Univerzitet u Sarajevu
Elektrotehnički fakultet
Sarajevo

Unit testiranje

Članovi tima:

Mirnes Fehrić (vođa)
Emin Begić
Aldin Velić
Zana Beljuri

Sarajevo, Novembar 2025

1 Uvod

U okviru zadatka smo implementirali skup unit testova koji pokrivaju glavne slojeve i funkcionalnosti sistema `TeamTaskManager`: servisni sloj (`TaskService`), repozitorij (`InMemoryTaskRepository`) i pomoćnu klasu za analizu zadataka (`TaskAnalyzer`). Klasu `Program` nismo testirali u skladu sa specifikacijom zadatka.

Cilj je bio postići minimalno 90% pokrivenosti koda po klasama, testirati osnovne funkcionalnosti i izuzetke, primijeniti data driven testiranje u najmanje dvije forme te prikazati korištenje zamjenskih objekata (*mock* repozitorija) za module koji nisu u fokusu testiranja.

2 Pregled testnih klasa

Implementirana su tri glavna skupa testova:

- `TaskServiceTests` – testiranje servisnog sloja i poslovne logike.
- `TaskRepositoryTests` – testiranje repozitorija u memoriji.
- `TaskAnalyzerTests` – testiranje analitičke logike nad zadacima.

Dodatno, za ilustraciju naprednijeg data driven testiranja sa vanjskim datotekama implementirane su i dvije pomoćne testne klase:

- `XmlTests` – data driven testovi zasnovani na čitanju podataka iz XML datoteke `taskdata.xml`.
- `CsvTests` – data driven testovi zasnovani na čitanju podataka iz CSV datoteke `taskdata.csv`.

2.1 `TaskServiceTests`

Ova testna klasa provjerava većinu javnih metoda servisa `TaskService`:

- Dohvat korisnika:
  - `GetAllUsers_ShouldReturnFourSeededUsers`
  - `GetUserById_ExistingId_ShouldReturnUser`
  - `GetUserById_NotExistingId_ShouldReturnNull`
- Dohvat zadataka:
  - `GetAllTasks_ShouldReturnAllFromRepository`
  - `GetTasksForUser_ShouldReturnOnlyTasksForThatUser`
  - `GetTasksByStatus_ShouldReturnOnlyWithThatStatus`
  - `GetTasksByPriority_ShouldReturnOrderedByDueDate`
  - `GetTaskById_Existing_ShouldReturnTask`
  - `GetTaskById_NotExisting_ShouldReturnNull`
- Kreiranje zadatka:
  - `CreateTask_ValidData_ShouldAddTaskToRepository`
  - `CreateTask_EmptyTitle_ShouldThrow` (test izuzetka uz `[ExpectedException]`)
- Ažuriranje statusa zadatka:
  - `UpdateTaskStatus_ExistingTask_ShouldChangeStatus`
  - `UpdateTaskStatus_NotExistingTask_ShouldThrow`
- Brisanje zadatka:

  - `DeleteTask_Existing_ShouldReturnTrueAndRemove`
  - `DeleteTask_NotExisting_ShouldReturnFalse`
- Naprednije operacije:
  - `SearchTasks_ByText_DataRow` (data driven test)
  - `SearchTasks_FilterByAssignedUser`
  - `SearchTasks_FilterByPriority`
  - `SearchTasks_OnlyNotOverdue_ShouldExcludeExpired`
  - `SearchTasks_SortByDueDateAsc`
  - `SearchTasks_SortByPriorityDesc`
  - `SearchTasks_SortByCreatedDateDesc`
  - `BulkUpdateStatus_ValidIds_ShouldUpdateAll`
  - `BulkUpdateStatus_EmptyList_ShouldThrow`
  - `BulkUpdateStatus_IdNotFound_ShouldThrow`
  - `BulkUpdateStatus_TaskAlreadyDone_ShouldThrow`
  - `SeedDemoData_ShouldCreateThreeTasks`
  - `GetReport_ShouldReturnCorrectCounts`
- Testovi sa zamjenskim objektima (*mock* `ITaskRepository`):
  - `CreateTask_ShouldCallRepositoryAddTask`
  - `UpdateTaskStatus_ShouldCallRepositoryUpdateTask`
  - `DeleteTask_WhenRepositoryReturnsFalse_ShouldReturnFalse`
  - `GetTasksForUser_ShouldCallRepositoryGetTasksByUser`
- Dodatni data driven test:
  - `SearchTasks_DynamicData_ShouldReturnCorrectResults`

2.2 `TaskRepositoryTests`

Ova testna klasa pokriva osnovne CRUD operacije nad `InMemoryTaskRepository`:

- Dodavanje zadatka:
  - `AddTask_CallTheMethod_ShouldAddTask`
  - `AddTask_NullTask_ShouldThrowException`
  - `AddTask_MissingPriority_ShouldThrowException`
- Dohvat zadatka po ID-u:
  - `GetTaskById_CallTheMethod_ShouldReturn`
  - `GetTaskById_NonExistentId_ShouldReturnNull`
- Dohvat svih zadataka:
  - `GetAllTasks_CallTheMethod_ShouldReturnAllTasks`
  - `GetAllTasks_EmptyRepository_ShouldReturnEmptyList`
- Brisanje zadatka:
  - `DeleteTask_CallTheMethod_ShouldDeleteTask`
  - `DeleteTask_NonExistentId_ShouldReturnFalse`
- Ažuriranje zadatka:
  - `UpdateTask_CallTheMethod_ShouldUpdateTask`

2.3 TaskAnalyzerTests

Ova testna klasa pokriva napredniju logiku u TaskAnalyzer:

- Metoda EvaluateTaskStatus:
  - EvaluateTaskStatus_NullTask_ReturnsInvalid
  - EvaluateTaskStatus_CompletedTask_ReturnsCompleted
  - EvaluateTaskStatus_OverdueCritical_ReturnsCriticalMessage
  - EvaluateTaskStatus_NoDeadline_ReturnsNoDeadline
  - EvaluateTaskStatus_OnTrackHigh_ReturnsOnTrackHigh
- Metoda AnalyzeTeamPerformance:
  - AnalyzeTeamPerformance_EmptyRepo_ReturnsNoData
  - AnalyzeTeamPerformance_NormalRepo_ReturnsText
- Metoda GetTasksSummaryForUser:
  - GetTasksSummaryForUser_NoTasks_ReturnsZero
  - GetTasksSummaryForUser_WithTasks_ReturnsDictionaryWithValues
- Metoda PredictDelayRisk:
  - PredictDelayRisk_NullTask_ReturnsNepoznat
  - PredictDelayRisk_HighPrioritySoonDue_ReturnsHighRisk
  - PredictDelayRisk_DoneOldTask_ReturnsNizakRizik

3 Data driven testiranje

Data driven testiranje smo implementirali u više različitih formi, u skladu sa zahtjevom zadatka.

3.1 Testovi sa atributom [DataRow]

Metoda SearchTasks_ByText_DataRow u klasi TaskServiceTests koristi MSTest atribut [DataRow] kako bi ista testna metoda bila izvršena sa više skupova ulaznih podataka:

- [DataRow("Task ", 3)] – očekujemo da sva tri zadatka budu pronađena.
- [DataRow("nepostoji", 0)] – očekujemo da nijedan zadatak ne bude pronađen.

Na ovaj način smo jednim testom pokrili različite scenarije pretrage po tekstu, bez dupliciranja programske logike u samom testu.

3.2 Testovi sa [DynamicData] i kolekcijom podataka

Druga forma data driven testiranja je realizovana kroz [DynamicData] atribut i svojstvo SearchTestData koje vraća IEnumerable<object[]>:

- new object[] { "Task", 3 }
- new object[] { "1", 1 }
- new object[] { "Desc", 3 }
- new object[] { "xxx", 0 }

Test `SearchTasks_DynamicData_ShouldReturnCorrectResults` zatim za svaki red iz `SearchTestData` izvršava isti obrazac:

1. Postavi se `TaskSearchOptions.Text` na zadani tekst.
2. Poziva se `SearchTasks`.
3. Provjerava se da broj rezultata odgovara očekivanoj vrijednosti.

Na ovaj način smo pokazali data driven testiranje sa pohranjenim podacima u dvije forme: direktno u atributima (`DataRow`) i kroz odvojenu kolekciju (`DynamicData`).

3.3 Testovi sa `[DynamicData]` nad XML datotekom

Dodatno smo implementirali i naprednije data driven testiranje zasnovano na vanjskoj XML datoteci `taskdata.xml`, u posebnoj testnoj klasi `XmlTests`. Za ovo smo koristili:

- `using Microsoft.VisualStudio.TestTools.UnitTesting;`
- `using System;`
- `using System.Collections.Generic;`
- `using System.IO;`
- `using System.Xml.Linq;`
- alias `TaskStatusModel = TeamTaskManager.Model.TaskStatus;`

Metoda `XmlData` je statička metoda koja vraća `IEnumerable<object[]>` i koristi se kao izvor podataka za `[DynamicData]`:

- Putanja do datoteke se gradi pomoću `AppDomain.CurrentDomain.BaseDirectory` i `Path.Combine`.
- Ukoliko datoteka ne postoji, baca se `FileNotFoundException` sa jasnom porukom (`"XML NOT FOUND!"`).
- XML dokument se učitava preko `XDocument.Load`, a zatim se kroz sve elemente `<Task>` u korijenu generiraju redovi oblika `new object[] { id, title, priority, expected }`.

Testna metoda:

- `[TestMethod]`
- `[DynamicData(nameof(XmlData), DynamicDataSourceType.Method)]`
- `public void Xml_Test(int id, string title, int priority, string expected)`

U okviru testa, početni status zadatka se postavlja na `TaskStatusModel.ToDo`. Zatim se, na osnovu vrijednosti prioriteta, status mijenja:

- Ako je `priority >= 3`, status postaje `TaskStatusModel.InProgress`.
- Ako je `priority == 4`, status postaje `TaskStatusModel.Testing`.

Na kraju se poredi očekivana vrijednost pročitana iz XML-a sa stvarnim string prikazom statusa:

- `Assert.AreEqual(expected, status.ToString());`

Ovim smo pokazali kako se `[DynamicData]` može povezati sa eksternom XML datotekom i kako se testovi mogu lako proširiti dodavanjem novih elemenata u `taskdata.xml`, bez mijenjanja koda testa.

3.4 Testovi sa [DynamicData] nad CSV datotekom

Slično XML pristupu, implementirana je i klasa `CsvTests` koja koristi CSV datoteku `taskdata.csv` kao izvor podataka za data driven testove. Korišteni su sljedeći `using` direktoriji:

- `using Microsoft.VisualStudio.TestTools.UnitTesting;`
- `using System;`
- `using System.Collections.Generic;`
- `using System.IO;`
- alias `TaskStatusModel = TeamTaskManager.Model.TaskStatus;`

Metoda `CsvData` vraća `IEnumerable<object[]>` i radi na sljedeći način:

- Formira se putanja do `taskdata.csv` pomoću `AppDomain.CurrentDomain.BaseDirectory` i `Path.Combine`.
- Ako datoteka ne postoji, baca se `FileNotFoundException` (`"CSV NOT FOUND!"`).
- Čitaju se sve linije datoteke (`File.ReadAllLines(path)`).
- Preskaču se prazne linije (`string.IsNullOrWhiteSpace(line)`).
- Linija se dijeli na dijelove preko `line.Split(',')`.
- Preskače se zaglavlje (kada je prvi element `"Id"`) i svi redovi koji nemaju tačno 4 kolone.
- Za važeće redove generira se `new object[] { int.Parse(p[0]), p[1], int.Parse(p[2]), p[3] }`.

Testna metoda:

- `[TestMethod]`
- `[DynamicData(nameof(CsvData), DynamicDataSourceType.Method)]`
- `public void Csv_Test(int id, string title, int priority, string expected)`

Logika unutar testa identična je XML testu:

- Početni status: `TaskStatusModel.ToDo`.
- Ako je `priority >= 3`, status prelazi u `TaskStatusModel.InProgress`.
- Ako je `priority == 4`, status prelazi u `TaskStatusModel.Testing`.
- Na kraju se provjerava: `Assert.AreEqual(expected, status.ToString());`

Na ovaj način smo demonstrirali treću varijantu data driven testiranja: korištenje `[DynamicData]` sa eksternom CSV datotekom kao izvorom podataka. Dodavanjem novih redova u `taskdata.csv` lako proširujemo skup test primjera.

4 Zamjenski objekti (mock) i izolacija modula

Za ilustraciju unit testiranja sa zamjenskim objektima koristili smo biblioteku `Moq` i interfejs `ITaskRepository`. Ideja je bila da se `TaskService` testira izolovano od konkretne implementacije repozitorija.

Ključni testovi:

- `CreateTask_ShouldCallRepositoryAddTask`
  Zamjenski objekat `Mock<ITaskRepository>` se prosljeđuje u konstruktor `TaskService`. Nakon poziva `CreateTask` provjerava se da li je `AddTask` pozvan tačno jednom (`Times.Once`).
- `UpdateTaskStatus_ShouldCallRepositoryUpdateTask`
  Mock je podešen tako da `GetTaskById(10)` vraća zadatak. Nakon poziva `UpdateTaskStatus(10, TaskStatus.Done)` provjerava se da li je `UpdateTask` pozvan sa zadatkom kojem je promijenjen status.
- `DeleteTask_WhenRepositoryReturnsFalse_ShouldReturnFalse`
  Mock repozitorij vraća `false` za `DeleteTask(999)`. Test provjerava da servis prosljeđuje rezultat bez dodatne logike.
- `GetTasksForUser_ShouldCallRepositoryGetTasksByUser`
  Mock repozitorij vraća unaprijed pripremljenu listu zadataka. Test provjerava broj elemenata i da je metoda `GetTasksByUser` pozvana tačno jednom.

Na ovaj način smo demonstrirali stereotip testova interakcije sa saradnicima (interaction tests) i primjenu zamjenskih objekata za module koji nisu predmet direktnog testiranja.

5 Pokrivenost koda

Za mjerenje pokrivenosti koda korišten je alat za code coverage u sklopu razvojne okoline. Fokus je bio na projektnom sklopu `TeamTaskManager`. Klasa `Program` nije ulazila u mjerenje, u skladu sa zahtjevima.

Na nivou cijelog sklopa postignuti su sljedeći rezultati:

- **TeamTaskManagerTest – Line coverage**: 91.1%
- **TeamTaskManager – Line coverage**: 91.6%

Name	Covered	Uncovered	Coverable	Total	Line coverage	Covered	Total	Branch coverage
+ TaskManagerTest	547	53	600	1065	91.1%	22	40	55%
– TeamTaskManager	342	31	373	609	91.6%	147	194	75.7%
TeamTaskManager.Model.Comment	5	0	5	13	100%	0	0	
TeamTaskManager.Model.InMemoryCommentRepository	14	1	15	31	93.3%	5	6	83.3%
TeamTaskManager.Model.Task	10	0	10	23	100%	0	0	
TeamTaskManager.Model.TaskSearchOptions	5	0	5	20	100%	0	0	
TeamTaskManager.Model.User	4	0	4	16	100%	0	0	
TeamTaskManager.Repository.InMemoryTaskRepository	42	0	42	71	100%	9	10	90%
TeamTaskManager.Service.TaskAnalyzer	78	22	100	153	78%	70	102	68.6%
TeamTaskManager.Service.TaskService	184	8	192	282	95.8%	63	76	82.8%

Slika 1: Prikaz ukupne i pojedinačne pokrivenosti koda testovima

Minimalni uslov od 90% pokrivenosti koda je ispunjen na nivou cijelog projekta. Detaljna pokrivenost po klasama (prema izvještaju sa slike) prikazana je u tabeli ispod.

Klasa	Line coverage	Branch coverage
TeamTaskManager.Model.Comment	100%	–
TeamTaskManager.Model.InMemoryCommentRepository	93.3%	83.3%
TeamTaskManager.Model.Task	100%	–
TeamTaskManager.Model.TaskSearchOptions	100%	–
TeamTaskManager.Model.User	100%	–
TeamTaskManager.Repository.InMemoryTaskRepository	100%	90%
TeamTaskManager.Service.TaskAnalyzer	78%	68.6%
TeamTaskManager.Service.TaskService	95.8%	82.8%

Model klase su u potpunosti pokrivene testovima. Repozitorij `InMemoryTaskRepository` također je u potpunosti pokriven linijama, uz 90% pokrivenosti grananja.

Servisna klasa `TaskService`, koja sadrži najveći dio poslovne logike, postiže 95.8% pokrivenosti linija i 82.8% pokrivenosti grananja.

Najnižu pokrivenost ima `TaskAnalyzer` (78% linija, 68.6% grana), uglavnom zbog kompleksnijih rubnih slučajeva. Ipak, svi funkcionalno najvažniji scenariji korišteni u aplikaciji su pokriveni.

Ukupna pokrivenost pokazuje da projekt ispunjava zahtjev minimalnih 90% pokrivenosti testovima.

6 Doprinosi članova tima

U nastavku je pregled doprinosa svakog člana tima u pisanju unit testova:

Mirnes Fehrić(vođa)

- Dizajn i implementacija većine testova u klasi `TaskServiceTests` za osnovne operacije nad zadacima (kreiranje, dohvat, ažuriranje statusa, brisanje).
- Implementacija testova za `BulkUpdateStatus`, `SeedDemoData` i `GetReport`.
- Usklađivanje testova sa principima: jasan *Arrange-Act-Assert*, dobra imena metoda, izbjegavanje logike u testovima, međusobna neovisnost testova uz `[TestInitialize]` i `[TestCleanup]`.

Emin Begić

- Implementacija data driven testova u `TaskServiceTests`: `SearchTasks_ByText_DataRow` i `SearchTasks_DynamicData_S` sa kolekcijom `SearchTestData`.
- Testovi za filtriranje i sortiranje zadataka (`SearchTasks_FilterByAssignedUser`, `SearchTasks_FilterByPriority`, `SearchTasks_OnlyNotOverdue_ShouldExcludeExpired`, `SearchTasks_SortByDueDateAsc`, `SearchTasks_SortByPrior` `SearchTasks_SortByCreatedDateDesc`).
- Provjera rubnih slučajeva (nema rezultata pretrage, zadaci van roka).

Aldin Velić

- Implementacija testova u `TaskRepositoryTests` za sve CRUD operacije nad `InMemoryTaskRepository`.
- Testovi izuzetaka u repozitoriju (`AddTask_NullTask_ShouldThrowException`, `AddTask_MissingPriority_ShouldThrow`
- Test `GetAllTasks_EmptyRepository_ShouldReturnEmptyList` koji pokriva slučaj praznog repozitorija.

Zana Beljuri

- Implementacija testova u `TaskAnalyzerTests` za metode `EvaluateTaskStatus`, `AnalyzeTeamPerformance`, `GetTasksSummaryForUser` i `PredictDelayRisk`.
- Implementacija testova sa zamjenskim objektima zasnovanih na `Moq` u `TaskServiceTests`, sa fokusom na provjeru interakcije servisa i repozitorija.
- Testiranje tekstualnih rezultata (*string* povratne vrijednosti) korištenjem `StringAssert` (`StringAssert.Contains`) gdje je primjenjivo.

7 Zaključak

Implementirani unit testovi zadovoljavaju zadane kriterije:

- Pokrivenost koda po klasama je iznad 90% za `TaskService`, `InMemoryTaskRepository` i `TaskAnalyzer`.
- Testirane su osnovne funkcionalnosti i izuzetci.
- Primijenjeno je data driven testiranje u više formi: `[DataRow]`, `[DynamicData]` sa kolekcijom ulaznih podataka te dodatno `[DynamicData]` sa eksternim XML i CSV datotekama.
- Ilustrirana je upotreba zamjenskih objekata (`Moq` mockovi) za izolaciju servisa od repozitorija.
- Testovi su jasno imenovani, dobro organizirani, bez nepotrebne logike i međusobno neovisni, uz korištenje `[TestInitialize]` i `[TestCleanup]` atributa.

Na ovaj način je ostvarena tražena razina kvaliteta unit testiranja u skladu sa zahtjevima zadatka (tačke 3.1 i 3.2).