Portfolio

Sherief Badran (sb222rf)

Webbprogrammerare

LINNEUNIVERSITET VT-13

Portfolio Applikation

1 Abstrakt

Det finns mängder av portfolio-siter ute på webben, många som är extremt inspirerande med fantastiska bilder och professionell design. Många erbjuder visuellt godis med animationer och häftiga scrollupplevelser. Hur gör man en portfolioapplikation som berättar om mig och mitt jobb, samtidigt som den sticker ut bland en massa andra fantastiska portfolios? Hur kan jag utvecklas som programmerare samtidigit som jag skapar min portfolioapplikation?

Det är frågor jag ställde mig inför detta projekt, och försöker besvara i denna rapport.

2 INLEDNING

Vision

Min vision har varit att skapa en portfolio till mig själv i webbläsaren. Ett av de första kraven som sattes var att applikationen ska kännas som en ”play-ground” som lockar användaren till att klicka runt och prova appar och spel som ska demonstreras. En *unique selling point* för portfolion är att den i sig själv ska vara en applikation som imponerar i sitt utförande. En *singel page applikation* där målgruppen kan testköra appar och spel direkt på plats, utan att länkas iväg någon annanstans.

I ett tidigt skede av projektet tog jag även beslutet att utveckla *mobile first*. Min målgrupp är webbprogrammerare och/-eller rekryter av webbprogrammerare vilket innebär personer som med stor sannolikhet kommer att besöka min portfolioapplikation från en mobil enhet. Som webbprogrammerare vill jag att min portfolio ska vara tillgänglig i mobila enheter.

Syfte

Syftet med mitt val av projekt var att uppnå följande personliga mål:

- Skriva kod som är strukturerad
- Lätt att underhålla
- Testvänlig
- Html separerad från javascript
- Testvänlig kod
- Lätt att skala upp
- Prova på *enhetstesting*
- Prova på testdriven utveckling.
- Ytterligare utvecklas i arbetsprocesser med SCRUM och iterativ mjukvaruutveckling.

Genomförande och teknik

Portfolioapplikationen är utvecklad i följande miljö:

Backbone: Ett MV(C) ramverk. Javascriptkoden delas upp i modeller och vyer, vilket förenklade principen för *”separations of concern”*. Backbone används tillsammans med javascriptbiblioteken **underscore** och **jQuery**. Underscore har mestadels använts för att hantera html templates.

Require.js: Ett bibliotek för att dela upp koden i en modul per fil och ladda in dessa filer efter beroenden och behov (lazy loading). Jag har delat upp det så att varje backbone modell, vy eller

samlingsvy får utgöra en AMD-modul. Projektet innehåller också en del hjälpmoduler, t.ex. *Validator* som enbart utför validering.

Testmiljön består kortfattat av testramverket mocha, tillsammans med *phantomjs* som är en ”*huvudlös browser*” och assertion-biblioteket *chai*. Även när tester skulle skrivas var require.js mycket användbart.

Grunt: En javascript task manager som jag i huvudsak har använt för att köra tester i webbläsaren samt köra jshint.

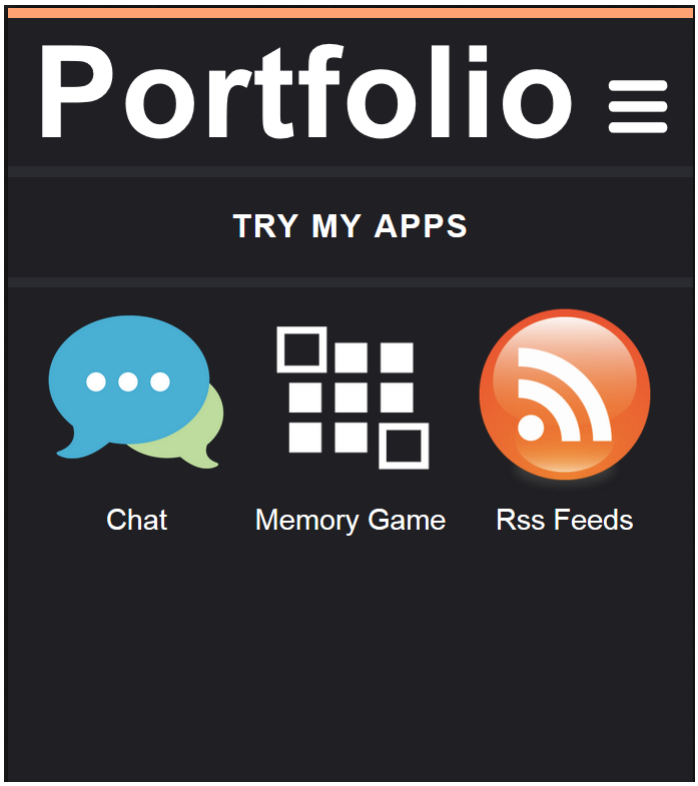
Nodejs: Med **npm** (node package manager) har jag laddat in moduler och paket till min utvecklingsmiljö. Node tillsammans med ramverket express har fått tjäna som applikationens server. Servern har kommunicerat med en nosql databas, MongoDB. I databasen lagras chatmeddelanden och meddelanden som är skickade via kontaktformuläret.

Resultat/Slutprodukt

Slutresultatet är en portfolioapplikation som visar upp en chat. När användaren går in på chatten får övriga anslutna reda på att en ny användare anslutit. En användare som ansluter till chatten måste ange ett smeknamn och kan därefter kommunicera genom att skriva meddelanden till övriga anslutna. Användaren kan också radera sina egna meddelanden. Om det dröjer innan chatten laddas möts användaren av en ”loader” som indikerar just detta.

En besökare som navigerar i applikationens drop-down meny kan välja att läsa lite information om mig, kontakta mig via ett kontaktformulär samt navigera tillbaka till startsidan.

Trots att det är en "single page application" så kan användaren välja att navigera med "fram-och-tillbaka" knapparna i webbläsaren. Jag har också gjort så att varje sida ska har en så talande url som möjligt, så att användaren kan få ytterligare en indikator på vart hen befinner sig i applikationen. Om användaren önskar kan en url användas för att surfa in på en specifik del av applikationen eller helt enkelt öppna en specifik app i en annan webbläsarflik.



Figur 1: Applikationens startsida.

Figur 2: Applikationens chat.

3 POSITIVA ERFARENHETER

Personliga Mål

Syftet med mitt val av projekt var glasklart redan från början. Jag har under en ansenlig period läst både böcker och artiklar om tillvägagångssätt för att skriva bättre javascriptkod, men aldrig fått varken tid eller möjlighet att försöka tillämpa mina teoretiska kunskaper.

Jag visste från början att mina personliga mål med projektet låter väldigt bra och är lätta att både skriva och säga men betydligt svårare att genomföra. Som väntat har jag inte uppnått mina mål till fullo men många av målen är delvis uppnådda. Jag är ändå väldigt nöjd eftersom jag gjort ett stort lyft i förhållande till hur jag tidigare skrivit javascript. Jag ser detta som början på en mycket positiv utveckling.

Att göra ett ajax-anrop till databasen, hantera event, ta emot data, skapa html och lägga in det i DOMen på en och samma plats i koden är något jag tror många kan känna igen från sitt tidiga arbete med javascript. Jag tror också att de flesta håller med om att det inte är hanterbart. Buggar i sådan kod kan slå ut hela applikationen och är dessutom svåra att hitta. Ytterligare ett problem är att sådan kod blir omöjlig att enhetstesta.

Skräckscenariot jag precis nämnde kan istället delas upp i olika ansvarsområden med hjälp av ramverket Backbone med dess modeller och vyer. En modell sköter anrop till servern för att skicka eller hämta data. En vy lyssnar efter nya händelser i modellen, för att sedan sköta html rendering och DOM-operationer. Vyns uppgift är också att lyssna till vad användaren gör för att i nästa steg eventuellt uppdatera modellen. Genom att dela upp koden på detta sätt går det mycket lättare att spåra eventuella buggar samt att göra ändringar i koden, utan att ändringarna sprider oönskade effekter på andra ställen.

Det blir också mycket lättare att enhetstesta dessa olika ”ansvarsområden”. Fungerar vyn som jag förväntar mig? Gör tillhörande modell sitt jobb?

I kombination med Backbone har jag använt mig av requirejs för att dela upp koden i moduler. Upplägget är en modul per fil och att varje modul ska begränsas till att utföra en uppgift som inte är alltför stor. Min tanke är att det upplägget ska medföra främst två positiva effekter:

1. Det blir enklare att testa att modulen faktiskt utför sin uppgift som förväntat.

2. Det blir enklare att bygga modulerna enligt principen för det som på engelska kallas för *loose coupling* (http://en.wikipedia.org/wiki/Loose_coupling), en innebörd av begreppet är att varje modul har så lite inbördes beroende som möjligt till en annan modul. Detta har inte varit lätt att tillämpa men har fungerat bra till viss del. Det återstår en del arbete på detta plan.

En positiv effekt jag har märkt är att buggar i kod som ansvarar för vissa delar av applikationen inte har haft någon negativ påverkan på andra delar av koden. Jag har också refaktorerat kod i moduler och nästan aldrig har det resulterat i fel utanför modulens ramar. Detta ser jag som mycket positivt eftersom jag tidigare alltför ofta fått erfara det motsatta.

Mer nya kunskaper

Ända sedan projektets start har javascript legat i fokus för mina personliga mål, därför föll det sig naturligt att välja nodejs som back-end lösning. Jag har länge varit nyfiken på node och tycker det har varit mycket givande och intressant att prova på. Jag har använt node på ett väldigt enkelt plan och inte på något sätt fördjupat mig i det. Men det är en start och jag har nu förutsättningarna att själv experimentera och fördjupa mig i hur node kan användas på flera sätt. Dock har det varit fullt tillräckligt inom tidsramarna för projektet.

En node server i kombination med databasen MongoDB föll sig naturligt även av anledningen att Backbone är som gjort för att skicka JSON - objekt till servern och i Mongo lagras data som JSON - objekt.

Under projektets start och initiala inlärningsprocess av Backbone, insåg jag snabbt att ramverket i grunden är anpassat för ett REST api på serversidan. Eftersom jag aldrig hört talas om REST eller RESTful api så läste jag på om det och implementerade sedan ett REST api i min server.js fil. Det var ett REST api som täckte full CRUD (post, get, put och delete) och efter en del tester visste jag att det gick att ansluta till en node-server med databas. Det visade sig senare att jag inte skulle behöva hela REST - implementationen och det mesta fick omarbetas. Dock är det något jag väljer att se som en positiv erfarenhet eftersom jag fick användning av delar av implementationen, och så småningom kommer att få användning av hela implementationen, samt att jag lärde mig både det praktiska och teoretiska kring ett nytt begrepp.

Anledningen att jag utvecklade ett REST api med full CRUD var kravet att chatten skulle innehålla funktionalitet för att både radera och uppdatera meddelanden. Eftersom jag saknade erfarenheter om hur en chatt fungerar ”under huven” så tänkte jag inte på tekniska detaljer om hur meddelanden skapas/raderas på flera klienter samtidigt. Därför gjorde jag ”misstaget” att först implementera full CRUD med ett REST api för att sedan backa tillbaka och läsa på om node och web sockets.

Därefter tog jag beslutet att använda mig av socket.io för att implementera kravet om att skapa och radera meddelanden i chatten. Funktionalitet för att uppdatera meddelanden är skjutet framåt i tiden. Trots buggar och problem har det varit mycket intressant att lära sig om node och socket.io

4 NEGATIVA ERFARENHETER

Min utvecklingsmiljö

Jag tycker att min utvecklingsmiljö har varit minst sagt utmanande. Under projektets elaborativa fas var det en mycket stressig upplevelse att sätta sig in i ca 8 st nya ramverk och bibliotek inklusive nya backend tekniker. Det fanns många riskfyllda moment som skulle rymmas inom ganska åtstramade tidsramar. Det tog lång tid att konfigurera och få alla parametrar att stämma för att kunna hosta projektet och som jag ser det åstadkomma delleveranser. Min första delleverans blev försenad vilket är allvarligt p.g.a. vikten av att släppa första delleveransen så snabbt som möjligt. Efter första delleveransen kunde jag dock leverera kontinuerligt.

När man ska välja en utvecklingsmiljö i javascript krävs det en omfattande research, eftersom det finns ett brett utbud av ramverk och bibliotek som liknar varandra, men fungerar lite olika beroende på bl.a. användningsområde och typ av projekt. Många av ramverken och biblioteken är dessutom ofullständiga på så vis att de är beroende av andra ramverk/bibliotek. T.ex. är ramverket Backbone beroende av biblioteket underscore.js och testramverket mocha.js är inte komplett utan sinon.js som i sin tur behöver kompletteras med sinon-chai om chai används som assertion-bibliotek. Allt detta blir väldigt rörigt och ställer till det en hel del under utvecklingsprocessen om man inte redan innan är fullt medveten om hur olika komponenter hänger ihop. Eftersom jag inte hade full koll på detta, innebar det att massor av tid gick åt till att åtgärda eller komplettera. Att komplettera sin utvecklingsmiljö med något nytt är inte helt smärtfritt, eftersom det kräver tid att sätta sig in i det man kompletterar med.

Min kod på klientsidan

I denna rapport har jag tydligt framhävt mina personliga mål med hur javascript koden ska skrivas. Som jag tidigare nämnt tycker jag att jag lyckats delvis men långt ifrån överallt.

Jag har spenderat mycket tid på att refaktorera kod och ändå återstår mycket jobb med den biten. Speciellt i min Backbone router fil som heter router.js. I den filen har jag haft ett krig mot mig själv. Jag har försökt att bryta ut kod i neutrala hjälpmoduler, vilka jag inte heller är helt nöjd med ännu. Ändå ser jag tydliga mönster av återupprepningar som måste brytas ut och generaliseras.

Jag har inte varit helt konsekvent i hur mina Backbone modeller och vyer är uppbyggda. Backbone som ramverk upplevde jag som lätthanterligt fram till att projektets omfattning växt till en viss nivå. Jag hamnade i svåra situationer där jag funderade på hur olika vyer hängde ihop, hur jag skulle välja att kategorisera vyer och om en vy kan lyssna till vad som händer i andra vyer? På grund av tidsbrist valde jag att bortse ifrån dessa frågeställningar och tog istället beslut som inte var av någon vidare kvalitét. En del mindre efterforskningar fick mig att inse att ramverket Backbone kan kompletteras med en del moduler som skulle kunna lösa en del av de

saker jag upplevde som problematiska, vilket återigen påvisar delar av den problematik som beskrivs i föregående rubrik.

I övrigt har jag sett delar av min kod där jag slarvar med vad en vy bör göra och vad en modell bör göra. Det är helt upp till användaren av ramverket Backbone att undvika exempelvis att logik som hör till modellen hamnar i vyn eller att logik som hör till vyn hamnar i samlingsvyn (en överordnad vy som omsluter flera mindre vyer med 1:1 relation till sina modeller).

Min kod på serversidan

Följande punkter redovisar briser, problem och åtgärdspunkter angående kod på serversidan:

- Min kod i filen server.js är skriven på ett ostrukturerat sätt och all kod är samlad i en enda fil. Detta kommer att åtgärdas genom att dela upp koden i *node modules* och sedan inkludera dessa till filen server.js.

- Ingen förberedd testmiljö eller tester är skrivna mot serversidan.

- Har ej hunnit djupdyka i hur man implementerar säkerhet och validering med node. Det är definitivt något som återstår att göras. Jag antar att jag kan återanvända valideringskod från klientsidan på servern. Det enda jag gör i nuläget är att tvätta inkommande data från skadlig html och javascript (skydd mot xss). I skrivande stund är detta inte tillräckligt bra gjort för inkommande data från mitt kontaktformulär. Jag prioriterade den säkerhetsåtgärden för chatten eftersom att data därifrån ska tillbaka ut på klienten.

- Jag har läst lite om JSON - hijacking men är inte tillräckligt insatt för att vidta några säkerhetsåtgärder på klienten eller servern inom ramarna för tidsprojektet.

Min testmiljö

När jag påbörjade projektet hade jag betydligt högre ambitioner vad gäller enhetstester än vad som faktiskt gjordes. Jag var säker på att hinna med en kortare period av testdriven utveckling. Till min stora besvikelse blev det inte så. Under projektets början var jag mycket nöjd med min testmiljö och hade ordnat så att jag kunde sätta mina tester i *watch-mode.* Mina tester kördes då under själva utvecklingsarbetet. Min plan var att skriva tester mot delar av koden som utgjorde applikationens viktigaste funktionalitet med hjälp av *watch mode* och i realtid kontrollera att refaktorering eller ny implementation inte skapade fel eller problem för de testade delarna av koden.

Den planen gick i stöpet efter att jag påbörjat implementation av funktionalitet med web-sockets. Av någon outgrundlig anledning upphörde min testmiljö att fungera. Mina tester ville inte köras om jag inte stängde ner min socketanslutning till servern. Jag spenderade mycket tid på att försöka lösa problemet men lyckades inte. Detta var, och är fortfarande, en av projektets största besvikelser. Istället för att direkt kunna fortsätta utvecklingen av min portfolio efter tidsramarna för detta projekt, måste jag lägga tid på att reda ut problemet med att mina enhetstester inte vill köras.

5 SAMMANFATTNING

Sammanfattning och slutsats

Javascript är ett språk som kräver en ganska komplex sammansättning av ramverk och bibliotek för att större typer av projekt ska hålla kvalité, vara underhållsvänlig, testvänlig och skalbar. Problemet är bara att utbudet är väldigt brett och spretigt. Det saknas en allmängiltig paketlösning. En sådan lösning skulle dessutom behöva vara mycket väl dokumenterad och lätt att komma igång med. Detta för att bespara utvecklingsteamet på massor av tid som går åt till att undersöka vilken sammansättning av komponenter som ger den bästa tänkbara utvecklingsmiljön till det tänkta projektet, men också bespara utvecklaren på obehagliga överraskningar.

Ett stort antal ramverk, bibliotek och *add-on´s* ökar riskfaktorn i projektet. Risken att något api gör arbetsprocessen omständig istället för att underlätta är ganska stor om man inte förskaffat sig tillräckligt med erfarenheter om hur ett ramverk används och kanske ännu viktigare hur flera ramverk/bibliotek används tillsammans. Även om många javascriptramverk liknar varandra är det viktigt att i förväg undersöka vad ramverket ifråga faktiskt är avsett för och vilka kompletterande API:er som kan komma att krävas för att programmera det som specificerats i användarfallen.

Med anledning av ovanstående har jag under projektets gång råkat ut för en del obehagliga och oönskade överraskningar. T.ex. att ramverk som jag trodde var kompletta behöver kompletteras eller att min tillsynes perfekta miljö för enhetstestning plötsligt slutar att fungera för att det uppstår en konflikt med någon av de många andra ingående komponenterna i miljön. Det blir tydligt att ju fler komponenter som ska samspela desto större risk för att oväntade konflikter uppstår. Detta är tyvärr något jag tror är ganska typiskt för en javascriptmiljö.

Utvecklingspotential

Applikationens utvecklingsområden kan redovisas i följande punkter.

- Längre fram kommer applikationens gränssnitt givetvis att anpassas även till större skärmar. Jag valde utvecklingsmodellen *mobile first* för att säkerställa att min portfolioapplikation går att besöka med mobila enheter. Jag är hellre i situationen att skala upp ett gränssnitt än att skala ner.
- Om något går snett i chatten eller någon annanstans i applikationen så hanteras inte detta med varken meddelanden till användaren eller med åtgärder för att hjälpa användaren att förstå situationen. Detta är förberett med try-catch block över kritiska partier av koden, dock återstår det att rendera ut begripliga meddelanden till användaren.
- Givetvis vill användaren kunna se datum och tid för när ett chat-meddelande skrivits. Detta saknas och ska implementeras.
- När användaren ansluter till chatten första gången efter att webbläsarens historik raderats måste ett nickname anges. På grund av tidsbrist implementerades detta med en prompt som kräver ett nickname. Detta är en mycket dålig lösning men fick bli en quick-fix tills att jag ordnat med antingen inloggning eller helt enkelt en html ruta med input för att ange ett nickname, och i så fall också möjligheten att byta nickname. I början av projektet tyckte jag att login var helt onödigt för en portfolioapplikation men har insett att det kan vara till stor nytta för olika applikationers funktionalitet.
- När en ny användare ansluter till chatten får alla anslutna användare reda på detta. Denna funktionalitet skall skrivas om samt att en sådan notifiering skall ske även när någon lämnar chatten.
- Funktionalitet för att se när en användare skriver något i chatten.
- Implementera en lista som visar vilka som är anslutna till chatten, hur skall detta presenteras på ett bra sätt på den mobila skärmen?
- Inmatningsfältet för chatmeddelanden måste omarbetas så att det är tillgängligt hur användaren än scrollar i chatten.
- När applikationen har login funktionalitet vill jag implementera möjligheten att ladda upp en profilbild som även visas som en del i användarens chatmeddelanden.
- Påbörja implementation av fler applikationer/spel att demonstrera. Har planer på att implementera ett memoryspel (eller annat kortspel) och använda css 3 tekniker samt ljud för att flippa runt spelkort.

Listan skulle kunna göras oändlig, det är bara fantasin som sätter gränser. Jag ser mitt portfolioprojekt som ett evighetsprojekt som jag kommer att ha som en hobby och som en plattform för att fortsätta lära mig nya saker. Det känns väldigt bra att ha en sådan plattform att bygga vidare på.