Projektrapport - 1DV449

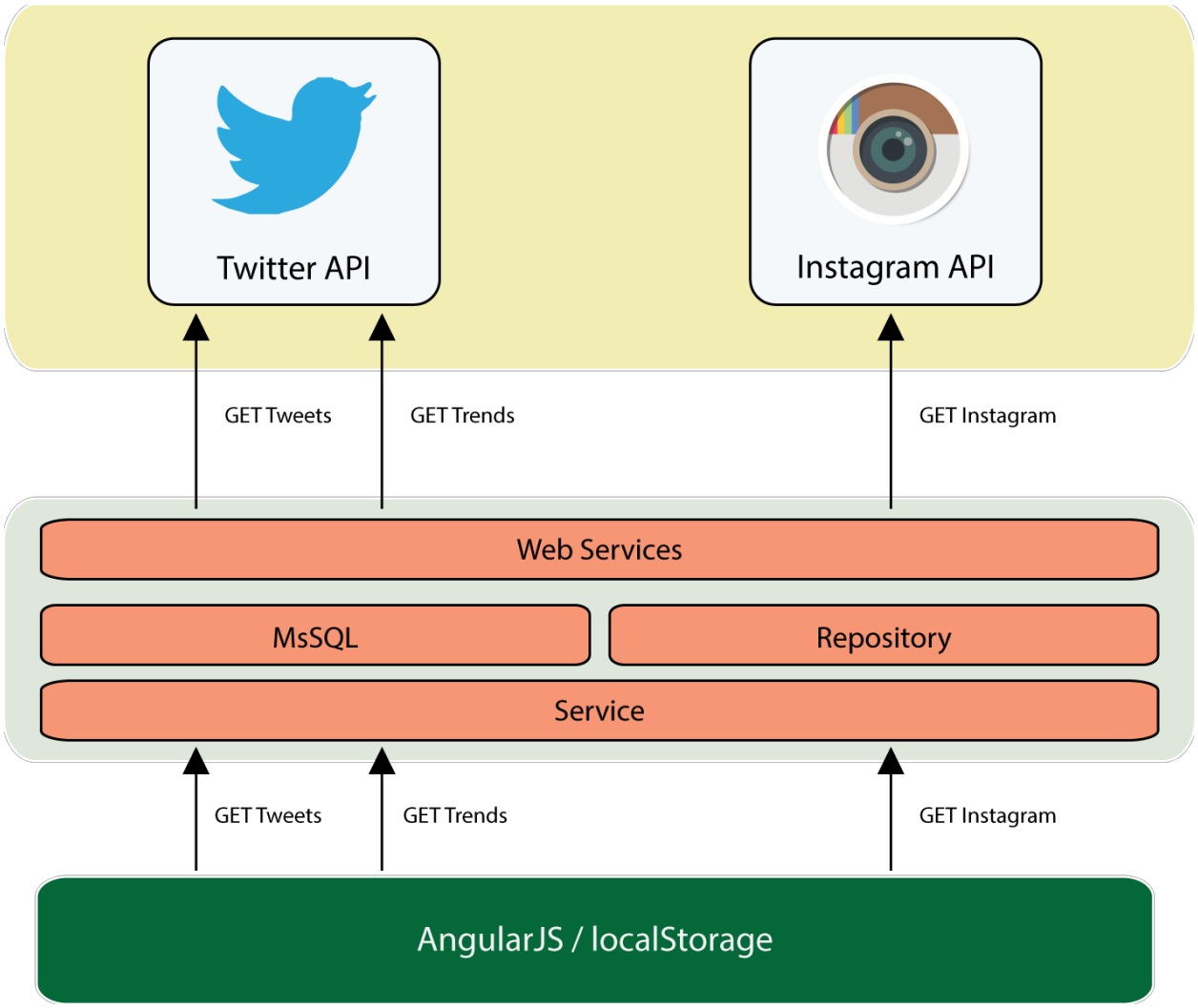
Rickard Karlsson

Inledning

Det jag valde att göra till detta projekt är en applikation som kollar upp vilka #-tags som används mest just nu av Sveriges twitteranvändare och sedan samlar in (och visar) ett utdrag av inlägg som använder dessa #-tags från både Twitter och Instagram.

Jag valde att göra detta för att jag tycker det är intressant att arbeta med stora API:er som Twitter och Instagram. Jag tror också det är en bra erfarenhet eftersom så mycket av Internets applikationer implementerar de stora sociala mediernas API:er på något sätt.

Här nedan är en väldigt högst förenklad bild över hur flödet i applikationen fungerar.



tt

Back-end

Kortfattat

För min back-end valde jag att arbeta med .NET ramverket och mer specifikt MVC 4. Till min hjälp för att sköta caching så använder jag Entity Framework tillsammans med en MsSQL databas.

Min back-end består i grunden av ett Servicelager som leverar data från antingen databasen eller hämtar ny från API:erna beroende på om det finns någon data lagrad och/eller om datan behöver uppdateras.

Som programmeringsspråk valde jag C#.

Cachning

När ny data har hämtats från databasen så får varje entity (trend, tweet eller instagram) en egenskap, ExpiresAt, vilket är en datum- och tidsangivelse för när datan anses vara “utdaterad” och borde förnyas. Datan sparas sedan med hjälp av Entity Framework i en databas.

När ett anrop sker (för att till exempel hämta tweets) så jämförs ExpiresAt med den nuvarande tiden och så länge datan fortfarande anses vara giltig så levererar min back-end den cachade datan istället för att hämta ny. Om datan däremot skulle anses vara utgången så hämtas ny data och så tas den gamla bort ur databasen och ersätts av den nya.

Felhantering

Som jag ser det så kan det ske två stycken fel.

Det första är generella fel som hanteras genom att fånga alla undantag i mina controllers och sedan returnera JSON med en statuskod och ett meddelande. I dessa fall så är det osäkert på vad som gått fel och därför leveraas ingen data alls.

Det andra fallet jag ser är om min back-end av någon anledning inte kan nå det API:et för att hämta data. Detta är till exempel om ett av mina API:er jag använder är nere för tillfället. I dessa fall så fångas undantaget upp i mitt Servicelager och istället för att slänga undantaget vidare till min front-end så levereras istället datan (om det finns någon) från databasen även om den kan vara utdaterad.

Front-end

Kortfattat

Min front-end består av en AngularJS single-page applikation som tar hjälp av modulen RESTangular för att göra anrop till min back-end. Cachningen på front-end sker genom localStorage.

Cachning

När användaren första gången besöker sidan så görs det tre anrop. Det första för att hämta trends och sedan två till när vi har trends för att hämta tweets och instagrams baserat på datan vi fick av det första anropet. Efter de första anropen så sparas all data som hämtades i användarens localStorage och vid varje kommande omladdning så kontrolleras datan i localStorage om den för det första existerar och sedan också om den är giltig. Om den inte är giltig så hämtar applikationen ny data från back-end.

Detta betyder att det inte sker onödigt många anrop till varken back-end eller till API:erna men också att sidan laddar så snabbt som det är möjligt.

Felhantering

Felhanteringen på front-end är enkel. Om något går fel vid anrop till back-end så försöker applikationen först arbeta med någon eventuell data som finns sparad i localStorage och om det inte finns någon data sparad och det inte går att hämta från back-end så visas ett felmeddelande.

Reflektion

Projektet gick bra överlag. Både MVC 4 och AngularJS är bra ramverk och det gjorde hela processen lättare. Jag brukar ofta ha problem med att arbeta med JavaScript och framför allt AJAX eftersom det kan vara svårt att hålla koll på alla asynkrona anrop som sker i en sån här applikation men med hjälp av $q och promises så gick det ändå förvånansvärt smidigt.

Jag är ganska så nöjd överlag med min kod förutom att det blev lite repetering av kod i AngularJS men det är mest för att jag själv är ovan och inte är helt säker på hur man ska bryta ut kod till komponenter.

Om jag skulle arbeta vidare på applikationen så tror jag att jag skulle implementera funktionalitet för att låta användaren söka på sina egna #-tags och kanske låta trender vara en sidfunktionalitet istället för huvud. Min back-end tillåter att man hämtar data från Twitter och Instagram med egna #-tags men inte min front-end så det vore inte allt för svårt att implementera.

En annan sak som jag skulle kunna tänka mig att implementera är ett gemensamt “interface” för datakällorna. Om jag hade haft detta skulle jag lättare kunna lägga till nya datakällor för att hämta in ännu mer data att visa för användaren.

Risker

Eftersom min applikation inte har någon egentlig användarinmatning så slipper jag väldigt många annars vanliga risker. Det jag istället ser som den största risken är att min applikation inte har tillräckligt bra cachning om användartalet skulle vara högt. Det jag menar med det är att om min applikation hade varit publik och på något sätt också väldigt omtyckt och använd så skulle Twitter och Instagram ganska så snabbt låsa ute mig.

Detta skulle man kunna lösa genom att noga tänka igenom hur många användare man har i genomsnitt och arbeta fram en lagom tid att cacha inlägg baserat på detta. Genom att i förväg hämta och cacha data skulle man också kunna minska antalet anrop som sker till API:erna.

En annan risk som finns med min applikation är att den är väldigt känslig för förändringar i API:erna. Mina datamodeller i back-end mappar sin data på ett väldigt hårdkodat sätt från mina API:er och om det sker en förändring i hur något av API:erna levererar sin data (förändring av JSON-struktur osv) så bryts min kod. Detta är en risk som existerar för alla applikationer som arbetar med API:er skulle jag säga och det betyder att man som utvecklare måste hålla sig uppdaterad och vara förberedd på att åtgärda eventuella problem som kan uppstå.