# Sök

Problem: En användare kan manipulera söket.

Exempel: Skriv in följande sträng i sökfältet. Abba&limit=2  
Användaren har lyckats manipulera applikationen genom att styra parameterar som skickas till Spoitfys API.

Lösning: Innan söksträngen skickas till Get-metoden ska den encodas.

T.ex. använd klassen urlhelper som finns i .net

Där blir nya söksträngen: "abba%26limit%3d2"

# Dependecy injection

Problem:

Jag bygger på en struktur där jag antingen skapar en statisk klass, eller att jag skapar nya instanser av en klass när jag behöver komma åt klassens metoder.

Lösning:

Detta går att lösa genom att använda dependency injection designmönster. Med dependency injection kan jag lättare jobba mot klasser som är ”loosely coupled”, vilket innebär ”single-responsibilty”.

Fördelar med det är:

* Testbar kod.
* Lättare att förvalta
* Lättare att bygga ut

Exempel på Dependency injection lösningar:

Install unity: <https://www.dotnettricks.com/learn/dependencyinjection/dependency-injection-in-aspnet-mvc-4-using-unity-ioc-container>

Install Ninject: <https://github.com/ninject/Ninject/wiki/Dependency-Injection-With-Ninject>

# 

# Token

Problemet:

I min lösning hämtar jag ett nytt token varje gång som jag gör ett anrop mot Spotifys Api.

Lösning:

Då min applikation inte kräver att användaren identifierar sig med ett eget användarnamn eller lösenord kan jag använda samma token för varje aktiv session.

Enligt ovan har jag lagt till dependency injection. Då kan jag skapa en service som hanterar mitt token. Jag registerar att den servicen ska vara av typ singleton. Eftersom att min serviceklass är av typ singleton kommer bara en instans av klassen leva så länge applikationen lever.

# Data som skickas till vyn

Problem:

Mycket data som skickas till vyn idag används inte. Detta kan skapa förvirring för andra utvecklare som kommer in i projektet. T.ex. en själv efter 6 månader.

Lösning:

Utforma vymodeller där jag bara skickar med data som vyn bryr sig om.

# Settings data

Problem:

Idag är t.ex. användarnamn och lösenord hårdkodat i applikationen.

Lösning:

Använd config-fil för att hantera denna typ av data. Detta gör det enkelt att deklarera olika värden till olika typer av miljöer. T.ex. transformationsfiler för Test, staging och produktionsmiljö.

# Hantering av parameterar

Problem:

Mycket hårdkodade paramterar i applikationen idag.

Lösning:

För att göra lösningen mer dynamisk kan jag hantera parameterar till anrop med en dictionary. Denna måste såklart valideras innan den används i anropet.