedOneElementList);
```

  TC3:

  Lista sa dva elementa (Nesortirana) – Vraća sortiranu listu

```
        // TC3: List with two elements (Unsorted) - Returns a sorted list
    List<Course> unsortedTwoElementList = new List<Course> { new Course(), new
Course() };
    List<Course> sortedTwoElementList = SelectionSort(unsortedTwoElementList);
    Assert.IsTrue(IsSorted(sortedTwoElementList));
```

  TC4:

  Lista sa dva elementa (Sortirani) – Vraća istu sortiranu listu

```
// TC4: List with two elements (Sorted) - Returns the same sorted list
    List<Course> sortedTwoElementList = new List<Course> { new Course(), new
Course() };
    List<Course> sortedTwoElementList = SelectionSort(sortedTwoElementList);
    Assert.AreEqual(sortedTwoElementList, sortedTwoElementList);
```

  TC5:

  Lista sa više elemenata (Nesortirani) – Vraća sortiranu listu

```
// TC5: Multi-Element List (Unsorted) - Returns a sorted list
```

```
    List<Course> unsortedMultiElementList = new List<Course> { new Course(), new
Course(), new Course() };
    List<Course> sortedMultiElementList = SelectionSort(unsortedMultiElementList);
    Assert.IsTrue(IsSorted(sortedMultiElementList));
```

TC6:

Lista sa više elemenata (Sortirani) – Vraća sortiranu listu

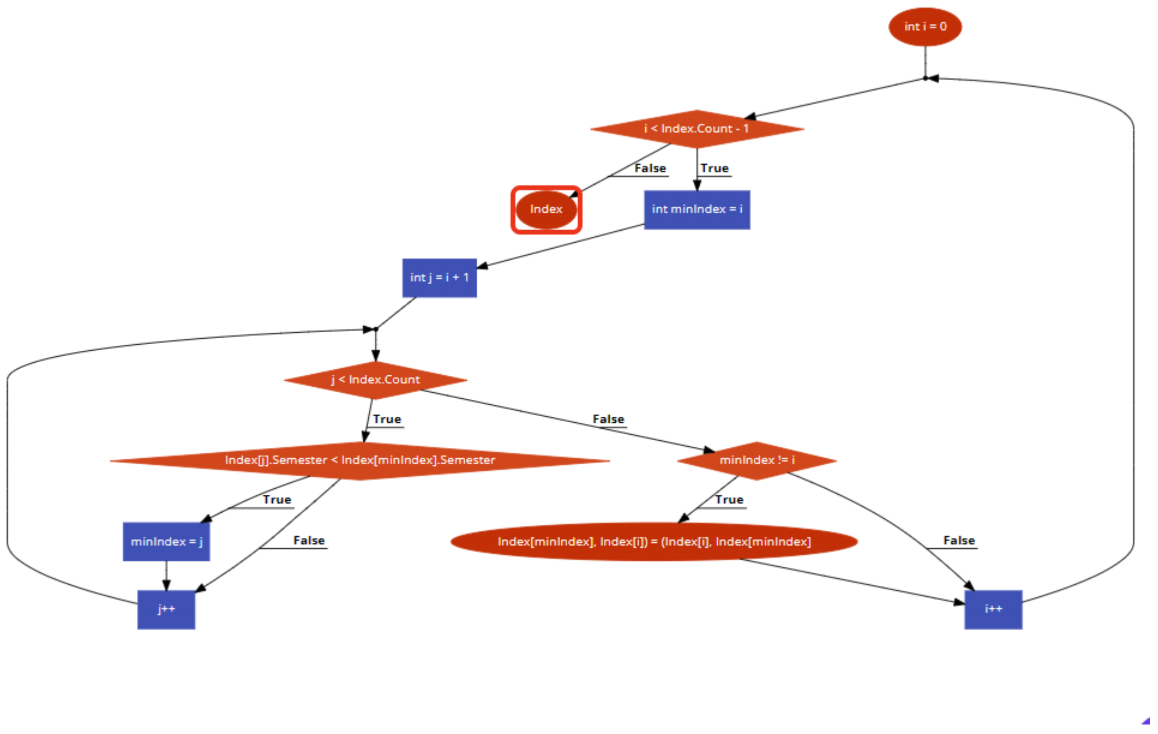
```
// TC6: List with multiple elements (Sorted) - Returns a sorted list
    List<Course> sortedMultiElementList = new List<Course> { new Course(), new
Course(), new Course() };
    List<Course> sortedMultiElementList = SelectionSort(sortedMultiElementList);
    Assert.AreEqual(sortedMultiElementList, sortedMultiElementList);
```

TC7:

Lista sa n, n-1, n+1 elementa – Vraća sortiranu listu

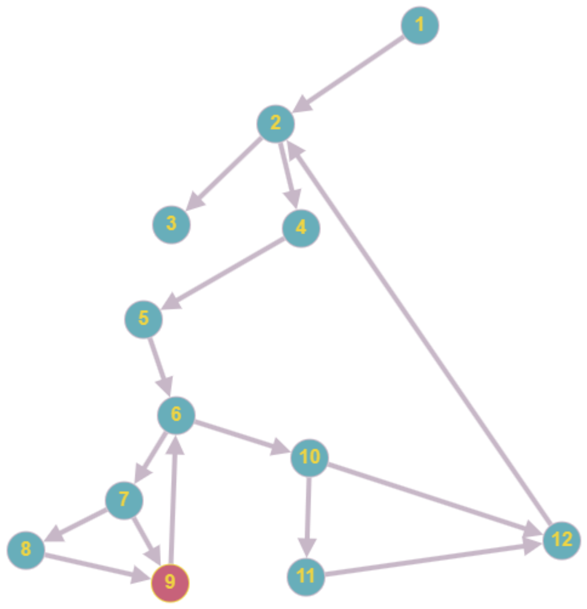
```
// TC7: List with n, n-1, n+1 elements - Returns a sorted list
    List<Course> nElementList = new List<Course> { new Course(), new Course(), new
Course() };
    List<Course> sortedNElementList = SelectionSort(nElementList);
    Assert.IsTrue(IsSorted(sortedNElementList));
}

private bool IsSorted(List<Course> list)
{
    for (int i = 0; i < list.Count - 1; i++)
    {
        if (list[i].Semester > list[i + 1].Semester)
        {
            return false;
        }
    }
    return true;
}
```

- Path coverage



Slika 1: Dijagram toka modula

Ajdin Šuta 19163



Slika 2: Programski graf modula

Broj puta	Put
1	1-2-3
2	1-2-4-5-6-7-8-9-6-10-11-12-2-3
3	1-2-4-5-6-7-8-9-6-10-12-2-3
4	1-2-4-5-6-7-9-6-10-11-12-2-3
5	1-2-4-5-6-7-9-6-10-12-2-3
6	1-2-4-5-6-7-8-9-6-7-8-9-6-10-11-12-2-3
7	1-2-4-5-6-7-8-9-6-7-8-9-6-10-12-2-3
8	1-2-4-5-6-10-11-12-2-3
9	1-2-4-5-6-10-12-2-3
10	1-2-4-5-6-7-9-6-7-8-9-6-10-11-12-2-3
11	1-2-4-5-6-7-9-6-7-8-9-6-10-12-2-3

Uočava se 11 različitih puteva što pokazuje da je potrebno najmanje 11 testnih slučajeva da bi se postigla potpuna obuhvatnost puteva.

Ajdin Šuta 19163

```
[TestInitialize]
public void setup()
{
    var configuration = new
ConfigurationBuilder().AddJsonFile("appsettings.test.json").Build();
    var options = new DbContextOptionsBuilder<ApplicationDbContext>()
    .UseSqlServer(configuration.GetConnectionString("DefaultConnection"))
    .Options;
    _context = new ApplicationDbContext(options);


    _mockContext = new Mock<ApplicationDbContext>(options);
    _mockUserManager = new Mock<UserManager<Person>>(new
Mock<IUserStore<Person>>().Object, null, null, null, null, null, null, null, null);
}



public async Task testPath1()
{
    //1-2-4-5-6-7-8-9-6-10-11-12-2-3
    //Arrange
    List<Course> list = new List<Course> { new Course { Semester = 3 }, new Course {
Semester = 1 } };
    var controller = new CourseController(_mockContext.Object,
_mockUserManager.Object);

    //Act
    var result = controller.SelectionSort(list);

    //Assert
    Assert.IsTrue(result.Any(c => c.Semester < result.IndexOf(c)))
}

[TestMethod]
public async Task testPath2()
{
    //1-2-4-5-6-7-9-6-10-12-2-3
    //Arrange
    List<Course> list = new List<Course> { new Course { Semester = 3 }, new Course {
Semester = 2 }, new Course { Semester = 1 } };
    var controller = new CourseController(_mockContext.Object,
_mockUserManager.Object);

    //Act
    var result = controller.SelectionSort(list);

    //Assert
    Assert.IsTrue(result.Any(c => c.Semester < result.IndexOf(c)));
}
```