Hitta Mitt Hem

Peter Emilsson

2014-01-18

Webbteknik II

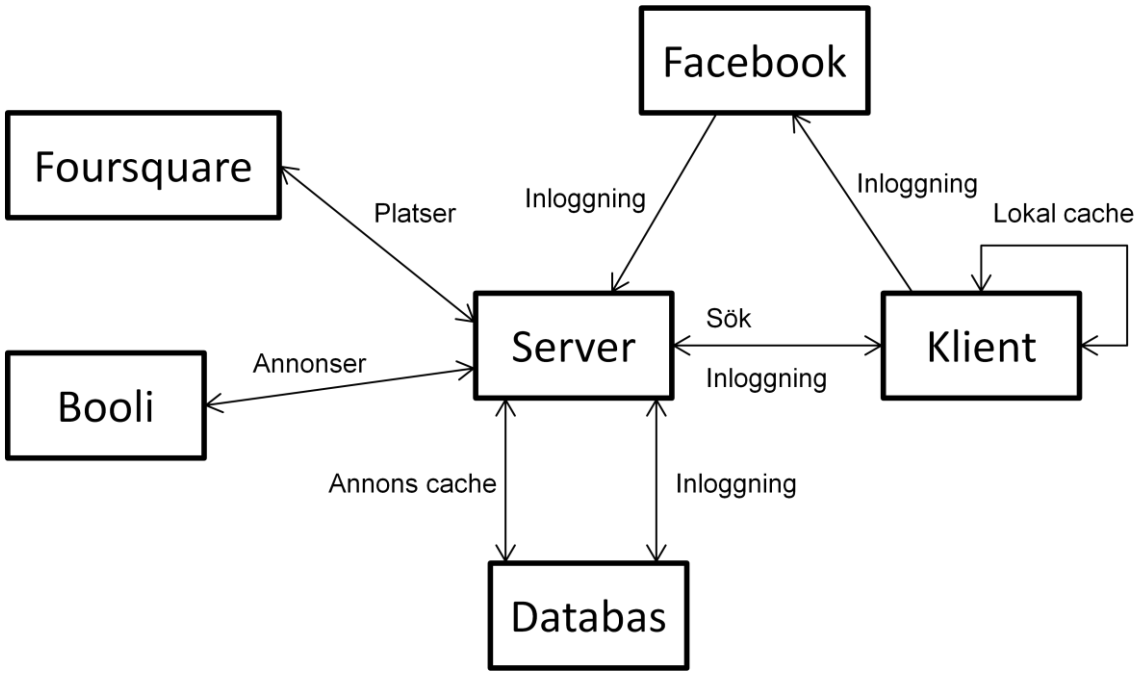
1DV449

Innehållsförteckning

Inledning

Jag har gjort en applikation där det går att söka efter bostadsannonser och samtidigt söka efter närliggande platser från Foursquare. Foursquare är en tjänst för att söka och betygsätta olika platser och samhällsfunktioner. I min applikation går det på ett enkelt sätt att söka och filtrera sökningar efter bostäder. Samtidigt som man söker efter bostäder går det att söka på närliggande platser. Letar man efter en villa i t.ex. Kalmar kan man samtidigt söka efter skolor och vårdcentraler i Kalmar och få upp två listor sida vid sida med resultaten.

Dataflöde



Serversida

Serversidan består av en server med ASP.NET MVC 4 samt datalagring med MS SQL Server 2008 R2 som hanteras av Entity Framework. Applikationen är uppdelad i två delar, en domänmodell och en webbmodell. Domänmodellen exponerar en serviceklass som innehåller de metoder som applikationen behöver för att fungera. De controllers som sedan behöver användning av denna service använder sig av "dependency injection" för t.ex. att göra det möjligt att på ett enkelt sätt testa kontrollerna genom att byta ut servicen. För att detta ska fungera så ärver serviceklassen från en basklass som i sin tur ärver från ett interface. Detta gör att det enkelt går att byta bakomliggande struktur för service klassen, utan att användaren av serviceklassen ska behöva ändra något. De som använder sig av serviceklassen kan då enkelt byta ut den till en annan serviceklass, så länge som den ärver från rätt basklass. För att förenkla datalagringslagret så används ett mönster som kallas "Unit of Work Pattern" och "Generic Repository". Detta gör att man inte behöver skapa en DAL klass för varje ny tabell som skapas i databasen. Det går att använda samma DAL klasser men med nya entiteter.

För att göra anropen till de två API:erna i applikationen används två stycken klasser, en per API. Klassen för att göra anrop mot Boolis API har en publik metod som används för att söka efter bostadsannonser. Den metoden tar minst en parameter som är söktermen som ska sökas med, t.ex. "Kalmar". Den har även ett antal valfria parametrar som filtrerar sökningen på t.ex. annonstyp (Villa, Lägenhet etc.), maxpris. Det går även att välja om man vill hoppa över ett antal annonser, exempelvis hämta annons 30-60, vilket används till att visa flera sidor med annonser. När en sökning görs så returneras resultatet i JSON som sedan görs om till c# objekt. Ett "container" objekt som innehåller vad som söktes på och hur många annonser det finns. "Container" objektet innehåller sedan en lista med alla annonser som objekt.

Klassen för att göra anrop mot Foursquares API har en liknande uppbyggnad. Sökmetoden i den klassen tar två parametrar, söktermen t.ex. "Kalmar" och vilka olika kategorier som sökningen ska innehålla. Den returnerar en lista med alla platser som hittades. Det går även att hämta alla kategorier som finns från Foursquare. Kategorierna består av ett fåtal huvudkategorier med många barnkategorier. För att få över det till c# struktur används en rekursiv metod för att lägga till alla barnkategorier för varje kategori.

Om något av API:erna skulle returnera ett felmeddelande (HTTP koder 4xx - 5xx) så fångas dessa upp och kontrolleras av servern. Beroende på vad det är för något fel så kastas ett nytt undantag som den som anropade metoden får ta hand om. Om Foursquare API:et returnerar ett "failed_geocode" fel så kastas inget undantag utan det skickas bara tillbaks en tom lista.

Serviceklassen som domänmodellen exponerar innehåller ett antal metoder för att göra sökning och att få tillbaks svar för t.ex. "autocomplete". Alla sökningar som görs efter annonser från Booli lagras i en databas där de är giltiga i 3 timmar innan det ska uppdateras genom en ny sökning mot Boolis API. När en sökning ska göras efter annonser på Booli så kontrolleras det först om en exakt likadan sökning har gjorts tidigare. Om en sökning har gjorts tidigare och inte är för gammal så returneras den. Om det inte har gjorts en sådan sökning innan eller om den är för gammal sker en ny sökning mot Boolis API och resultatet sparas i databasen. Om användaren är inloggad så sparas också de tio senaste sökningar som den användaren har gjort. Sökningar efter platser hos Foursquare sker genom att en metod tar en sökterm och namnen på de kategorier som ska sökas efter. API:et vill ha listan med kategoriernas som ID:n, applikation använder sig av kategoriernas namn. Kategorierna som skickas med till metoden kontrolleras mot databasen och om det är samma antal ID:n som hittas i databasen så görs ett anrop till Foursquares API. Resultaten från Foursquares API sparas inte i någon databas. Serviceklassen innehåller även två metoder för "autocomplete". Dessa returnerar platser som har sökts efter eller kategorier som innehåller den medskickade termen. Alla de olika Foursquare kategorierna lagras i databasen och kan uppdateras genom att en metod anropas i serviceklassen.

Webbmodellen innehåller den delen av applikation som direkt kommunicerar med klienten. Detta sker genom ett antal controllers. Alla parametrar som skickas med till någon controller valideras genom vymodeller. För att kontrollera om de klarar valideringen används ett "Action Filter" som validerar indata innan servern går in i metoden i den anropade controllern. Detta gör att all data som kommer in i en controller kan ses som giltigt och redo att användas. Controllerna som används till AJAX anrop returnerar data som JSON och om något går fel, validering eller serverfel så skickas ett HTTP fel till klienten med information om vad som gick fel. För att validera de olika annonstyperna som finns används en uppräkningsbar typ på servern där alla olika annonstyper finns.

När klienten begär startsidan (HTML) så skickas även alla de olika annonstyperna som finns med i en scripttagg. Om användaren är inloggad skickas också de 10 senaste sökningarna med. Detta för att minska antalet HTTP anrop som behövs.

För användarhantering används ASP.NET Simple Membership med lokal inloggning och genom Facebook.

Klientsida

Klientsidan består av två delar. En del som är en SPA (Single Page Application) och sedan en del där en användare kan registrera sig, logga in och hantera sitt användarkonto som inte är en SPA. SPA-delen är uppbyggd med hjälp av JavaScript-ramverket AngularJS. När användaren först laddar in sidan så initieras applikationen. Då hämtas bl.a. de två sakerna som skickas med i HTML när sidan först laddas. De sparas i minnet och tars bort från HTML strukturen. Sedan presenteras sökformuläret. Sökformuläret är uppbyggt av två vyer. Dessa vyerna är vanliga HTML filer med HTML kod och även AngularJS specifika attribut och element. Textfältet för att skriva var annonserna kommer ifrån (t.ex. Kalmar) har "Autocomplete". När användaren har skrivit minst två tecken så skickas de till servern och jämförs mot tidigare sökningar. För att kunna filtrera sökningen som görs går det även att sätta ett maxpris och en maxhyra för annonserna. Annonstyperna som skickades med från servern vid sidladdning visas som checkrutor som kryssas i om man begränsar sökningen till en eller flera annonstyper.

Samtidigt som användaren gör en sökning på bostadsannonser kan den också välja att söka efter närliggande platser på Foursquare. Vilka sorts platser detta ska vara bestäms genom att användaren t.ex. skriver "Food" i ett textfält. Vad som händer då är att söksträngen som användaren har skrivit skickas till server och servern returnerar alla Foursquare kategorier som innehåller den sekvensen av tecken. Det returnerade resultatet visas sedan i en lista under textfältet. Användaren måste då välja ett värde ur listan. Det går inte att välja ett värde som man själv har skrivit. Användaren kan lägga till en eller flera kategorier som ska sökas efter.

Hela formuläret valideras hela tiden genom AngularJS. Alla formelementen är bundna till AngularJS genom tvåvägs databindning. Detta gör att så fort ett värde ändras i formuläret så ändras det också i den bakomliggande JavaScript koden. AngularJS har ett antal inbyggda validerings direktiv som används för att validera. All data valideras också på servern, om ogiltig data skickas till servern returnerar den HTTP kod 400 tillsammans med vad som var fel. Applikationen presenterar fel validerings fel och server fel för klienten.

När användaren väljer att göra en sökning så hämtas informationen från formuläret och läggs in i URL:en. Controllern gör då en ny sökning genom att hämta ut de nya värdena från URL:en och gör ett AJAX anrop till servern. Om detta AJAX anrop returnerar ett lyckat resultat (HTTP kod 200) så lagras resultatet i lokalstorage. Alla sökningar som sedan sker med exakt samma parametrar hämtas från localstorage istället för från servern. Ett sökresultat lagras i tre timmar på server och lika länge på klienten. Servern returnerar som mest 30 annonser, men många sökningar kan innehåller fler annonser än så. Vid varje resultat skickas också information med hur många annonser det finns totalt. Detta används sedan för att lägga ut resultatet på flera sidor. Här hjälper den lokala lagringen till mycket. Om användaren bläddrar fram till nästa sida men sedan vill gå tillbaka, behöver det inte ske nya AJAX anrop hela tiden. Sökresultatet presenteras i en lista med alla annonser. Listan går att sortera efter pris.

Eftersom en sökning sker genom att URL:en uppdateras så går det att kopiera hela URL:en och sedan klistra in den i ett annat fönster och ändå trigga en sökning när applikation laddas. Eftersom data också valideras på servern så valideras även denna sökning.

Alla AJAX anrop som görs mot servern är skyddade mot CSRF genom att ett värde från HTML koden skickas med vid varje anrop och valideras. Detta gör att ingen annan öppen sida kan göra AJAX anrop till servern. Om något skulle gå fel på servern eller om Boolis API skulle ligga nere så returneras ett HTTP 500 fel tillsammans med information om vad som gick fel. Om Foursquare API:et inte är tillgängligt så returneras inget fel. Detta eftersom sökningar mot Foursquare görs bara om några annonser från Booli har hittats. Det är då onödigt att ett ens returnera dem bara för att Foursquare inte var tillgängligt.

Risker

Riskerna med en applikation som denna är samma risker som nästan alla SPA och mashup applikationer har. Applikationen är obrukbar om användaren inte har JavaScript aktiverat. I dagens teknikvärld så är det mycket sällan någon kör utan JavaScript aktiverat, men det finns ändå personer som gör det. Eftersom applikationen är till för att söka bostäder så kan det också vara så att någon användare kanske sitter på en mycket gammal dator som har en webbläsare som inte stödjer det som krävs av JavaScript koden.

Om Boolis API skulle ligga nere så fungerar inte applikationen. Detta gör att man är beroende av en extern datakälla som inte garanterar att deras tjänst alltid kommer att ligga upp.

Reflektion

Beroende på hur jag ser på projektet så har det gått både bra och dåligt. Jag har lärt mig nya tekniker, men jag hann inte med i närheten av vad jag hade tänkt mig och projektet blev inte alls som jag hade tänkt mig från början.

Jag valde att använda JavaScript ramverket AngularJS till klientsidan. Efter att ha använt Backbone.js innan så har jag velat testa ett större ramverk. AngularJS visade sig vara ett bra ramverk att använda, även om jag bara har skrapat på ytan för vad det kan göra. Nackdelen med att använda AngularJS var att en stor del av tiden gick åt till att lära mig hur det fungerar. Jag hade inte sett en rad AngularJS innan jag började så det tog ett tag att lära sig grunderna. Det som var speciellt svårt var att förstå exakt hur "$scope" arv fungerar i alla olika delar av AngularJS. En mycket positiv sak med AngularJS är att det har ett stort "community" av användare så det finns nästan alltid något tillägg för det man vill göra eller om man har något problem så har någon annan redan haft det.

Som jag skrev under risker så är applikationen beroende av externa datorkällor. Detta märkte jag när jag utvecklade och Boolis API låg nere i någon timme. Booli hade någon intern ändring av sitt system som gjorde att API:et låg nere. Problemet var att felet som jag fick tillbaka var att jag inte var auktoriserad att använda API:et.

Jag ångrar att jag valde att göra samma projekt i båda kurserna. Även om den andra kursen gick bra så var det mycket extra arbete när egentligen allt jag gör är att kommunicera med JSON, från API till server till klient. Slutresultatet för server sidan blev ok, men om jag skulle kunnat använda en dokument databas så skulle det gått mycket snabbare. Även användningen av Simple MemberShip användarhantering gjorde att det tog tid att komma på hur jag skulle göra det med SPA, vilket jag till slut fick avstå.

I nuläget tycker jag inte att min applikation är värd att jobba vidare på. Just nu så är det enda mervärdet min applikation ger att man kan söka på två saker samtidigt. Något som skulle göra applikationen mycket bättre, men även mycket svårare att bygga är att t.ex. bara returnera sökningar som har x plats nära sig.

De delar jag anser vara betygshöjande är att jag har använt AngularJS som klientramverk. Detta till stor del för att jag var tvungen att också lära mig det från grunden innan jag kunde börja använda det.