**TESTONTWERP & -REALISATIE**

|  |  |
| --- | --- |
| **Naam** | Xue Kun Zhou |
| **Studentnummer** | 20170173 |
| **Klas** | 6 |

**OPDRACHT**

Geef hieronder een korte beschrijving van je opdracht (een samenvatting van je productvisie) zodat je docent OPT3 ook begrijpt met welke opdracht jij bezig bent.

MyLibrary is een webapp dat bijhoudt welke films, series, boeken en games je hebt gezien, gelezen en gespeeld. De gebruiker heeft daarmee een digitale “collectie” van media die hij heeft geconsumeerd. Per item wordt er ook een score bijgehouden zodat wanneer de gebruiker op een later moment kan terugzien wat hij ervan vond. Door het bijhouden van deze items kan de gebruiker ook een overzicht krijgen van hoeveel tijd hij hierin besteed.

De app houdt niet bij welke films etc. er al bestaan en houdt dus geen gemiddelde score bij van alle gebruikers en geeft ook geen aanbevelingen van welke items de gebruiker misschien wel leuk vindt.

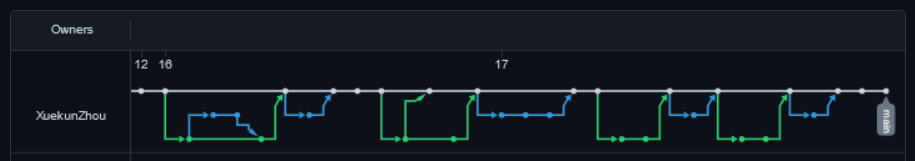
Vrienden kunnen wel de lijsten van elkaar zien.

**GITHUB EN TDD**

Maak je GitHub-repo Private en geef je docent toegang tot je repo:

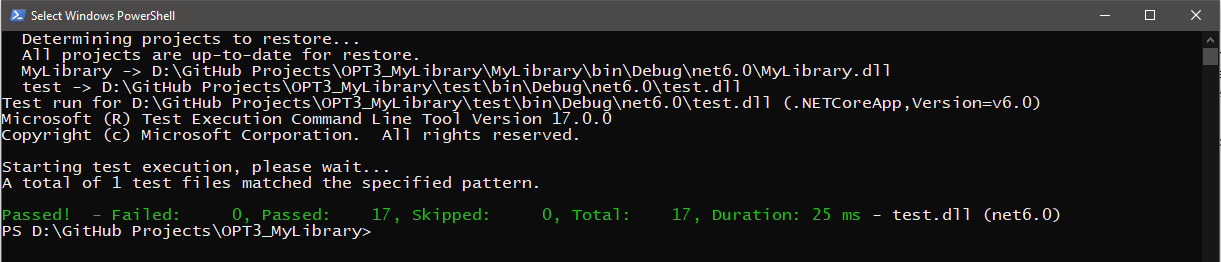
https://github.com/XuekunZhou/OPT3\_MyLibrary

Om aan te tonen dat je volgens TDD hebt gewerkt voor deze opdracht neem je hieronder een kopie van je netwerk over (zie de toelichting op het template voor instructie):



**GESLAAGDE TESTS**

Toon met een relevante screenshot aan dat alle onderstaande tests inderdaad zijn geslaagd:



**SAMENGESTELDE DECISION**

In mijn software neem ik een besluit op basis van de volgende voorwaarden (maximaal 3 te testen):

Voorwaarde A: Gebruikers lijst is openbaar

Voorwaarde B: Ingelogde gebruiker is vrienden met gebruiker2

Voorwaarde C: Gebruiker is ingelogd

Het besluit is gebaseerd op de volgende combinatie van deze voorwaarden (bijv. D = A && (B || C)):

D = (A || B) && C

**CONDITION COVERAGE**

Als ik mijn software voor deze Decision zou testen, dan zou ik met Condition Coverage de volgende test cases testen (als er teveel regels in de tabel staan, kun je de waarden vervangen door een spatie):

|  |  |  |  |
| --- | --- | --- | --- |
| **Voorwaarde A** | **Voorwaarde B** | **Voorwaarde C** | **D = (A || B) && C** |
| True | True | True | True |
| False | False | False | False |
| <waarde> | <waarde> | <waarde> | <waarde> |
| <waarde> | <waarde> | <waarde> | <waarde> |
| <waarde> | <waarde> | <waarde> | <waarde> |
| <waarde> | <waarde> | <waarde> | <waarde> |
| <waarde> | <waarde> | <waarde> | <waarde> |
| <waarde> | <waarde> | <waarde> | <waarde> |
| <waarde> | <waarde> | <waarde> | <waarde> |

**DECISION COVERAGE**

Als ik mijn software voor deze Decision zou testen, dan zou ik met Decision Coverage de volgende test cases testen (als er teveel regels in de tabel staan, kun je de waarden vervangen door een spatie):

|  |  |  |  |
| --- | --- | --- | --- |
| **Voorwaarde A** | **Voorwaarde B** | **Voorwaarde C** | **D = (A || B) && C** |
| True | True | True | True |
| False | False | False | False |
| <waarde> | <waarde> | <waarde> | <waarde> |
| <waarde> | <waarde> | <waarde> | <waarde> |
| <waarde> | <waarde> | <waarde> | <waarde> |
| <waarde> | <waarde> | <waarde> | <waarde> |
| <waarde> | <waarde> | <waarde> | <waarde> |
| <waarde> | <waarde> | <waarde> | <waarde> |
| <waarde> | <waarde> | <waarde> | <waarde> |

**CONDITION/DECISION COVERAGE**

Als ik mijn software voor deze Decision zou testen, dan zou ik met Condition/Decision Coverage de volgende test cases testen (als er teveel regels in de tabel staan, kun je de waarden vervangen door een spatie):

|  |  |  |  |
| --- | --- | --- | --- |
| **Voorwaarde A** | **Voorwaarde B** | **Voorwaarde C** | **D = (A || B) && C** |
| True | True | True | True |
| False | False | False | False |
| <waarde> | <waarde> | <waarde> | <waarde> |
| <waarde> | <waarde> | <waarde> | <waarde> |
| <waarde> | <waarde> | <waarde> | <waarde> |
| <waarde> | <waarde> | <waarde> | <waarde> |
| <waarde> | <waarde> | <waarde> | <waarde> |
| <waarde> | <waarde> | <waarde> | <waarde> |
| <waarde> | <waarde> | <waarde> | <waarde> |

**MODIFIED CONDITION/DECISION COVERAGE**

Als ik mijn software voor deze Decision zou testen, dan zou ik met Modified Condition/Decision Coverage de volgende test cases testen (als er teveel regels in de tabel staan, kun je de waarden vervangen door een spatie). Geef met <l>/<f> aan met welke logische en fysieke test cases je werkt:

|  |  |  |  |
| --- | --- | --- | --- |
| **Voorwaarde A** | **Voorwaarde B** | **Voorwaarde C** | **D = (A || B) && C** |
| False | False | True | False |
| False | True | True | True |
| False | True | False | False |
| True | False | True | True |
| <waarde> | <waarde> | <waarde> | <waarde> |
| <waarde> | <waarde> | <waarde> | <waarde> |
| <waarde> | <waarde> | <waarde> | <waarde> |
| <waarde> | <waarde> | <waarde> | <waarde> |
| <waarde> | <waarde> | <waarde> | <waarde> |

**MULTIPLE CONDITION COVERAGE**

Als ik mijn software voor deze Decision zou testen, dan zou ik met Multiple Condition Coverage de volgende test cases testen (als er teveel regels in de tabel staan, kun je de waarden vervangen door een spatie):

|  |  |  |  |
| --- | --- | --- | --- |
| **Voorwaarde A** | **Voorwaarde B** | **Voorwaarde C** | **D = (A || B) && C** |
| True | True | True | True |
| True | True | False | False |
| True | False | True | True |
| True | False | False | False |
| False | True | True | True |
| False | True | False | False |
| False | False | True | False |
| False | False | False | False |
| <waarde> | <waarde> | <waarde> | <waarde> |

**MODIFIED CONDITION/DECISION COVERAGE (JUNIT)**

Ik heb de testcases voor Modified Condition/Decision Coverage omgezet naar de volgende JUnit-code om mijn software voor deze beslissing te testen.

#nullable disable

using System.Collections.Generic;

using System.Linq;

using System.Security.Claims;

using Microsoft.AspNetCore.Identity;

using Microsoft.AspNetCore.Mvc;

using Microsoft.EntityFrameworkCore;

using MyLibrary.Models;

using Moq;

using Xunit;

using MyLibrary.Controllers;

using System.Threading.Tasks;

namespace test;

public class TestFilmController

{

    // A: list is public

    // B: UserOne is friends with UserTwo

    // C: UserOne is logged in

    private ApplicationDbContext context;

    private ApplicationUser alice;

    private ApplicationUser bob;

    private ApplicationUser carol;

    private void Setup()

    {

        var options = new DbContextOptionsBuilder<ApplicationDbContext>().UseInMemoryDatabase("TestFilmController").Options;

        context = new ApplicationDbContext(options);

        context.Database.EnsureCreated();

        alice = new ApplicationUser();

        alice.Id = "111";

        bob = new ApplicationUser();

        bob.Id = "222";

        bob.listsArePublic = true;

        carol = new ApplicationUser();

        carol.Id = "333";

        var friend = new Friend();

        friend.UserOne = alice;

        friend.UserTwo = bob;

        context.Add(alice);

        context.Add(bob);

        context.Add(carol);

        context.Add(friend);

        context.SaveChanges();

    }

    private void Dispose()

    {

        context.Database.EnsureDeleted();

    }

    private Mock<UserManager<ApplicationUser>> GetMockUserManager(ApplicationUser loggedInUser, ApplicationUser wachtedUser)

    {

        var mgr = new Mock<UserManager<ApplicationUser>>(Mock.Of<IUserStore<ApplicationUser>>(), null, null, null, null, null, null, null, null);;

        mgr.Setup(x => x.GetUserAsync(It.IsAny<ClaimsPrincipal>())).ReturnsAsync(loggedInUser);

        mgr.Setup(x => x.FindByIdAsync(It.IsAny<string>())).ReturnsAsync(wachtedUser);

        return mgr;

    }

    [Fact]

    public async Task Test\_NotA\_NotB\_C\_Results\_FalseAsync()

    {

        // Given

        Setup();

        var mgr = GetMockUserManager(alice, carol);

        var sut = new FilmController(context, mgr.Object);

        // When

        var res = await sut.ListAsync(carol.Id);

        var viewResult = Assert.IsType<ViewResult>(res);

        // Then

        Assert.Equal("Private", viewResult.ViewName);

        Dispose();

    }

    [Fact]

    public async Task Test\_NotA\_B\_C\_Results\_TrueAsync()

    {

        // Given

        Setup();

        var mgr = GetMockUserManager(alice, bob);

        var sut = new FilmController(context, mgr.Object);

        // When

        var res = await sut.ListAsync(bob.Id);

        var viewResult = Assert.IsType<ViewResult>(res);

        // Then

        Assert.Equal("List", viewResult.ViewName);

        Dispose();

    }

    [Fact]

    public async Task Test\_NotA\_B\_NotC\_Results\_FalseAsync()

    {

        // Given

        Setup();

        var mgr = GetMockUserManager(null, carol);

        var sut = new FilmController(context, mgr.Object);

        // When

        var res = await sut.ListAsync(carol.Id);

        var viewResult = Assert.IsType<RedirectToActionResult>(res);

        // Then

        Assert.Equal("Login", viewResult.ActionName);

        Dispose();

    }

    [Fact]

    public async Task Test\_A\_NotB\_C\_Results\_TrueAsync()

    {

        // Given

        Setup();

        var mgr = GetMockUserManager(carol, bob);

        var sut = new FilmController(context, mgr.Object);

        // When

        var res = await sut.ListAsync(bob.Id);

        var viewResult = Assert.IsType<ViewResult>(res);

        // Then

        Assert.Equal("List", viewResult.ViewName);

        Dispose();

    }

}

Hiermee test ik de volgende methode (van het type boolean, waarin de waarde van D = <Decision> wordt bepaald):

public async Task<IActionResult> ListAsync(string? id)

        {

            var loggedInUser = await \_userManager.GetUserAsync(User);

            var watchedUser = await \_userManager.FindByIdAsync(id);

            var films = \_context.FilmEntries.Where(u => u.User.Id == id).ToList();

            if (loggedInUser == null)

            {

                return RedirectToAction("Login", "Account");

            }

            if ((id != null) && (watchedUser.listsArePublic || loggedInUser.IsFriendsWith(watchedUser)))

            {

                ViewData["Title"] = watchedUser.UserName + "'s list";

                return View("List", films);

            }

            return View("Private");

        }

**EQUIVALENTIEKLASSEN EN RANDWAARDEN**

Voor het nemen van een besluit in de software heb ik een domein dat bestaat uit de volgende equivalentieklassen (maximaal 3 te testen):

|  |  |  |
| --- | --- | --- |
|  | Logische weergave | Besluit |
| Equivalentieklasse 1 | T < 840 | Geen waarschuwing |
| Equivalentieklasse 2 | T >= 840 | Waarschuwing 1 |
| Equivalentieklasse 3 | T >= 1260 | Waarschuwing 2 |

Geef nu alle relevante fysieke testgevallen (incl. de meest intensieve test van randwaarden) en geef voor elk testgeval aan voor welke equivalentieklasse dit een test is (als je de laatste regels niet meer nodig hebt, kun je deze velden vullen met een spatie):

|  |  |  |
| --- | --- | --- |
| Equivalentieklasse | Fysiek testgeval | Besluit |
| 1 | 0 | Geen waarschuwing |
| 1 | 839 | Geen waarschuwing |
| 2 | 840 | Waarschuwing 1 |
| 2 | 1259 | Waarschuwing 1 |
| 3 | 1260 | Waarschuwing 2 |
| 3 | 1000000 | Waarschuwing 2 |
| <Equivalentieklasse> | <Testgeval> | <Besluit> |
| <Equivalentieklasse> | <Testgeval> | <Besluit> |
| <Equivalentieklasse> | <Testgeval> | <Besluit> |
| <Equivalentieklasse> | <Testgeval> | <Besluit> |
| <Equivalentieklasse> | <Testgeval> | <Besluit> |
| <Equivalentieklasse> | <Testgeval> | <Besluit> |
| <Equivalentieklasse> | <Testgeval> | <Besluit> |
| <Equivalentieklasse> | <Testgeval> | <Besluit> |
| <Equivalentieklasse> | <Testgeval> | <Besluit> |
| <Equivalentieklasse> | <Testgeval> | <Besluit> |
| <Equivalentieklasse> | <Testgeval> | <Besluit> |
| <Equivalentieklasse> | <Testgeval> | <Besluit> |
| <Equivalentieklasse> | <Testgeval> | <Besluit> |
| <Equivalentieklasse> | <Testgeval> | <Besluit> |
| <Equivalentieklasse> | <Testgeval> | <Besluit> |

**EQUIVALENTIEKLASEN EN RANDWAARDEN (JUNIT)**

Ik heb de testcases voor Equivalentieklassen en randwaarden omgezet naar de volgende JUnit-code om mijn software voor deze beslissing te testen.

#nullable disable

using System;

using Microsoft.EntityFrameworkCore;

using MyLibrary.Models;

using Xunit;

namespace test

{

    public class TestOverviewViewModel

    {

        // Equivalance testing

        private ApplicationDbContext context;

        private ApplicationUser alice;

        private void Setup()

        {

            var options = new DbContextOptionsBuilder<ApplicationDbContext>().UseInMemoryDatabase("TestOverviewViewModel").Options;

            context = new ApplicationDbContext(options);

            context.Database.EnsureCreated();

            alice = new ApplicationUser();

            context.Add(alice);

            context.SaveChanges();

        }

        private void Dispose()

        {

            context.Database.EnsureDeleted();

        }

        [Fact]

        public void EquivalenceClassOneLowerEdge()

        {

            // Given

           Setup();

            // When

            var sut = new OverviewViewModel(context, alice, 7);

            // Then

            Assert.Equal(0, sut.TimeSpentOnFilmsInMinutes);

            Dispose();

        }

        [Fact]

        public void EquivalenceClassOneUpperEdge()

        {

            // Given

            Setup();

            context.Add(new FilmEntryModel{Title = "a", LengthInMinutes = 100, DateOfEntry = DateTime.Now, User = alice});

            context.Add(new FilmEntryModel{Title = "b", LengthInMinutes = 100, DateOfEntry = DateTime.Now, User = alice});

            context.Add(new FilmEntryModel{Title = "c", LengthInMinutes = 639, DateOfEntry = DateTime.Now, User = alice});

            context.SaveChanges();

            // When

            var sut = new OverviewViewModel(context, alice, 7);

            // Then

            Assert.Equal(839, sut.TimeSpentOnFilmsInMinutes);

            Dispose();

        }

        [Fact]

        public void EquivalenceClassTwoLowerEdge()

        {

            // Given

            Setup();

            context.Add(new FilmEntryModel{Title = "a", LengthInMinutes = 100, DateOfEntry = DateTime.Now, User = alice});

            context.Add(new FilmEntryModel{Title = "b", LengthInMinutes = 200, DateOfEntry = DateTime.Now, User = alice});

            context.Add(new FilmEntryModel{Title = "c", LengthInMinutes = 540, DateOfEntry = DateTime.Now, User = alice});

            context.SaveChanges();

            // When

            var sut = new OverviewViewModel(context, alice, 7);

            // Then

            Assert.Equal(840, sut.TimeSpentOnFilmsInMinutes);

            Dispose();

        }

        [Fact]

        public void EquivalenceClassTwoUpperEdge()

        {

            // Given

            Setup();

            context.Add(new FilmEntryModel{Title = "a", LengthInMinutes = 100, DateOfEntry = DateTime.Now, User = alice});

            context.Add(new FilmEntryModel{Title = "b", LengthInMinutes = 100, DateOfEntry = DateTime.Now, User = alice});

            context.Add(new FilmEntryModel{Title = "c", LengthInMinutes = 659, DateOfEntry = DateTime.Now, User = alice});

            context.Add(new FilmEntryModel{Title = "d", LengthInMinutes = 400, DateOfEntry = DateTime.Now, User = alice});

            context.SaveChanges();

            // When

            var sut = new OverviewViewModel(context, alice, 7);

            // Then

            Assert.Equal(1259, sut.TimeSpentOnFilmsInMinutes);

            Dispose();

        }

        [Fact]

        public void EquivalenceClassThreeLowerEdge()

        {

            // Given

            Setup();

            context.Add(new FilmEntryModel{Title = "a", LengthInMinutes = 100, DateOfEntry = DateTime.Now, User = alice});

            context.Add(new FilmEntryModel{Title = "b", LengthInMinutes = 100, DateOfEntry = DateTime.Now, User = alice});

            context.Add(new FilmEntryModel{Title = "c", LengthInMinutes = 660, DateOfEntry = DateTime.Now, User = alice});

            context.Add(new FilmEntryModel{Title = "d", LengthInMinutes = 400, DateOfEntry = DateTime.Now, User = alice});

            context.SaveChanges();

            // When

            var sut = new OverviewViewModel(context, alice, 7);

            // Then

            Assert.Equal(1260, sut.TimeSpentOnFilmsInMinutes);

            Dispose();

        }

        [Fact]

        public void EquivalenceClassThree()

        {

            // Given

            Setup();

            context.Add(new FilmEntryModel{Title = "a", LengthInMinutes = 1000000, DateOfEntry = DateTime.Now, User = alice});

            context.SaveChanges();

            // When

            var sut = new OverviewViewModel(context, alice, 7);

            // Then

            Assert.Equal(1000000, sut.TimeSpentOnFilmsInMinutes);

            Dispose();

        }

        [Fact]

        public void TestOldEntries()

        {

            // Given

            Setup();

            context.Add(new FilmEntryModel{Title = "a", LengthInMinutes = 100, DateOfEntry = DateTime.Now, User = alice});

            context.Add(new FilmEntryModel{Title = "b", LengthInMinutes = 100, DateOfEntry = DateTime.Now, User = alice});

            context.Add(new FilmEntryModel{Title = "c", LengthInMinutes = 100, DateOfEntry = DateTime.Now.AddDays(-8), User = alice});

            context.SaveChanges();

            // When

            var sut = new OverviewViewModel(context, alice, 7);

            // Then

            Assert.Equal(200, sut.TimeSpentOnFilmsInMinutes);

            Dispose();

        }

    }

}

Hiermee test ik de volgende methode (waarin de besluiten op basis van de equivalentieklassen worden bepaald):

private void GetTimeSpentOnFilmsInMinutes()

        {

            var date = DateTime.Now.AddDays(-TimePeriodInDays);

            var totalTime = \_context.FilmEntries.Where(u => u.User == \_user).Where(f => f.DateOfEntry >= date).ToList().Sum(x => x.LengthInMinutes);

            TimeSpentOnFilmsInMinutes = totalTime;

        }

**PAIRWISE TESTING**

Voor Pairwise Testing maak ik gebruik van de volgende parameters (op basis waarvan een besluit wordt genomen; vereenvoudig tot een set van 4 parameters die ieder 3, 2, 2 en 2 mogelijkheden hebben (anders wordt handmatig testen met Pairwise Testing veel te ingewikkeld; vul een spatie in, als je een optie niet wilt gebruiken):

|  |  |  |  |
| --- | --- | --- | --- |
| **Parameter** | **Optie 1** | **Optie 2** | **Optie 3** |
| Ingelogd | True | False | <Optie 3> |
| < 7 dagen | True | False | <Optie 3> |
| T | T < 840 | T >= 840 | T >= 1260 |
| <Parameter 4> | <Optie 1> | <Optie 2> | <Optie 3> |

Na de tussenstappen in het proces heb ik de volgende testtabel met fysieke testgevallen opgesteld (vul spaties in, als er teveel testgevallen in de tabel staan voor jouw casus):

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **T** | **<7 dagen** | **Ingelogd** | **<Parameter** 4> | **Decision** |
| 120 | True | True | <waarde> | Geen waarschuwing |
| 120 | False | False | <waarde> | Geen waarschuwing |
| <waarde> | <waarde> | <waarde> | <waarde> | <waarde> |
| 850 | True | False | <waarde> | Geen waarschuwing |
| 850 | False | True | <waarde> | Geen waarschuwing |
| <waarde> | <waarde> | <waarde> | <waarde> | <waarde> |
| 1320 | True | True | <waarde> | Waarschuwing 2 |
| 1320 | False | False | <waarde> | Geen waarschuwing |
| <waarde> | <waarde> | <waarde> | <waarde> | <waarde> |
| <waarde> | <waarde> | <waarde> | <waarde> | <waarde> |
| <waarde> | <waarde> | <waarde> | <waarde> | <waarde> |
| <waarde> | <waarde> | <waarde> | <waarde> | <waarde> |
| <waarde> | <waarde> | <waarde> | <waarde> | <waarde> |

**PAIRWISE TESTING (JUNIT)**

Ik heb de testcases voor Pairwise Testing omgezet naar de volgende JUnit-code om mijn software voor deze beslissing te testen.

#nullable disable

using System;

using System.Security.Claims;

using System.Threading.Tasks;

using Microsoft.AspNetCore.Identity;

using Microsoft.AspNetCore.Mvc;

using Microsoft.EntityFrameworkCore;

using Moq;

using MyLibrary.Controllers;

using MyLibrary.Models;

using Xunit;

namespace test

{

    public class TestHomeController

    {

        // Pairwise testing

        // A: T = 120

        // B: T = 850

        // C: T = 1320

        // X: Entry is younger than 7 days

        // Y: Entry is older than 7 days (so its t gets subtracted from T)

        // P: User is logged in

        // Q: User is not logged in

        private ApplicationDbContext context;

        private ApplicationUser alice;

        private void Setup()

        {

            var options = new DbContextOptionsBuilder<ApplicationDbContext>().UseInMemoryDatabase("TestHomeController").Options;

            context = new ApplicationDbContext(options);

            context.Database.EnsureCreated();

            alice = new ApplicationUser();

            context.Add(alice);

            context.SaveChanges();

        }

        private void Dispose()

        {

            context.Database.EnsureDeleted();

        }

        private Mock<UserManager<ApplicationUser>> GetMockUserManager(ApplicationUser loggedInUser)

        {

            var mgr = new Mock<UserManager<ApplicationUser>>(Mock.Of<IUserStore<ApplicationUser>>(), null, null, null, null, null, null, null, null);;

            mgr.Setup(x => x.GetUserAsync(It.IsAny<ClaimsPrincipal>())).ReturnsAsync(loggedInUser);

            return mgr;

        }

        [Fact]

        public async Task A\_X\_PAsync()

        {

            // Given

            Setup();

            var mgr = GetMockUserManager(alice);

            var sut = new HomeController(context, mgr.Object);

            context.Add(new FilmEntryModel{Title = "a", LengthInMinutes = 120, DateOfEntry = DateTime.Now, User = alice});

            context.SaveChanges();

            // When

            var res = await sut.IndexAsync();

            var viewRes = Assert.IsType<ViewResult>(res);

            // Then

            Assert.Equal("", viewRes.ViewData["Warning"]);

            Dispose();

        }

        [Fact]

        public async Task A\_Y\_QAsync()

        {

            // Given

            Setup();

            var mgr = GetMockUserManager(null);

            var sut = new HomeController(context, mgr.Object);

            context.Add(new FilmEntryModel{Title = "a", LengthInMinutes = 120, DateOfEntry = DateTime.Now.AddDays(-8), User = alice});

            context.SaveChanges();

            // When

            var res = await sut.IndexAsync();

            var viewRes = Assert.IsType<ViewResult>(res);

            // Then

            Assert.Equal("", viewRes.ViewData["Warning"]);

            Dispose();

        }

        [Fact]

        public async Task B\_X\_QAsync()

        {

            // Given

            Setup();

            var mgr = GetMockUserManager(null);

            var sut = new HomeController(context, mgr.Object);

            context.Add(new FilmEntryModel{Title = "a", LengthInMinutes = 850, DateOfEntry = DateTime.Now, User = alice});

            context.SaveChanges();

            // When

            var res = await sut.IndexAsync();

            var viewRes = Assert.IsType<ViewResult>(res);

            // Then

            Assert.Equal("", viewRes.ViewData["Warning"]);

            Dispose();

        }

        [Fact]

        public async Task B\_Y\_PAsync()

        {

            // Given

            Setup();

            var mgr = GetMockUserManager(alice);

            var sut = new HomeController(context, mgr.Object);

            context.Add(new FilmEntryModel{Title = "a", LengthInMinutes = 850, DateOfEntry = DateTime.Now.AddDays(-8), User = alice});

            context.SaveChanges();

            // When

            var res = await sut.IndexAsync();

            var viewRes = Assert.IsType<ViewResult>(res);

            // Then

            Assert.Equal("", viewRes.ViewData["Warning"]);

            Dispose();

        }

        [Fact]

        public async Task C\_X\_PAsync()

        {

            // Given

            Setup();

            var mgr = GetMockUserManager(alice);

            var sut = new HomeController(context, mgr.Object);

            context.Add(new FilmEntryModel{Title = "a", LengthInMinutes = 1320, DateOfEntry = DateTime.Now, User = alice});

            context.SaveChanges();

            // When

            var res = await sut.IndexAsync();

            var viewRes = Assert.IsType<ViewResult>(res);

            // Then

            Assert.Equal("You REALLY should spent less time on this", viewRes.ViewData["Warning"]);

            Dispose();

        }

        [Fact]

        public async Task C\_Y\_QAsync()

        {

            // Given

            Setup();

            var mgr = GetMockUserManager(null);

            var sut = new HomeController(context, mgr.Object);

            context.Add(new FilmEntryModel{Title = "a", LengthInMinutes = 1320, DateOfEntry = DateTime.Now.AddDays(-8), User = alice});

            context.SaveChanges();

            // When

            var res = await sut.IndexAsync();

            var viewRes = Assert.IsType<ViewResult>(res);

            // Then

            Assert.Equal("", viewRes.ViewData["Warning"]);

            Dispose();

        }

    }

}

Hiermee test ik de volgende methode (waarin het besluit op basis van de meegegeven parameters wordt bepaald):

public async Task<IActionResult> IndexAsync()

        {

            var loggedInUser = await \_userManager.GetUserAsync(User);

            OverviewViewModel? model = null;

            ViewData["Warning"] = "";

            if (loggedInUser != null)

            {

                model = new OverviewViewModel(\_context, loggedInUser, 7);

                if (model.TimeSpentOnFilmsInMinutes >= 1320)

                {

                    ViewData["Warning"] = "You REALLY should spent less time on this";

                }

                else if (model.TimeSpentOnFilmsInMinutes >= 850)

                {

                    ViewData["Warning"] = "You should spent less time on this";

                }

            }

            return View(model);

        }