



# MVC med Javascript och Ajax

Filip Ekberg

# Hur går det?





# MVC med Javascript och Ajax

Lektion 6 – Inloggning, Testning & Separation of Concerns

[mail@filipekberg.se](mailto:mail@filipekberg.se)



# Dagens mål

- Introducera inloggning, testning och bra kodstruktur
  - Hur läggere jag till inloggning?
  - Hur testar man ASP.NET MVC kod
  - Hur testar jag en Action, Controller och Modell
  - Hur testar jag mitt service, business eller datalager
  - Vad är enhetstester, automattester, UI-tester och andra typer av tester
  - Hur skriver jag tester utan att gå mot databasen
  - Dependency Injection & Inversion of Control
  - Mock & Fake
- Repetition
  - Hur introducerar vi EntityFramework
  - Hur använder vi Repository Pattern
  - Hur utyttjar vi AjaxExtensions

# Inloggning med ASP.NET Identity

# Hur bygger man en säker inloggning?



# Varför Hasha och inte Kryptera?



# Vad är 2FA?

# Authentication

Inloggning

# Authorization

Behörighet

# Använder sig av OWIN

Som vi pratade om i lektion 1



# **ASP.NET Identity är Claims-baserat**

The **ClaimsIdentity** class is a concrete implementation of a **claims-based identity**; that is, an **identity** described by a collection of **claims**. A **claim** is a statement about an entity made by an issuer that describes a property, right, or some other quality of that entity.

**Ett Claim innehåller information  
om användaren**

**name : Filip Ekberg**  
**email : mail@filipekberg.se**



**Claims skickas fram och tillbaka  
till webbservern – med hjälp av  
cookies!**

**Kan man ändra på sina Claims  
då, i och med att Cookies är  
"osäkert"?**

**Din Cookie är signerad – det  
betyder att om du ändrar på  
den vet servern detta!**

**Tänk på HTTPS!**



# Roles vs Claims

**Stöd för OAuth, OpenId, AD,  
Office 365, Azure AD, med mera**

# ASP.NET Authentication with Identity

<https://channel9.msdn.com/Series/Customizing-ASPNET-Authentication-with-Identity/01>

```
Install-Package Microsoft.Owin.Host.SystemWeb
```

```
Install-Package Microsoft.AspNet.Identity.OWIN
```

```
Install-Package Microsoft.AspNet.Identity.EntityFramework
```



UserManager<IdentityUser>


```
public class MyIdentityDbContext
    : IdentityDbContext<IdentityUser>
{ }
```



```
var context = new MyIdentityDbContext();
var store = new UserStore<IdentityUser>(context);

userManager = new UserManager<IdentityUser>(store);
```

Hantera allt som har med  
användaren att göra




# Hur registrerar jag en användare?

```
var user = new IdentityUser
{
    UserName = username,
    Email = email
};
```

```
var result = await userManager.CreateAsync(user, password);
```

```
if (result.Succeeded)
{
}
```



Vi vill verifiera att det gick att skapa användaren – Vi kanske har specialvalidering av längd på lösenord

Skapa användaren med ett anviget lösenord

# IdentityUser

✚ AccessFailedCount	0
▶ ✚ Claims	Count = 0
✚ Email	🔍 ▾ "mail@filipekberg.se"
✚ EmailConfirmed	false
✚ Id	🔍 ▾ "fab47a11-991e-4826-be3d-c414873179ee"
✚ LockoutEnabled	false
✚ LockoutEndDateUtc	null
▶ ✚ Logins	Count = 0
✚ PasswordHash	🔍 ▾ "AK0/UobX9gMrCKxMKPQHfpDSK2TjbepCvZ33YdhJeVpZyNMFd6AxOWGE/QCiNo6soA= ="
✚ PhoneNumber	null
✚ PhoneNumberConfirmed	false
▶ ✚ Roles	Count = 0
✚ SecurityStamp	🔍 ▾ "70217fa1-2eab-4695-850c-cb6943c41fac"
✚ TwoFactorEnabled	false
✚ UserName	🔍 ▾ "asdfasdfsdfsd"

**Nu finns användaren –  
Hur loggar jag in?**



**Vi måste skapa en "Identity"**

User.Identity



ClaimsIdentity : IIdentity

När man är inloggad vet ASP.NET detta genom att kolla på en Cookie. Den sätter då User.Identity till en ClaimsPrincipal!

# Hur skapar man en Identity då?

```
var identity =  
    await userManager.CreateIdentityAsync(user,  
        DefaultAuthenticationTypes.ApplicationCookie);
```

**Nu måste vi säga till ASP.NET  
att vi ska loggas in med denna  
identiteten**

# IAuthenticationManager

```
var authenticationManager =  
    HttpContext.GetOwinContext().Authentication;
```

Vi använder OWIN för att kunna applicera ASP.NET Identity. Detta konfigurerar vi t.ex. i Startup.cs. Vi kan sen hämta ut detta context här och använda en AuthenticationManager.

**Nu vill vi sätta en Cookie för att  
säga att vi är inloggad!**



```
var properties = new AuthenticationProperties
{
    IsPersistent = false
};

authenticationManager.SignIn(properties, identity);
```

**Hur vet ASP.NET vilken Cookie  
den ska sätta, och ens att den  
ska kolla efter en Cookie?**

# Konfigureras i Startup.cs

```
app.UseCookieAuthentication(new CookieAuthenticationOptions
{
    AuthenticationType =
        DefaultAuthenticationTypes.ApplicationCookie,
    LoginPath = new PathString("/Home/Login")
});
```

**Hur kräver jag inloggning på en  
action?**

[Authorize]

# Hur skulle inloggningen se ut?

```
[AllowAnonymous]
[HttpPost]
[ValidateAntiForgeryToken]
public async Task<ActionResult> Login(string username,
    string password)
{
    var user = await userManager.FindAsync(username, password);

    if (user != null)
    {
        var identity = await userManager.CreateIdentityAsync(user,
            DefaultAuthenticationTypes.ApplicationCookie);

        var authenticationManager = HttpContext.GetOwinContext().Authentication;
        authenticationManager.SignIn(new AuthenticationProperties { IsPersistent = true },
            identity);

        return RedirectToAction("Index");
    }
    return View();
}
```

**Hur tillåter jag anonyma användare till en action?**



```
[AllowAnonymous]  
[HttpPost]  
[ValidateAntiForgeryToken]  
public async Task<ActionResult> Login(string username,  
    string password)  
{  
}
```

**Vad har Claims med detta att göra?**

**Claims innehåller t.ex. din email**

```
[Authorize]
public ActionResult Index()
{
    var identity = User.Identity as ClaimsIdentity;
    var email = identity.FindFirst("email").Value;
}
```

**Du kan utöka med egna!**

```
identity.AddClaim(new Claim("gender", "Male"));  
authenticationManager.SignIn(properties, identity);
```

```
[Authorize]
public ActionResult Index()
{
    var identity = User.Identity as ClaimsIdentity;
    var gender = identity.FindFirst("gender").Value;
}
```

**Detta verkar jättekompext..**



**Säkerhet är komplext, och svårt!**

# Testning

# Varför Testa vår kod?

**Räcker det inte att kompilera  
och köra vår applikation?**

# Dyrt i längden!

**Lager-strukturen i MVC gör det  
enklare att skriva tester för  
varje lager**

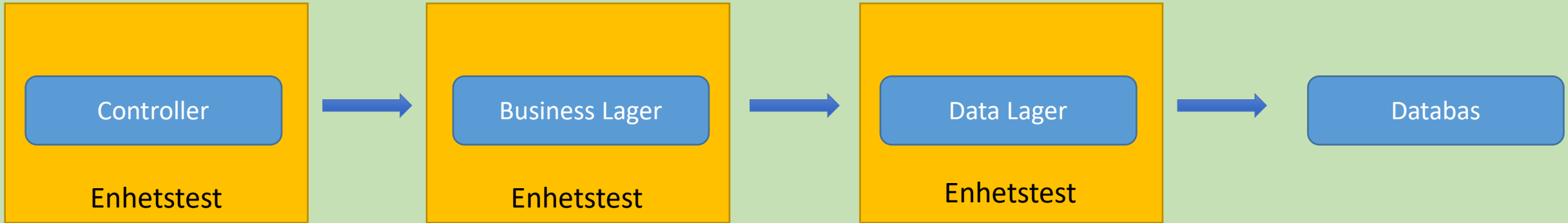
# Enhetstester (Unit Test)

Testa specifik funktionalitet – inga externa dependencies

# Integrationstester

Testa ett realistiskt flöde – går mot alla externa dependencies





Integrationstest

# Testbar kod

```
public class AddressBookRepository
{
    private static List<Entry> entries = new List<Entry>();

    public bool Add(Entry entry)
    {
        if (entries.Any(e => e.Name == entry.Name)) return false;

        entries.Add(entry);

        return true;
    }
}
```

# Hur skriver vi ett test för detta?

Add New Project?×

Recent

Installed

Visual C#

- Windows
- Web
- .NET Core
- Android
- Cloud
- Cross-Platform
- Extensibility
- iOS
- LightSwitch
- Office/SharePoint
- Silverlight
- Test
- Tizen
- tvOS
- WCF
- Workflow

Other Languages

Other Project Types









Modeling Projects

Online

.NET Framework 4.6.1

Sort by: Default

Search Installed Templates (Ctrl+E)

	Coded UI Test Project	Visual C#
	Unit Test Project	Visual C#
	Web Performance and Load Test Project	Visual C#
	UI Test App (Xamarin.UITest   Android)	Visual C#
	UI Test App (Xamarin.UITest   iOS)	Visual C#
	UI Test App (Xamarin.UITest   Cross-Platform)	Visual C#
	Unit Test App (Android)	Visual C#
	Unit Test App (iOS)	Visual C#

[Click here to go online and find templates.](#)

Type: Visual C#

A project that contains unit tests.

Name:MvcDemo.Tests

Location:C:\Users\Filip\Documents\Visual Studio 2015\Projects\MvcDemo

Browse...

OKCancel

# **Ett Test Projekt är bara ett Class Library**

# **Finns massor med olika Test Ramverk**

# MSTest



# NUnit

# XUnit

```
[TestClass]  
public class AddressBookTests  
{  
}
```

# En Test Class per område

# Tester definieras genom att skapa metoder

```
[TestMethod]  
public void Add_WithDuplicateInformation_ShouldReturnFalse()  
{  
}
```

**Långa, beskrivande namn är bra!**

# Arrange, Act, Assert



- 1) Skapa object som krävs innan vi kör koden som ska testas**
- 2) Kör koden som ska testas**
- 3) Verifiera att resultatet blev som väntat**

```
[TestClass]
public class AddressBookTests
{
    [TestMethod]
    public void Add_WithDuplicateInformation_ShouldReturnFalse()
    {
        // ARRANGE
        var entry = new Entry { Name = "Filip" };

        var addressBook = new AddressBookRepository();

        // ACT
        var couldAddEntryOnce = addressBook.Add(entry);


        var couldAddEntryTwice = addressBook.Add(entry);

        // ASSERT
        Assert.IsTrue(couldAddEntryOnce);
        Assert.IsFalse(couldAddEntryTwice);
    }
}
```

**Vad är skillnaden på att köra  
Action via ett test och via en  
webbläsare?**

# **Vad behöver vi veta om hur ASP.NET MVC fungerar internt?**


```
return View(photos);
```



```
protected internal virtual ViewResult View(string viewName,  
                                             string masterName, object model)  
{  
    if (model != null)  
        this.ViewData.Model = model;  
    ViewResult viewResult = new ViewResult();  
    viewResult.ViewName = viewName;  
    viewResult.MasterName = masterName;  
    viewResult.ViewData = this.ViewData;  
    viewResult.TempData = this.TempData;  
    viewResult.ViewEngineCollection = this.ViewEngineCollection;  
    return viewResult;  
}
```

```
var controller = new GalleryController();
```

```
var index = controller.Index() as ViewResult;
```



```
public ActionResult Index()  
{  
    ViewBag.ClientIp      = Request.UserHostAddress;  
    ViewBag.QueryString   = Request.QueryString;  
    ViewBag.RawUrl        = Request.RawUrl;  
  
    return View();  
}
```

# **Var sätts Request, Response, etc?**



# **ASP.NET hanterar det via sin Request/Response Life Cycle**

**Det sker alltså massor mer än att det bara skapas en instans av vår Controller**

**Finns det något vi kan göra för att testa delar av funktionaliteten?**

# **Integrationstest är en bättre lösning!**

**Kan jag inte bara sätta alla värden...**

```
[TestClass]
public class GalleryControllerTests
{
    [TestMethod]
    public void Index_ShouldReturn_IndexViewWithRequestDetailsInViewBag()
    {
        // ARRANGE
        var controller = new GalleryController();
        var httpRequest = new HttpRequest("", "http://localhost/Gallery/", "");
        var httpContext = new HttpContext(httpRequest, new HttpResponse(null));
        var requestContext = new RequestContext(new HttpContextWrapper(httpContext),
                                                new RouteData());

        controller.ControllerContext = new ControllerContext(requestContext, controller);

        // ACT
        var index = controller.Index() as ViewResult;

        // ASSERT
        Assert.IsNotNull(index);
        Assert.AreEqual("/Gallery/", index.ViewBag.RawUrl);
    }
}
```

**Får jag ut HTML nu?**

**NEJ!**

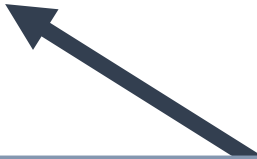
**Testar jag verkligen att rätt  
HTML returneras via ett  
enhetstest?**

**NEJ!**



# Varför inte?

```
var index = controller.Index() as ViewResult;  
index.ExecuteResult(controller.ControllerContext);
```



Det är först nu ASP.NET letar upp vår View, kompilerar Razor-koden, mappar värden och sedan skriver HTML till vår Response stream

Detta kommer dock inte fungera, det fattas fortfarande mycket som krävs, vilket är anledningen till att ett integrationstest är mer lämpligt.

# Mock & Fake

**Lämpligt när vi skriver tester!**

# Vad är ett Interface?

```
public ActionResult Index()
{
    var repository = new AddressRepository();
    repository.Find(x => x.Id == Guid.Empty);

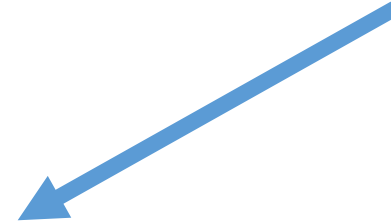
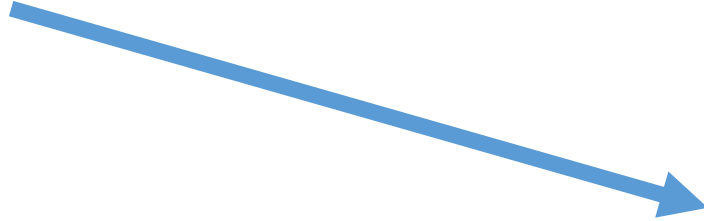
    return View(repository.All());
}
```

```
public ActionResult Index(IAddressRepository repository)
{
    repository.Find(x => x.Id == Guid.Empty);

    return View(repository.All());
}
```

`new FakeAddressRepository()`

`new AddressRepository()`



```
public ActionResult Index(IRepository repository)
{
    repository.Find(x => x.Id == Guid.Empty);

    return View(repository.All());
}
```

```
[TestClass]
public class AddressControllerTests
{
    [TestMethod]
    public void Index_SimpleCall_ShouldReturn_ViewDataWithAddresses()
    {
        var controller = new AddressController(null);

        ViewResult indexResult = controller.Index() as ViewResult;

        Assert.IsNotNull(indexResult.Model);
    }
}
```



**Skapa en "proxy" i runtime som aggerar  
som vi säger till den**

# NSubstitute

<http://nsubstitute.github.io/>

```
var repository = Substitute.For<IAddressRepository>();  
var controller = new AddressController(repository);
```

**Kan jag returnera fake data?**

```
var addresses = new[]  
{  
    new Address {StreetName = "From a test!"}  
};
```

```
var repository = Substitute.For<IAddressRepository>();  
repository.All().Returns(addresses);
```

Skicka in vår mockade  
IAddressRepository

Vår Action returnerar ett ViewResult

```
var controller = new AddressController(repository);
```

```
ViewResult indexResult = controller.Index() as ViewResult;
```

```
Assert.IsNotNull(indexResult.Model);
```

Som vi kan få ut en modell ifrån

```
var firstAddressInModel = ((IEnumerable<Address>) indexResult.Model).First();
```

```
Assert.AreEqual("From a test!", firstAddressInModel.StreetName);
```

Första Adressen i modellen är samma som vi skickade in genom att säga vad vår mockade klass skulle returnera när vi kallar på All!

# Dependency Injection & Inversion of Control

# Separation of Concerns



**“Löst Kopplat”**

**Hur lätt är det att skriva ett test nu som  
inte går mot databasen?**

# Dependency Injection

**Automatiskt skapa instanser av våra  
klasser och skicka in de i konstruktorn**

# Ramverk för att göra detta åt oss i ASP.NET MVC

# Autofac

<https://autofac.org/>

# Install-Package Autofac.Mvc5

<http://docs.autofac.org/en/latest/integration/mvc.html>

```
public class MvcApplication : System.Web.HttpApplication
{
    protected void Application_Start()
    {
        SetupContainer();

        AreaRegistration.RegisterAllAreas();
        RouteConfig.RegisterRoutes(RouteTable.Routes);
    }
}
```



```
private void SetupContainer()
{
    var builder = new ContainerBuilder();
    builder.RegisterControllers(typeof(MvcApplication).Assembly);

    builder
        .RegisterType<AddressRepository>()
        .As<IAddressRepository>()
        .InstancePerLifetimeScope();

    var container = builder.Build();
    DependencyResolver.SetResolver(
        new AutofacDependencyResolver(container));
}
```

# Starta en ny konfiguration av hur vi mappar t.ex Interface till Klasser

```
var builder = new ContainerBuilder();
```

# Registrera alla Controllers i vårt assembly (MVC Projektet)

```
builder.RegisterControllers(typeof(MvcApplication).Assembly);
```

# Manuellt registrera typ-mappning mot interface

builder

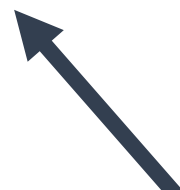
```
.RegisterType<AddressRepository>()  
.As<IAddressRepository>()  
.InstancePerLifetimeScope();
```

# Bygg konfigurationen

```
var container = builder.Build();
```

# Vad gör en Container?

```
var repository = container.Resolve<IAddressRepository>()
```



Gör i princip `new AddressRepository()`

# Varför är det bra då?

```
var controller = container.Resolve<AddressController>()
```

Containern fattar att den ska automatiskt skjuta in IAddressRepository och skapar därför intern en instans av det objektet!

```
public AddressController(IAddressRepository repository)
{
    this.repository = repository;
}
```

# Hur får vi MVC att använda vår container?

```
var container = builder.Build();
```

```
DependencyResolver.SetResolver(  
    new AutofacDependencyResolver(container));
```



# Sätt en break-point i konstruktorn!

0 references

```
public AddressController(IAddressRepository repository)
{
    this.repository = repository;
}
```

repository {MvcMedEntityFramework.Data.Repositories.AddressRepository}



# Integrationstester & Automatiska UI Tester

# Hur funkar ett integrationstest?

**Gör en riktig Request via en browser:  
Automatiskt!**

# Selenium

<http://www.seleniumhq.org/>

# Körs som vanliga Test i Visual Studio

Install-Package Selenium.WebDriver




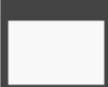



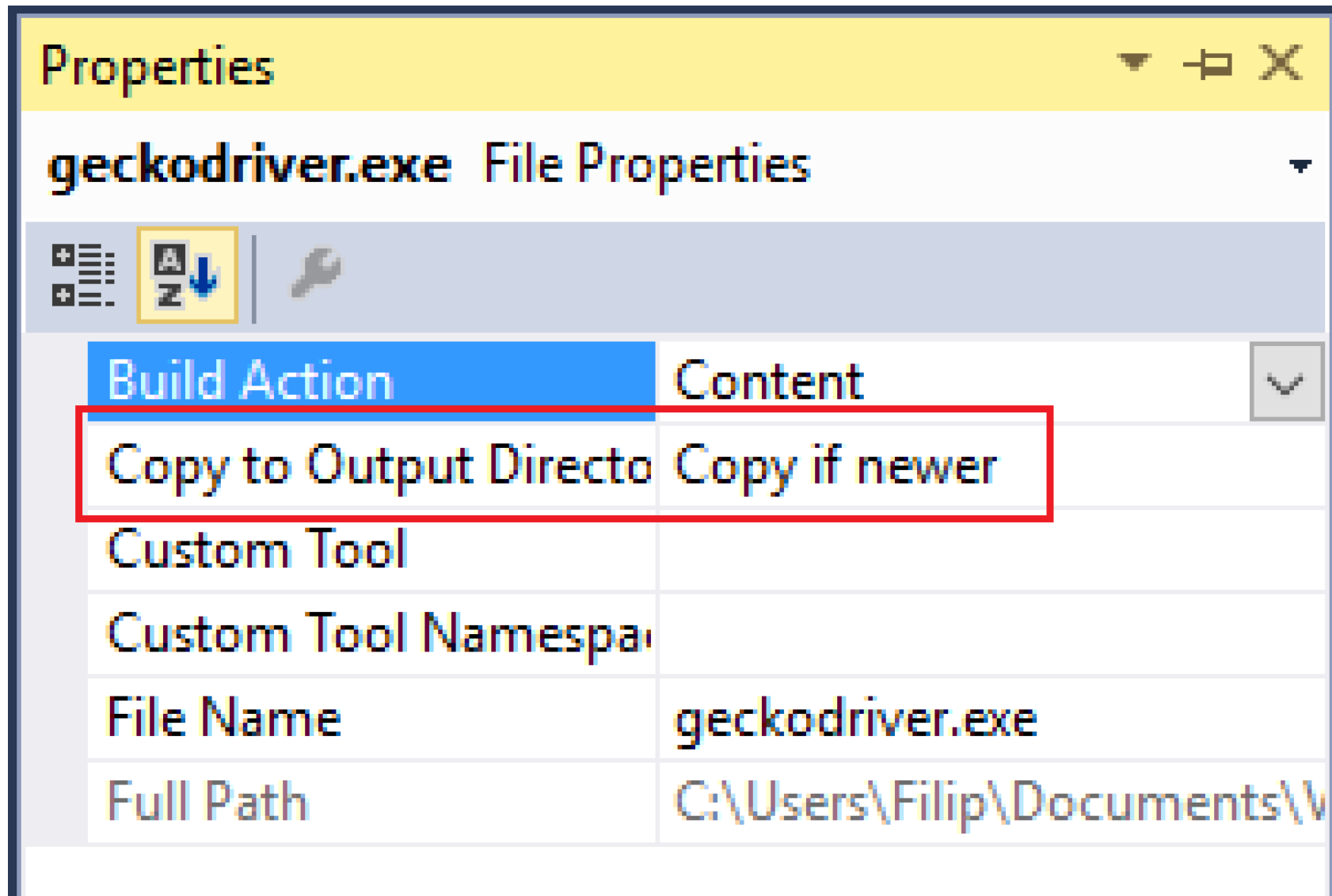
# Ladda ner Firefox

# Ladda ner GeckoDriver

<https://github.com/mozilla/geckodriver/releases>

## MvcDemo.EntityFramework.Tests

- ▶  Properties
- ▶  References
- ▶  GalleryControllerTests.cs
-  geckodriver.exe
-  packages.config




```
var firefox = new FirefoxDriver();
```

```
[TestClass]
public class GalleryControllerTests
{
    [TestMethod]
    public void Index_ShouldReturn_IndexViewWithListOfImages()
    {
    }
}
```

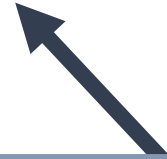
```
// ARRANGE
```

```
var firefox = new FirefoxDriver();  
firefox.Navigate().GoToUrl("http://localhost:58948/");
```

```
RemoveOldPhotos(firefox);
```



Vid integrationstester vill vi se till  
att vi jobbar med ett "cleant" state



Vår ASP.NET MVC webbsida –  
Detta kräver att vi startat  
projektet!

```
// ACT
```

```
var fileUploadElement = firefox.FindElementByName("file");  
fileUploadElement.SendKeys(@"C:\Users\Filip\Desktop\sunset.jpg");
```


```
var name = Guid.NewGuid();
```

```
var nameElement = firefox.FindElementById("Name");  
nameElement.SendKeys(name.ToString());
```

```
var submitButton = firefox.FindElementById("submit");  
submitButton.Click();
```

```
Thread.Sleep(2000);
```


```
var firstTitle =  
    firefox.FindElementByCssSelector("div#result div.photo h3");
```



Peka på en fil som ska laddas upp



Skicka iväg vårt formulär



Vänta en stund, tills vi vet att vår  
AJAX-request fått ett svar och vår  
DOM har uppdaterats



```
// ASSERT
```

```
Assert.AreEqual(name.ToString(), firstTitle.Text);
```

```
firefox.Quit();
```

```
private static void RemoveOldPhotos(FirefoxDriver firefox)
{
    while (true)
    {
        try
        {
            var link = firefox.FindElementByCssSelector("div#result div.photo div a");
            if (link == null) break;

            link.Click();
            var alert = firefox.SwitchTo().Alert();
            alert.Accept();
        }
        catch
        {
            break;
        }
    }
}
```

Vår DOM uppdateras varje gång, så vi behöver leta efter nya referenser till alla länkar

Klicka på "OK" i Confirm-dialogen

Hittar vi ingen länk att trycka på kastas ett exception, då kan vi helt enkelt gå tillbaka till testet

**Vår ASP.NET MVC Applikation testat som  
om en användare besöker den!**

# När är detta bättre än enhetstester?

# Labb 4

# Fortsätt med Bildgalleriet

- Utnyttja Dependency Injection i era Controllers
- Skriv tester för era Actions i ett nytt testprojekt
- Testerna skall inte gå hela vägen ner till databasen
- Skapa ett testprojekt för automatiserade UI tester med hjälp av Selenium

[mail@filipekberg.se](mailto:mail@filipekberg.se)

