Projektplan, TNM094

Team Bruse
Daniel Rönnkvist
Klas Eskilson
Ronja Grosz
Erik Olsson
Therése Komstadius

4 mars 2015

Sammanfattning

Detta är en plan för utvecklingen av en webb-baserad databas för hantering av filer på *Dropbox*.

Innehåll

Kapitel 1

Kort beskrivning av projektet och dess mål

Projektet går ut på att skapa en multifunktionell webb-databas som är tillgänglig för flera olika användare. Det ska vara möjligt för användaren att installera tjänsten lokalt på sin egen dator. Målet med webb-databasen är att hantera olika typer av filer och kunna hitta dem på ett lätt och smidigt sätt med hjälp av nyckelord. Dessa nyckelord kommer vara sökningsbara och med hjälp av dessa ska olika typer av filer kunna visas upp enkelt och sorterade efter olika parametrar, exempelvis nyckelord, filtyp, namn eller liknande.

1.1 Frågeställningar

- Hur ska en webbdatabas struktureras för att sökning efter sparade objekt ska kunna levereras enligt en användares förväntnignar om hastighet och resultat?
- Hur går det att säkerställa att en användares information som lagras i en webbdatabas inte är tillgänglig för någon som inte är den specifika användaren eller har blivit auktoriserad av den specifika användaren?
- Hur kan filer i en webbdatabas presenteras i en webbapplikation på ett sätt som gör de överskådliga, hanterbara och lättilgängliga för en användare, i en filstruktur utan kataloger eller annan hierarki?
- På vilket sätt kan en webbapplikations användargränssnitt designas för att demonstrera all funktionalitet ett system besitter och göra det intuitivt för en användare?

Kapitel 2

Tidsplan

För att snabbt få en överblick över projektet och se distribution gällande till exempel resurser och arbetsfördelning är *Gantt*-scheman ett bra verktyg. Varje dag under projektets gång specificeras här, och samtliga händelser under projektets gång markeras i schemat. Projektets Gantt schema finns att se i bilaga A.

2.1 Tid för planering, efterforskningar och tekniska studier

En övergripande efterforskning görs under projektuppstarten. Under projektets gång kommer sedan ytterligare tekniska studier ske inför och under varje *sprint*, för att ta reda på hur vissa krav ska realiseras.

2.2 Sprints och deras möten

En sprint pågår i tre veckor. I början på varje sprint kommer det hållas ett planeringsmöte på sex timmar inför kommande sprint. Där bestämmer gruppen tillsammans vilka scenarion som ska genomföras inför kommande sprint. Ett sprintmål kommer att formas efter dessa scenarier som gruppen kommer jobba mot att nå. Det är önskvärt att ha en fungerande produkt efter varje sprint. De scenarion som är intressant flyttas sedan över från projektets produkt-backlogg till aktuell sprint-backlogg tillsammans med de uppgifter som finns för att utföra arbetet.

Varje arbetsdag kommer ett 15 minuter långt dagligt Scrum-möte att hållas. De kommer att vara stående möten utan datorer för att hålla dem fokuserade och effektiva. Där diskuteras vad gruppmedlemmarna gjorde förra arbetsdagen och vad de ska göra den kommande dagen för att hjälpa till att nå målet för aktuell sprint. Där utvärderas också möjligheten att slutföra de krav som finns i nuvarande sprint backlogg.

Som avslutning i varje sprint kommer två möten att hållas. Först hålls en sprintgranskning, som är tidsbestämd till tre timmar, där det arbete som precis genomförts utvärderas. Här är det även bra att ha med kunden som kan få en demonstration av systemet som hittills utvecklats. Här redovisas även de problem som kan ha stötts på och hur de har blivit lösta. Mötet ger också en möjlighet att uppdatera projektets produkt-backlogg om nya krav eller förändringar i existerande krav dykt upp.

Till sist hålls en sprintåterblick som är ett möte på två timmar. Här går gruppen igenom den sprint som genomförts utifrån sina egna arbetsinsatser, verktyg, processer och kommunikationen i gruppen. Det framförs konstruktiv kritik och görs upp en plan för hur arbetet kan förbättras ur denna aspekt.

2.3 Milstolpar och leverabler

Efter varje sprint skall produkten vara ett inkrement på produkten från föregående sprint. Detta innebär att en förbättrad version av produkten skall finnas redo för leverans efter varje sprint.

Efter den första sprinten är målet ha en så kallad *minimal viable product*. Användaren ska kunna skapa ett konto, eller logga in på ett befintligt, och lägga till filer från dennes Dropbox-mapp samt ge filerna nyckelord. Andra sprinten kommer till stor del handla om att göra filerna lättåtkomliga och sökbara för användaren. Tredje sprinten kommer att fokusera på att förbättra användagränssnittet för att gör det mer attraktivt för användaren. Användartester ska vara genomförda innan starten för sprint fyra. Resterande sprintar kommer att användas för att utveckla lägre prioriterade önskemål som delning av filer.

2.4 Avstämningsmöten med ledningsgruppen

Bör hållas ett par gånger i samband med att en sprint avslutas. Tanken är att detta kan hållas efter sprintgranskning och sprintåterblick, så att dessa möten kan hålla fokus på det som är viktigt för utvecklingsprocessen.

Kapitel 3

Infrastruktur för programmering och systemutveckling

3.1 Utvecklingsplattform

Alla i projektet utvecklar i en *UNIX*-baserad miljö. Detta valdes för att majoriteten hade vana i denna miljö och för att förenkla samarbetet.

3.2 Kravhantering

De kravspecifikationer som finns gällande systemets funktionalitet från en användarsynpunkt kommer från kunden. Här är det önskvärt att kraven blir tydligt definierade och att alla menar samma sak. Det ligger i detta skede inget fokus på hur kraven ska uppnås. Dessa krav diskuteras sedan av utvecklingsteamet för att ta fram kravspecifikationer, alltså de krav som krävs för att uppnå kundens krav. Alla kravdefinitoner sparas som scenarion i projektets produkt-backlogg. Dessa bryts ner i mindre kravspecifikationer i form av uppgifter. Det ska vara möjligt att genomföra ett scenario över en sprint och en uppgift på en arbetsdag. Det är viktigt att det finns mycket detaljer för att arbetet ska bli rätt från början. Det är önskvärt att ha ett möte med kunden efter varje sprint för att visa upp arbetet så långt i projektet och se om några krav har förändrats.

3.3 Scenario- och uppgiftshantering

Varje krav kommer att förvandlas till ett scenario som kommer sparas ner i backloggen. Dessa scenarion kommer sedan att brytas ner till mindre uppgifter. För att få en bra översikt på hur backloggen och uppgifter hanteras kommer webbtjänsten *Trello* att användas.

3.4 Versionshanteringssystem och rutiner

För att hantera projektets kod kommer *Git* användas tillsammans med *Github*. Git är ett etablerat verktyg för versionshantering av kod. För att lätt kunna dela koden lagras den på Github. Versionshantering sker genom förgreningar från huvud- och utvecklingsgrenen där all utveckling sker i grenarna. När utvecklingen i någon förgrening är klar ska koden granskas av minst två andra personer innan den

sammanfogas med huvudgrenen. Koden får endast sammanfogas med huvudgrenen av en person för att minska risken för att trasig kod.

3.5 Testningsprinciper och rutiner

Inför varje sprint kommer den testansvarige att säkerställa att det finns tester som kan möta sprintens krav. Testen kommer att skrivas för att upptäcka fel i den kod som skrivs för kraven, även för att upptäcka fel som skapas. Detta kommer att ske löpande under varje sprint efter att en funktion har implementerats.

I projektets början kommer enhetstester att implementeras. Senare i projektet när ett användargränssnitt har byggts kommer integrationstester att tillämpas genom användartester. Användartesterna utförs genom att filma en testperson och skärmaktiviteten samtidigt medan systemet manövreras.

3.6 Dokumentationsprinciper och rutiner

All dokumentation i form av textdokument och filer samlas i *Google drive*. Vid varje möte förs ett kort mötesprotokoll och alla tavelanteckningar sparas som bilder. Backlogg samt sprints dokumenteras på Trello som kommer agera som Scrumtavla. Backloggen beskrivs genom olika fall där det finns kategorier för fall som är antingen vedertagna, idéer eller slutförda. Varje sprint har en egen gren som innehåller tillhörande scenarion, uppgifter, pågående uppgifter, uppgifter som ska godkännas och godkända uppgifter.

För att generera lättöverskådlig dokumentation från koden används *TomDoc*. Då skrivs exempelvis klass- och funktionskommentarer i koden, som sedan genereras till en *HTML*-sida.

3.7 Modelleringsstandard och rutiner

Systemet kommer att modelleras upp som UML-modeller på tavla. Det ger hela teamet en möjlighet att vara delaktiga och få samma bild av systemet. Inför varje ny implementation bör nya strukturer modelleras för att säkerställa att samtliga utvecklare har en gemensam bild av hur strukturerna skall komma att se ut.

Kapitel 4

Organisation

4.1 Teammedlemmar och kontaktuppgifter

- Klas Eskilson, klaes950@student.liu.se, 073-730 73 56
- Ronja Grosz, rongr946@student.liu.se, 076-218 64 26
- Therése Komstadius, theko867@student.liu.se, 073-940 42 28
- Erik Olsson eriol726@student.liu.se, 073-840 68 72
- Daniel Rönnkvist, danro716@student.liu.se, 076-396 74 24

4.2 Ansvarsfördelning och arbetsgrupper

4.2.1 Erik Olsson - *Scrummästaren*

Det är Scrummästarens uppgift att se till så att Scrum-teamet efterföljer Scrum-teorin och få de att förstå hur den fungerar. Scrummästaren hjälper också produktägaren med att hantera produkt backloggen på rätt sätt. Scrummästaren ska även ansvar för att produktutvecklingen flyter på och att det råder en god arbetsro.

4.2.2 Therése Komstadius - *Produktägare*

Dennes uppgift är att hålla kontakt med kunden och i takt med det hålla kraven uppdaterade. Största ansvarsområde är att se till att projektets produkt-backlogg är uppdaterad och organiserad på ett sätt som gör att projektmålen går att nås. Det är även viktigt att alla förstår kraven som finns i den och att den är synlig och tydlig för alla.

4.2.3 Ronja Grosz - *Dokumentansvarig*

Dokumentansvarige har hand om att dokumentationen och dokumentationsprinciperna av projektet följs och utförs korrekt. Så som att se till att mötesprotokoll alltid förs vid varje möte av beslutsgrundande karaktär och att samla alla relevanta dokument.

4.2.4 Daniel Rönnkvist - *Testansvarig*

Som testansvarig har denne som uppgift att säkerställa att tester skrivs som kontrollerar att kraven för den aktuella sprinten uppfylls. Det åligger även denne att se till så att testning sker kontinuerligt och att tester skrivs inför och under varje sprint. Denne ska även jobba för att skriva tester för att upptäcka brister i systemet.

4.2.5 Klas Eskilson - *Kodansvarig*

Uppgiften för den kodansvarige är huvudsakligen ansvarig för kodgranskningen inom gruppen. Koden som begärs att sammanfoga med redan befintlig kod skall hålla en god kvalité och inte upprepa sig för mycket. Det är den kodansvariges ansvar att se till att denna granskning görs av minst två personer som inte var involverade i utvecklingen av det som ska sammanfogas. Till sin hjälp har den kodansvarige verktyg som automatiserar detta, tex *Code Climate*.

4.3 Mötesprinciper och rutiner

Dagar börjar 08:30 och slutar 17:00 om inte annat anges. För principer kring Scrum-relaterade möten, se kapitlet 2.2. Mötena kommer att ske varje onsdag till fredag under vårens första period, diskussion om vilka arbetsdagar som kommer vara aktuella under andra perioden sker senare. All information angående salsbokning och mötestider sker via *Facebook*-gruppen. Personen som bokar sal varierar beroende på hur många timmar denne har att utnyttja i salsbokningsystemet.

Kapitel 5

Teknisk beskrivning

5.1 Målplattform, behov av programvara, grundläggande systemarkitektur, standarder och APIer

Systemet kommer att utvecklas i *Ruby* med ramverket *Ruby on Rails*. Detta ramverk är byggt enligt designmönstret *Model-View-Controller*, där olika delar i systemet är nedbrutna i olika komponenter. I *model* hanteras databasen och dess innehåll. I *view* lagras alla olika vyer, tex hur det ser ut när en användare ombeds logga in eller när någon besöker startsidan. *Controller* fungerar som spindeln i nätet. Här finns logik, och denna komponent förser view med data från model.

För databashantering används Ruby on Rails inbyggda *ActiveRecord*-modul, som bygger på *object relation model*. Detta innebär att det är enkelt att strukturera upp relationer mellan olika tabeller och inlägg. Exempelvis kan man säga att en användare har många filer, och på så sätt låta systemet enkelt lista en användares filer, utan att skapa avancerade *SQL*-frågor.

För att användarens användarupplevelse inte ska avbrytas utav att sidan laddas om för ofta kommer många anrop till ett eget API att ske via *Javascript* och med hjälp av *AngularJS* presentera detta för användaren. Detta för att sträva efter en snabb och effektiv upplevelse för användaren. Till exempel vid sökning så kommer anrop ske löpande och resultaten renderas i webbläsaren.

5.2 *Build*-miljö, *IDE*, kompilator, *debug*-verktyg, profileringsverktyg

Då Ruby on rails inte kräver en kompilator kommer valet av textredigerare vara upp till programmeraren. Ett alternativ till en vanlig textredigerare är att använda *IDE*:n *Rubymine*, som går att använda gratis som student.

De *debug*-verktyg som redan är inbyggda i rails kommer i huvudsak att användas, men även verktyg så som; *byebug*, *better-errors* och *did-you-mean*.

För testning kommer verktyget *minitest* att implementeras och tredjepartstjänsten *Travis CI* kommer att kopplas till systemet för att användas som ett komplement till kodgranskning för att försäkra kodstandard. Minitest kommer även att användas för att köra test för att mäta systemets prestanda. För att kolla *test-coverage* och *code smells* osv. ska tredjepartstjänsten Code Climate användas. Detta för att automatisera processen att hitta duplicerad kod och på så sätt snabba på och underlätta för refaktorering.

5.3 Systembegränsningar, responstid, körtid

För att systemet inte ska uppfattas som långsamt och irriterande för användaren är målet att en sökning får ta max tre sekunder. Efter lätta undersökningar på hur liknande tjänster levererar sin sökning är tre sekunder en rimlig begränsning.

Programmet kommer endast att utvecklas för att kunna köras i UNIX-miljö. *Windows* prioriteras inte på grund av att ingen av de tänkta servrar där projektet kommer att köras är en Windows server. Windows-maskiner kommer fortfarande att kunna utnyttja tjänsten via en webbläsare.

Ansvaret för vad som laddas upp på servern ligger endast på användaren. Detta krav måste användaren acceptera för att få ta del av webbtjänsten. Alltså är det användarens skyldighet att läsa på om reglerna som gäller för att använda vår webbtjänst. Olagliga filer har systemadministratörerna rätt att ta bort.

5.4 Standarder, metoder och tredjeparts-APIer

Som standard skall GitHubs stilguide för Ruby, *JavaScript* och *CSS* användas. Den finns här.

För att hantera användares filer via Dropbox kommer deras *API* att användas. Detta genom den Ruby-klient som Dropbox tillhandahåller för att enkelt uppdatera, lista och skapa filer på användarens Dropbox-konto. På sikt skall även *Google*:s och *Box*:s API för filhantering undersökas för att se om det kan användas.

5.5 Systemmiljö, filer/filformat, input och output

Dropbox kommer användas som server för att underlätta hantering av filer. Därav kommer webbtjänsten kunna hantera de filtyper Dropbox stödjer. I dagsläget innebär detta i princip samtliga filer som går att lagra på en dators hårddisk. Inledningsvis kommer fokus ligga på att kunna presentera bilder, länkar, *pdf*-filer, ljud-filer samt videor. För att kunna redigera en fil måste den laddas ner från webbtjänsten till sin dator sen är det upp till användaren att själv hitta ett passande program för att redigera filen.