Slutrapport

Snackis-forum

Rickard kujala

[rickard.kujala@iths.se](mailto:rickard.kujala@iths.se)

tfn: 076-6111714

**1. Beskrivning av projektet**

Ett webbaserat diskussionsforum där besökare har möjlighet att skapa registrera sig och göra inlägg, skapa trådar, skriva meddelanden eller gruppchattar. Forumet är uppbyggt med en “trädstruktur” där en “Huvudkategori” eller ett “forum” har en eller flera “underkategorier” eller “trådar” vilka i sin tur rymmer inlägg/poster knutna till tråden.

**2. Tidsöversikt**

*Startdatum:* 17 maj 2021

*Beräknat slutdatum:* 17 juni 2021

*Faktiskt Slutdatum:* juli 25

*Beräknad tidsåtgång:* ca: 150 timmar

*Faktisk tidsåtgång:* ca: 150 timmar

*Orsaker till försening:* Tids Bortfall pga VAB, 13 arbetsdagar (ca: 104 arbetstimmar) vilka jag efter bästa förmåga kompenserat för sena kvällar samt helger med varierande resultat.

**3. Tekniska krav**

*Teknisk plattform:* Windows 7 el. senare, Ubuntu el.senare

*Processor:* 1 Ghz

*RAM:* 512 MB

*Hårddiskutrymme:* minst 4,5 GB

*Ramverk:* .NET CORE 5

* Microsoft.Asp.NetCore.App
* Microsoft.NetCore.App

*Serverlösning:* IIS /Apache

*Bibliotek/nuget paket:*

* Mongo.DB.Driver(2.12.3)
* Swushbuckle..AspNetCore(5.6.3)
* Microsoft.Asp.NetCore.Identity.EntityFrameworkCore(5.0.6)
* Microsoft.Asp.NetCore.Identity.UI
* Microsoft.Asp.NetCore.Mvs.Razor.RuntimeCompilation(5.0.7)
* Microsoft.EntityFrameworkCore.SqlServer(5.0.6)
* Microsoft.EntityFrameworkCore.Tools(5.0.6)
* Microsoft.VisualStudio.Web.BrowserLink(2.2.0)
* Microsoft.VisualStudio.Web.CodeGEneration.Design(5.0.2)

*Databaslösningar:*

* MS SQL för att spara användare och relaterad data
* Cosmos MongoDB för att spara kategorier, alla poster samt övrig data relaterat till dessa

*Servermiljö:* Azure /egen server

**4. Applikationsstruktur**

*Design patterns:*

Dependency injection(DI)

Tack vare att koden är byggd med dependency injections är den lätt att underhålla eller ändra efter behov och önskemål. Detta eftersom DI reducerar antalet komponenter som är direkt beroende av varandra och istället injicerar dessa klasser i startup eller i en constructor.

***Projekt/Solutions:***

*Huvudprojekt/applikation:* Snackis.sln

* Dependency injections i Startup.cs

- IUserRepository, Interface till UserRepository.

- IChatRepository, Interface till ChatRepository. Hanterar CRUD operationer för chat och gruppchat via API

- IPostRepository, interface till PostRepository. Hanterar CRUD operationer för poster, kategorier

*Micro service:* Snackis.API.sln

* *Controllers :*

- ChatConroller

- PostController

* *Dependency injections i Startup.cs:*

- IChatDAL\*, interface till ChatDAL. Hanterar CRUD operationer för chat och gruppchatt mot databas (Cosmos mongoDB)

- IDAL, interface till DAL. Fungerar som “Data Access Layer” till cosmos mongo databas.

-IPostDAL\*, interface till PostDAL. Hanterar CRUD operationer för poster, kategorier mot cosmos mongo databas

Samtliga inlägg som postas på forumet går igenom ett “fula ord filter” i post controllerns “create” funktionen innan de sparas på databasen och blir synliga på forumet.

\*Fungerar ej som ett “Data access layer” trots att namnet antyder det. Klasserna ChatDAL samt PostDAL innehöll från början DAL men flyttades till en egen klass under projektets gång pga säkerhet och refaktorering enligt “DRY-code” principen.

**5. Installation**

**Snackis Webapp:**

I visual studio:

gå in på Package Manager Console, skriv sedan “update database” i konsolfönstret. Publicera sedan via Azure “publish to Azure”

I appsettings.json:

“SnackisContextConnection” ,här klistras anslutningssträng till MSSQL server in

"SnackisAPIPost", här skriver du URL till “Post” controller

"SnackisAPIChat", här skriver du URL till “Chat” controller

**Snackis API:**

I visual studio:

Publicera via Azure

I appsettings.json :

“CollectionName”, här skriver du namn på kollektionen

“DbName”, här skriver du namn på databasen

“Host”, här skriver du host

“Password”, här skriver du lösenordet

“UserName”, här skriver du användarnamnet

OBS: Ovanstående upprepas för båda controllers, “Chat” respektive “Post”

**6. Endpoints**

API et har två controllers, “ChatController” för meddelanden och gruppchattar samt “PostController” för poster, kategorier, subkategorier och forumnamn.

ChatController

Get:

* /api/chat- hämtar alla chattar
* /api/chat/{id}- hämtar alla chattar med ett specifikt id

Post:

* /api/chat- sparar en chatt

Put:

* /api/chat{id}- uppdaterar en chatt med ett specifikt id

Delete:

* /api/chat{id}- raderar en chatt med ett specifikt id

PostController

Get:

* /api/post- hämtar alla poster
* /api/post{id}- hämtar poster med specifikt id

Post:

* /api/post- sparar en post

Put:

* /api/post{id}- uppdaterar en post med specifikt id

Delete:

* /api/post{id}- raderar post med specifikt id

***7.* Funktioner på sidan**

*Besökare ej inloggad:*

Som besökare utan inlogg har man möjlighet att läsa se kategorier och dess innehåll samt läsa poster.

*Besökare inloggad:*

Samma som ovan men med möjlighet att

* skapa nya trådar
* svara på inlägg
* gilla/ogilla poster
* rapportera olämpligt innehåll
* använda meddelandefunktionen med privata konversationer och även gruppchattar

*Admin:*

Som admin tillkommer även möjlighet att

* Ändra namn på forumet
* Lägga till eller ta bort kategorier och underkategorier
* Granska och hantera anmälda inlägg
* Ta bort /stänga av användare

**8. Brister**

* API et saknar autentisering vilket gör det möjligt att för vem som helst att ändra, spara, hämta eller radera alla inlägg, meddelanden, poster osv.
* förutom poster sparas även kategorier, subkategorier och forumnamn i container för poster. dessa skulle med fördel lagras i egen container (detta var ett medvetet val för att minska datamängd på Azure)

**9. Förbättringsförslag**

* Möjlighet att på sin profil kunna ha en personlig beskrivning, även möjlighet att lägga upp länkar eller annan mer detaljerad information.
* Förbättra gränssnittet på “adminsidan”
* Möjligheter att blockera användare
* Möjlighet att filtrera eller söka på meddelanden, exempelvis efter datum eller namn på avsändare.
* Med javascript optimera användarupplevelse, t.ex. ge tumme upp eller ner där ett javascript sköter det som syns på sidan medan anrop till API görs i bakgrunden utan att sidan behöver laddas om.

**10. Sammanfattning**

Arbetet med projektet har i det stora hela gått bra. Det har varit stimulerande, lärorikt samt bitvis utmanande.

Har lärt mig många nya saker samt snubblat över en hel del intressanta saker som jag mer än gärna hade utforskat mycket mer, frestande som det har varit har jag avstått för att hinna till deadline. Exempel på några sådana saker är att arbeta med javascript för en mer responsiv sida, utforska bootstrap än mer grundligt och djupgående eller ett ordentligt djupdyk i microsofts “identity-lösning”.

Den största delen av tiden har tveklöst gått till databasmodellering, trots att uppgiften huvudsakligen handlade om webbprogrammering. Tidsmässigt tror jag ändå detta var klokt då problem pga en slarvigt planerad databas förmodligen resulterat i ännu mer tidsförlust. Inhämtad kunskap är ju heller aldrig en förlust!