CHALMERS | 

GÖTEBORGS UNIVERSITET

SuperBooky

- modernt webbaserat bokföringsprogram för småföretag

Kandidatarbete inom Data- och Informationsteknik

DŽENAN BAŽDAREVIĆ
DANIEL CHINIQUY ENGSTRÖM
ISABELLE FRÖLICH
JAKOB CSÖRGEI GUSTAVSSON
ALEXANDRA LAZIĆ
VICTOR WIBECK

Institutionen för Data- och Informationsteknik
CHALMERS TEKNISKA HÖGSKOLA
GÖTEBORGS UNIVERSITET

Göteborg, Sverige, Juni 2015

Kandidatrapport 2015:02

SuperBooky
Modernt webbaserat bokföringsprogram för småföretag

Kandidatarbete inom Data- och Informationsteknik

DŽENAN BAŽDAREVIĆ
DANIEL CHINIQUY ENGSTRÖM
ISABELLE FRÖLICH
JAKOB CSÖRGEI GUSTAVSSON
ALEXANDRA LAZIĆ
VICTOR WIBECK

Institutionen för Data- och Informationsteknik
Chalmers tekniska Högskola
Göteborg, Sverige, Juni 2015

SuperBooky - Modernt webbaserat bokföringsprogram för småföretag
Kandidatarbete inom Data- och Informationsteknik
DŽENAN BAŽDAREVIĆ
DANIEL CHINIQUY ENGSTRÖM
ISABELLE FRÖLICH
JAKOB CSÖRGEI GUSTAVSSON
ALEXANDRA LAZIĆ
VICTOR WIBECK

Handledare: Alexander Sjösten, Göteborgs Universitet
Examinatorer: Arne Linde, Institutionen för Data- och Informationsteknik
Jan Skansholm, Institutionen för Data- och Informationsteknik

Kandidatrapport 2015:02
Institutionen för Data- och Informationsteknik
Chalmers tekniska högskola
SE-412 96 Göteborg
Sweden
Telephone +46 31 772 1000

Göteborg, Sverige 2015

Abstract

The aim of this report is to give the reader insight into the development of the web-based accounting application *SuperBooky*. Accounting is a complex task that many beginner entrepreneurs struggle with. The development of this application was therefore focused on making accounting as convenient and as easy as possible. The end product targets small enterprises and its functionality was designed with this in mind.

To make sure that the application was designed in a user-friendly way, polls were carried out among individuals with basic accounting knowledge. These polls were then used as a basis when the application was under development, to ensure that it was well-suited to the target audience.

This project was carried out as a bachelor's thesis for Chalmers University of Technology and University of Gothenburg during Spring 2015. This report is written in Swedish.

Sammanfattning

Den här rapporten syftar till att ge läsaren en bild av utvecklingen av det webbaserade bokföringsprogrammet *SuperBooky*. Eftersom bokföring är ett komplext förehavande som många nyblivna företagare har svårigheter med utvecklandes den här applikationen med fokus på att göra bokföring så lätthanterligt som möjligt. Slutprodukten riktar sig mot småföretag och dess funktionalitet är anpassad därefter.

För att säkerställa att programmet blev så användarvänligt som möjligt utfördes flera enkätundersökningar bland individer med grundläggande bokföringskunskap. Dessa enkäter användes sedan som underlag när programmet utformades för att säkerställa att det riktade sig mot den tänkta målgruppen. Senare utfördes även ytterligare användartester som verifierade att den färdiga produkten fungerade som den skulle.

Förord

Detta projekt utfördes som ett kandidatarbete för Chalmers tekniska högskola och Göteborgs Universitet under våren 2015. Omfattningen för denna kurs är 15 högskolepoäng vilket innefattar cirka 400h per student. Kandidatarbetet har utförts vid institutionen för Data- och Informationsteknik på Chalmers tekniska högskola, av studerande på Datateknik 300 hp, Informationsteknik 300 hp och Industriell Ekonomi 300 hp på Chalmers tekniska högskola samt Datavetenskapligt program 180 hp på Göteborgs Universitet.

Projektgruppen vill även passa på att tacka Alexander Sjösten som genom hela projektet har varit tillgänglig för att vägleda gruppen. Tack går även ut till enheten *Fackspråk och kommunikation* på Chalmers tekniska högskola för handledning samt hjälpsamma föreläsningar om såväl språk- som kommunkationsrelaterade punkter.

Innehåll

Ordlista

- Java Platform, Enterprise Edition (Java EE) - Den officiella Javaplattformen för att bygga större system som kräver kopplingar till databas, stöd för webb, med mera.
- Model-View-Controller (MVC) - Ett beprövat arkitekturmönster inom programutveckling som beskriver ett effektivt tillvägagångssätt för att strukturera större applikationer (Krasner och Pope 1988).
- Hypertext Transfer Protocol (HTTP) - Ett protokoll som används för att överföra HTML sidor över nätverk.
- Hypertext Transfer Protocol Secure (HTTPS) - En variant av HTTP med tillägg som till exempel kryptering av data vilket gör det till ett säkrare protokoll.
- Object-relational mapping (ORM) - Ett sätt att spara objekt från ett objektorienterat programmeringsspråk till en relationsdatabas och sedan hämta den med databasförfrågningar.
- Verifikation - Sammanställning av den information som krävs för att redovisa en ekonomisk händelse i ett företag.
- Debet - Namnet på en av de två kolumner som redovisar summor i en verifikation i dubbel bokföring. Summan av alla rader i debetkolumnen ska alltid vara den samma som summan av raderna i kreditkolumnen.
- Kredit - Namnet på en av de två kolumner som redovisar summor i en verifikation i dubbel bokföring. Summan av alla rader i kreditkolumnen ska alltid vara den samma som summan av raderna i debetkolumnen.
- Årsbokslut - Sammanställning av ett företags bokföring som görs vid varje årsskifte. Innehåller bland annat en balansrapport och en resultatrapport.
- Balansrapport - Redovisar ett företags sammanställda tillgångar och skulder vid en given tidpunkt.
- Resultatrapporter - Redovisar ett företags omsättning (intäkter och kostnader) under en given period.

1 Inledning

I följande avsnitt ges en bakgrund till projektet, följt av dess syfte och problemformulering, som redogör för varför applikationen som projektet ämnar skapa behövs. Därefter sätts ett antal avgränsningar för att projektet inte skall bli alltför komplext inom den tidsram och med de resurser som finns tillgängliga.

1.1 Bakgrund

I Sverige har alla företag bokföringsplikt vilket innebär att företag måste anteckna och redovisa samtliga ekonomiska händelser i företaget (Riksdag 1999). Detta kan göras för hand, och har gjorts så under lång tid, men kan också med fördel göras i ett bokföringsprogram.

Att starta upp ett nytt företag kan kännas överväldigande om personen ifråga inte har tidigare bokföringserfarenhet. Momsregistrering, skattesatser och bokföringsregler är bara några av de utmaningar som nybörjare inom bokföring ställs inför. Det är därför viktigt att ha tillgång till ett enkelt och användarvänligt bokföringsprogram som kan hjälpa användaren att komma igång så fort som möjligt.

Ett antal olika bokföringsprogram finns tillgängliga både för användning i webbläsaren och för att köpa och ladda ner, exempelvis Zervant (Zervant u.å.), alla dessa program har olika funktioner och användargränssnitt. Många av dessa program riktar sig till stora och medelstora företag och har funktioner som småföretag inte behöver, såsom import och export av varor. Det kan därför vara gynnsamt för nya och oerfarna företagare att det introduceras ett enklare bokföringsprogram.

1.2 Syfte

Syftet med projektet är att skapa ett användarvänligt webbaserat bokföringsprogram för småföretagare som tillgodoser alla de grundläggande bokförings- och faktureringsbehov som en sådan användare kan tänkas ha. Utöver detta avser projektet även att förenkla bokföringsprocessen genom att automatisera den så mycket som möjligt.

1.3 Problemformulering

Projektets fokus ligger på att skapa ett användarvänligt program. Målet är att göra programmet lätt att förstå och använda, samtidigt som det ska tillgodose alla de bokförings- och faktureringsbehov som en småföretagare kan tänkas ha. För att uppnå detta krävs vid framtagning av applikationen en medvetenhet om vilka funktioner en småföretagare använder mest i vardagen och hur dessa funktioner kan formges för att de skall kännas så intuitiva som möjligt. Problemet ligger därför i att ta reda på vilka funktioner som användaren helst vill använda och hur dessa kan presenteras på ett användarvänligt vis. För att applikationen skall särskilja sig från

andra bokföringsapplikationer och fylla de behov som finns på marknaden skall bokföringsprocessen automatiseras i största möjliga mån. Automatiseringen syftar till att få nybörjare att uppleva applikationen som särskilt användarvänlig. Problemet ligger i att göra detta på ett korrekt sätt utan att viktiga funktioner faller bort.

Bortsett från användarvänlighet finns även ett behov av en robust back-end som håller reda på de regler som finns inom bokföring. Problemet här ligger i att säkerställa att inga komplikationer uppstår i form av fel i bokföringen som kan kompromettera användaren, då detta, i värsta fall, kan resultera i böter eller fängelse om användaren döms för skattefusk (Riksdag 1962). Utöver detta är det även viktigt att programmet är säkert och inte kan exploateras av olika typer av hacker-attacker eftersom kritisk information om användarnas företag sparas i applikationens databas.

1.4 Avgränsningar

Programmet riktar sig mot småföretag och oerfarna bokförare. Följaktligen krävs det att programmet avgränsas på ett sådant sätt att funktionerna inte blir överväldigande och komplexa för användaren. Eftersom applikationen riktar sig mot småföretagare stödjer den enbart kontantmetoden, vilken enligt lag endast får användas om företaget årligen omsätter mindre än 3 miljoner kronor (Riksdag 1999). Kontantmetoden innebär att bokföringen görs samtidigt som företaget tar emot pengarna, medan alternativet, fakturametoden, kräver att bokföring sker när fakturor skrivs eller tas emot (BFN 2011). Kontantmetoden används istället för fakturametoden eftersom det är den enklare metoden att jobba med för småföretag, då företagaren endast behöver bokföra när pengarna sätts in på kontot och inte också när fakturan skickas ut. På grund av ovanstående anledningar kommer endast kontantmetoden att stödjas.

Applikationen tillhandahåller inte heller bokföringsregler för handel utanför Sveriges gränser, då import och export har speciella skatteregler beroende på land. En sådan implementation ligger utanför tidsramen för projektet och bortprioriteras därför. Det bedöms också att om användaren mot förmodan vill importera eller exportera har denne förmodligen redan en bra förståelse för bokföring och kan då istället använda ett program som är speciellt avsett för sådana behov. Slutligen tillhandahåller inte applikationen de mer avancerade rapporterna som ett större, men inte mindre, företag behöver för ett årsbokslut (Riksdag 1995).

2 Teori

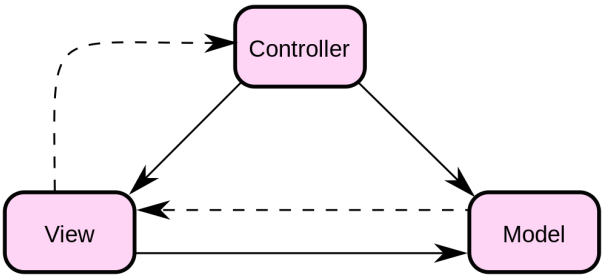
För att läsaren ska kunna ta del av denna rapport på ett tillfredsställande sätt presenteras nedan den teori som låg till grund för utvecklandet av applikationen. Först beskrivs de designmönster som finns integrerade i programmet för att hålla kodbasen hanterlig. Därefter redovisas de ramverk som varit speciellt viktiga under utvecklingen, bland annat Spring Framework och AngularJS. Vidare presenteras object-relational mapping (ORM), som var centralt i hantering av databasen och därefter presenteras generell teori om design och användarvänlighet för webbsidor. Efter detta följer ett kort avsnitt om säkerhet hos liknande applikationer och slutligen ges även en introduktion till ämnet bokföring, som är kärnan i applikationen.

2.1 Designmönster

För att undvika att ett mjukvaruprojekts kod blir svårhanterlig kan diverse designmönster användas. Genom att använda designmönster, såsom Model-view-controller som beskrivs nedan, inkapslas koden i mer hanterbara moduler. Dessutom kan systemet förbli hanterbart för utvecklarna i takt med att storleken på projektets källkod ökar (Martin 2000).

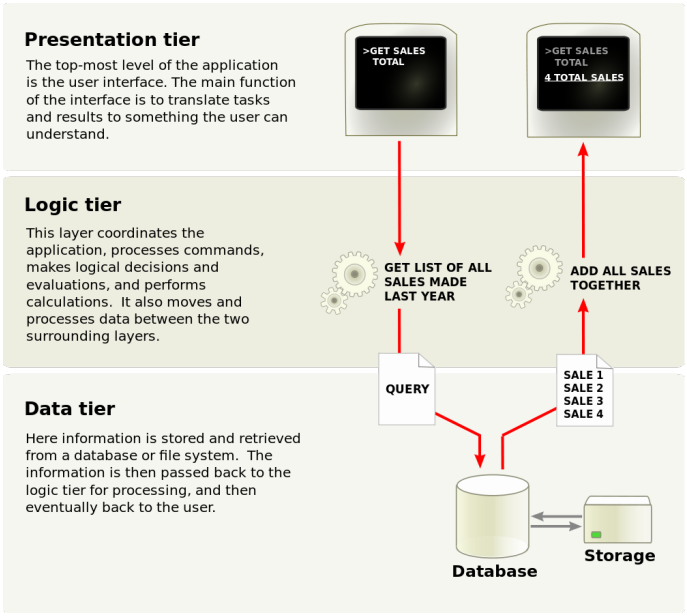
Model-View-Controller (MVC) är ett beprövat arkitekturmönster inom programutveckling som beskriver ett effektivt tillvägagångssätt för att strukturera större applikationer (Krasner och Pope 1988). I sin grund går det ut på att applikationen delas upp i tre distinkta delar: model, view och controller, eller modell, vy och kontroller. Modellen är representationen av datan i systemet, vyn presenterar modelldatan för användaren, och kontrollern hanterar inmatning från användaren för att visa den korrekta vyn.

En av de största styrkorna med MVC ligger i dess tredelade uppdelning. Genom att separera modellen från vyn kan ändringar göras i den ena komponenten utan att påverka den andra; det vill säga att så länge gränssnittet mellan de båda komponenterna hålls intakt kan ändringar i implementationen göras utan att kopplingarna i systemet förstörs. Samma resonemang kan appliceras på kopplingen mellan kontrollern och vyn, samt mellan kontrollern och modellen. Hur komponenterna sammankopplas kan ses i Figur 1.



Figur 1: Schema över MVC (Strannered 2010).

För applikationer som till exempel kräver att data sparas i en databas finns en variant av MVC som kallas *trelagrad arkitektur*. Detta arkitekturmönster har liksom MVC tre distinkta lager, dock med viss modifikation: presentation, logik och data. Datalagret tillhandahåller ett gränssnitt till en databas, som ofta används i större affärsapplikationer, och har ingen direkt motsvarighet i MVC. Presentationslagret är en kombination av vyn och kontrollern, där kontrollern hämtar data från logiklagret. Logiklagret i sin tur är analogt med modellen i MVC och kommunicerar med datalagret för att hämta och spara data i databasen.



Figur 2: Schema över trelagrad arkitektur (Bartledan 2009).

En av de största fördelarna med trelagrad arkitektur, liksom med all modulär arkitektur, är att implementationen av varje enskilt lager enkelt kan bytas ut mot en annan implementation (Meyer och Webb 2005). Detta åstadkoms genom att lagren endast kommunicerar med varandra genom fasta gränssnitt, vilket undangömmer implementationen från den som vill använda lagren, så kallad gränssnittsbaserad programmering (MSDN 1999).

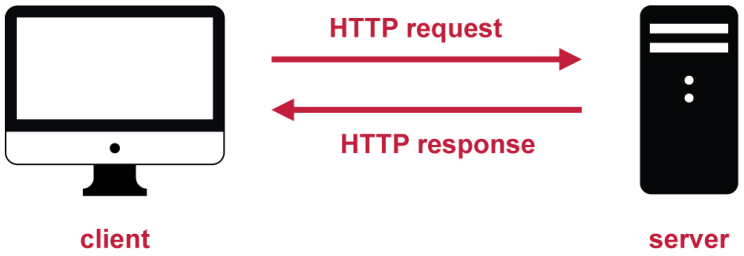
2.2 Object-relational mapping

Object-relational mapping (ORM) är en programmeringsteknik som bland annat gör att objekt i ett objekt-orienterat programmeringsspråk kan sparas ner till en databas och sedan hämtas ut med hjälp av databasförfrågningar. Detta gör att programmeraren inte behöver spendera onödig tid på att definiera databastabeller, när det är enklare att istället använda de redan skapade klasserna från domänmodellen. Till Java finns tillgängligt ramverk som tillsammans med en databaskopplare, till exempel Hibernate som är en av de mest kända, ger funktionalitet som åstadkommer just detta. Javaklasserna behöver då se ut på ett speciellt sätt: det skall finnas en konstruktor utan argument, samt att alla fält skall ha tillhörande metoder för åtkomst

och modifikation. Klasser som uppfyller dessa krav, samt taggas med annotationen `@Entity`, kallas kort och gott *entity*. Fälten i en entityklass kan taggas med annotationer såsom `@OneToMany` för att visa på en en-till-flera relation, eller `@OneToOne` för att visa på en en-till-en relation. Från dessa entities och relationer genererar sedan databaskopplaren automatiskt databastabeller och join-tabeller, vilket effektivt gör att programmeraren inte behöver skriva någon SQL-kod för att definiera databasens tabellstruktur.

2.3 Hypertext Transfer Protocol

Hypertext Transfer Protocol (HTTP) är ett applikationsprotokoll för hypermedia informationssystem (Fielding m. fl. 1999). HTTP är implementerat för klient- och serverprogram som kan kommunicera med varandra över webben genom att utbyta HTTP-meddelanden (Kurose och Ross 2013, ss. 123-125). Protokollet definierar strukturen på dessa meddelanden och hur meddelandet transporteras. Det finns två typer av HTTP-meddelanden: HTTP-förfrågningar och HTTP-svar. En kommunikationslänk skapas oftast genom att en användare begär en webbsida, varefter användarens webbläsare skickar en HTTP-förfrågan efter objekten till webbservern. När webbservern tar emot förfrågan skickas det begärda objektet tillbaka som ett HTTP-svar. En illustration på detta finns i figur 3.
Hypertext Transfer Protocol Secure (HTTPS) är en variant av HTTP. Ett tillägg som HTTPS har är kryptering av data, detta har inte HTTP. HTTPS krypterar all data som transporteras mellan klient- och serverprogramen, vilket förhindrar att något annat än sändaren och mottagaren förstår datan.



Figur 3: En illustration över HTTP-förfrågan (request) och svar (response).

2.4 Plattformar och ramverk

I dag finns många ramverk, plattformar och bibliotek som kan användas för att göra mjukvaruutveckling enklare och mer effektiv. Nedan följer en introduktion till några av dessa, som kan hjälpa till vid utvecklandet av en webbapplikation.

2.4.1 Java Platform, Enterprise Edition

Java Platform, Enterprise Edition (Java EE) är en plattform som utgörs av en samling teknologier gjorda för att bygga så kallade enterprise-applikationer såsom web-

bapplikationer. Applikationer som byggs med hjälp av Java EE karaktäriseras av att vara flerlagrade (presentation, affärslogik och persistens), skalbara och tillförlitliga (Jendrock m. fl. 2014). Java EE utökar funktionaliteten hos Javas standardversion genom att lägga till stöd för ORM, flerlagrade applikationer, databastransaktioner och webbapplikationer.

2.4.2 Spring Framework

Spring Framework, hädanefter refererat till som Spring, är i sin kärna ett ramverk som bygger på *Inversion of Control* (IoC) (Spring u.å.[a]). Detta innebär att programmeraren förser ramverket med moduler som ramverket kan utnyttja istället för att programmeraren själv anropar ramverket. I viss mån följer Spring alltså Hollywood-principen: *"Don't call us, we'll call you"*.

En annan viktig funktion Spring tillhandahåller är dess stöd för Dependency Injection. Dependency Injection innebär att programmeraren själv inte skapar objekt utan istället låter ett ramverk, såsom Spring, ta hand om varje objekts livscykel. På så sätt kan objekt enkelt injiceras i databasen utan att programmeraren själv instansierar objekten med Javas nyckelord `new`. Istället lagrar ramverket information om standardimplementationer av objekten, som sedan kan användas när de behövs. Detta ökar bland annat programmets testbarhet och gör att det enklare kan delas upp i lager (Fowler 2004).

Till skillnad från Java EE, som är en officiell basspecifikation för webbaserade applikationer, är Spring ett ramverk som är utvecklat av en tredjepartsutvecklare. Spring använder sig av Java EE-specifikationen men gör även vissa saker på sitt eget sätt, såsom den stora fokuseringen på IoC.

Spring Web MVC

Spring Web MVC är en komponent i Spring som gör det enkelt att skapa och kommunicera med en webbserver genom att tillhandahålla ett gränssnitt för att skicka och ta emot HTTP-förfrågningar. För att Spring ska veta vart det ska delegera inkommande förfrågningar taggas klasser i applikationen med annotationen `@Controller`. Därefter kan enskilda klassmetoder taggas med annotationen `@RequestMapping` och dess attribut `value` sättas till exempelvis `"/start"`. Detta kan se ut på följande vis: `@RequestMapping(value = "/start")`. Detta gör att alla förfrågningar som skickas till URL:en `"/start"` kommer att delegeras till just den klassens metod.

Spring har en inbyggd klass `(HttpMessageConverter)` som kan konvertera HTTP-meddelanden till Java-objekt och vice versa (Spring u.å.[c]). Genom att tagga en parameter i metoden med annotationen `@RequestBody` binds värdet i HTTP-förfrågan till metodens parameter. En liknande annotation (`@ResponseBody`) finns även för metodens returtyp, där typen översätts direkt till ett HTTP-svar.

Spring Data JPA

För att förenkla kommunikation mellan en applikation och dess databas tillhandahåller Spring komponenten *Spring Data JPA*. Genom att låta ett interface taggat

med annotationen `@Repository` ärva från `CrudRepository<T, ID>` kan programmeraren specificera egna databasförfrågningar utan att behöva skriva någon faktisk logik för dessa förfrågningar; logiken genereras automatiskt av Spring vid runtime. Exempelvis kan i ett interface `CustomerRepository` som ärver från `CrudRepository <Customer, Long>` deklareras följande metod i Listing 1, där Customer är en entity-klass.

Listing 1: Spring Data JPA-förfrågan för att söka efter ett Customer-objekt med vissa kriterier.

```
Customer findByNumberAndRegistrationDateBetween(int number, Date
    startDate, Date endDate);
```

Motsvarande pseudo-SQL-förfrågan kan ses i Listing 2. Detta förutsätter naturligtvis att klassen `Customer` innehåller fälten `number` och `registrationDate` av typerna `int` och `Date`.

Listing 2: Psuedo-SQL-förfrågan för findByNumberAndRegistrationDate-Between(int number, Date startDate, Date endDate).

```
SELECT *
FROM Customers c
WHERE number = c.number
AND c.registrationDate >= startDate
AND c.registrationDate <= endDate;
```

I större system kan det finnas flera hundra databasförfrågningar av olika slag. Istället för att skriva alla dessa för hand kan programmeraren med fördel definiera metoder i ett interface, enligt exemplet ovan, vilket tar avsevärt kortare tid. En ytterligare fördel är att programmeraren som konstruerar databasförfrågningsmetoderna inte behöver bry sig om den underliggande databasens syntax, då Spring sköter detta automatiskt. Detta medför att databasimplementationen kan bytas ut när som helst utan att en enda databasförfrågan behöver skrivas om.

2.4.3 Document Object Model

Document Object Model (DOM) är ett programmerings-API för dokument såsom HTML och XML (W3C 2010). API:ets struktur liknar ett träd, vars noder kallas objekt. DOM erbjuder programmerare möjlighet att bygga och ändra dokument. Varje webbsida skrivet i HTML representeras som en DOM, och som en följd av detta kan statiska sidor modifieras med hjälp av ett skriptspråk, som till exempel JavaScript.

2.4.4 JavaScript

JavaScript är ett lättolkat dynamiskt programmeringsspråk som främst används för webbprogrammering (Flanagan 2002, s. 1). De flesta moderna webbläsarna kan exekvera JavaScript genom en inbyggd JavaScript-motor.

När JavaScript används på klientsidan kan den ändra webbläsarens representation av statiska webbsidor. Detta leder till en bättre användarupplevelse eftersom webbsidan kan interagera med användaren på ett mer engagerande och interaktivt sätt (Goodman, Morrison och Eich 2007, s. 3).

2.4.5 RequireJS

RequireJS är ett verktyg skrivet i JavaScript och används mest för webbapplikationer. Dess huvudsakliga funktion är att läsa in JavaScript-filer och moduler (RequireJS u.å.[a]). Varje JavaScript-fil som är definierad i en HTML-fil läses in synkront, det vill säga stegvis. Vissa JavaScript-filer kan vara beroende av andra, varför hänsyn måste tas till ordningen av den synkrona inläsningen. Vid användning av många JavaScript-filer kan det vara svårt att hantera ordningen samt att få den väl strukturerad (RequireJS u.å.[b]). RequireJS löser dessa problem, och kan dessutom göra applikationen snabbare då den även erbjuder asynkron inläsning av JavaScript-filer.

2.4.6 AngularJS

AngularJS, eller bara Angular, är ett open-source JavaScript-ramverk vars arkitektur är baserad på MVC. Angular är skapad i tron att deklarativ programmering är bättre för att konstruera användargränsnitt, medan imperativ programmering är bättre för att implementera affärslogik (AngularJS u.å.[c]). En Angular-applikation initialiseras genom att kompilera en HTML-fil som kallas *template* eller en *mall*. En mall har en referens till en *modul* som startar Angular-applikationen. När mallen har kompilerats och byggt upp DOM:en visas för användaren en vy.

Moduler

Angular-komponenter definieras i en modul (AngularJS u.å.[d]), och kan till exempel innehålla konfigurationsinformation eller Angular-komponenter såsom Services eller Controllers. En modul kan även läsa in andra moduler. Eftersom en Angular-applikation inte har någon huvudmetod används moduler för att definiera hur applikationen ska initialiseras och struktureras.

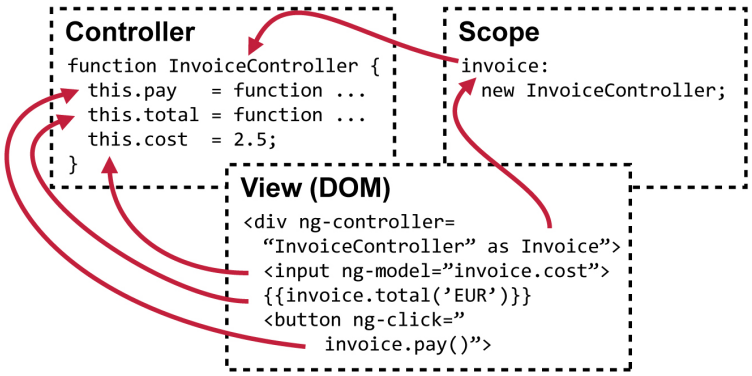
Controllers och Services

Både Controllers och Services är objekt skrivna i JavaScript som består av affärslogik, men har olika uppgifter. En Controller riktar sig till vyn och innehåller affärslogik som vyn behöver, medan en Service riktar sig åt andra komponenter (AngularJS u.å.[c]).

Scope

Applikationens modell kallas Scope. Vyn och Controllern har ingen direkt koppling mellan varandra utan tar istället hjälp av ett Scope (ibid.). Ett Scope lagrar alla variabler som delas mellan Vyn och Controllern, och så fort en variabel ändras

kommer variabeln att uppdateras både i Controllern och i Vyn (AngularJS u.å.[a]). Sammankopplingen mellan vyn och Controllern illustreras i figur 4.



Figur 4: En illustration på hur en vy (view) och en Controller är sammankopplade via ett Scope (AngularJS u.å.[a]).

Mallar

Mallar i Angular är skrivna i HTML med en utökad vokabulär. Den utökade vokabulären består av speciella element och attribut, med hjälp av vilka mallen kan visa information från till exempel en Controller (AngularJS u.å.[e]). De vanligaste typerna av element och attribut som används kallas *Directive* och *Markup*. Directive är en markör som kan läggas till i ett DOM-element (AngularJS u.å.[b]). Dessa markörer finns inbyggda i Angular, men fler kan implementeras om de inbyggda inte räcker till för ändamålet. Ett exempel på Directive finns i Listing 3. Markup börjar och slutar med två klammerparenteser och innehåller främst uttryck eller variabler från ett Scope.

Listing 3: Ett exempel på en Directive `ng-show` som används i en div.

```
<div ng-show="integer == 1">
    God dag!
</div>
```

En Angular-applikation innehåller oftast bara en mall. Genom att använda Directive `ngView` kan innehållet i HTML-elementet, kallat *partial*, ändras beroende på sökvägen.

2.4.7 Flying Saucer och Jsoup

Flying Saucer och *Jsoup* är två tredjeparts Javabibliotek som gör att PDF-filer enkelt kan genereras och exporteras från ett Javaprogram genom att ladda in XHTML-filer som sedan modifieras för att stämma överens med bibliotekens respektive syntax.

Flying Saucer

Flying Saucer erbjuder automatisk generering av PDF-filer från XML- och XHTML-filer. I genereringen används CSS som bestämmer utseendet på XHTML-sidorna.

Flying Saucer har stöd för stora delar av CSS 2.1-standard samt delar av 3.0-standarden (*Flying Saucer* 2013). Det innebär att de flesta moderna webbläsare visar XTHML-sidorna på samma sätt som de kommer ut som PDF-filer.

Jsoup
Jsoup gör att Java kan öppna HTML-filer och modifiera dem genom att till exempel ändra HTML-taggar eller injicera data på specifika platser. Jsoup klarar dock av många fler uppgifter än att bara ändra text i HTML-sidor, såsom att städa upp HTML-kod genom att ta bort både onödig syntax och skadlig kod som kan ha infekterat sidan. Dessutom kan Jsoup förhindra att skadlig kod, som skrivits in i inmatningsformulär, körs genom att hantera den som vanlig text.

2.5 Användarvänlighet

Att ha en bra design är ett sätt för utvecklare att sticka ut från konkurrenterna (Cooper m. fl. 2014). Genom att tänka på de potentiella användarna när produkten designas är det större chans att produkten når framgång. Det viktigaste för att användare ska uppfatta upplevelsen med webbsidan som positiv är att innehåll och tjänster uppför sig som förväntat, samt att sidan inte täcks av överflödig information (Gehrke och Turban 1999). För att en webbsida ska anses vara användbar krävs att den har ett grafiskt gränssnitt som tilltalar många användare (Cooper m. fl. 2014).

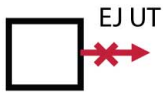
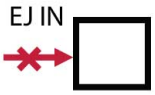
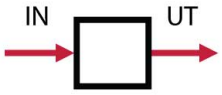
Ordet användarvänlighet byts ofta ut mot ordet användbarhet vilket härstammar från det engelska ordet *usability*, definierat som *"The extent to which a product can be used by specified users to achieve specified goals with effectiveness, efficiency, and satisfaction in a specified context of use."* (ISO 2010). Denna definition lägger vikten på att användaren kan utföra funktionen som efterfrågas på ett effektivt och tillfredsställande sätt.

Föregående definition används dock inte av alla. Nngroup beskriver usability på följande vis: *"Usability is a quality attribute that assesses how easy user interfaces are to use."* (NNGroup 2012). Gemensamt för dessa två definitioner är fokuset på hur effektivt produkten kan användas för att ge användaren det den vill ha.

2.6 Säkerhet

Det kan vara svårt att avgöra om ett system är säkert eller inte. Säkerhet definieras ofta av tre aspekter: sekretess, integritet och tillgänglighet (Stallings och Brown 2015, s. 33). Dessa koncept är på engelska kända som *the CIA (Confidentiality, Integrity, Availability) triad*. Utförligare beskrivning av de tre aspekterna av säkerhet finns i tabell 1.

Tabell 1: En beskrivning av Sekretess, Integritet och Tillgänglighet (Stallings och Brown 2015, s. 33)

Koncept	Beskrivning	Illustration
Sekretess	Säkerställer att information inte avslöjas för obehöriga individer.	
Integritet	Säkerstället att obehöriga individer inte kan modifiera information.	
Tillgänglighet	Säkerställer att information är tillgänglig för behöriga individer.	

2.6.1 Autentisering

Vissa delar av den information som finns i ett system skall endast kunna visas och modifieras av behöriga användare. Ett system måste därför kunna avgöra om en användare är behörig, vilket kan göras med hjälp av en autentiseringsprocess som består av två steg: identifikation och verifiering (Stallings och Brown 2015, s. 93). I identifikationssteget begär systemet att användaren identifierar sig genom att denne tillhandahåller autentiseringsinformation. Detta kan vara information som användaren vet (till exempel ett lösenord), något användaren äger (till exempel någon typ av säkerhetstoken, alltså en fysisk enhet) eller något användaren är (biometrisk autentisering) (ibid., s. 95). När användaren har angett autentiseringsinformationen överförs denna till verifikationssteget. I verifikationssteget jämför systemet den information som användaren angett med den information som systemet lagrat om användaren. Baserat på jämförelsen kan systemet antingen bevilja eller neka användaren tillträde (ibid., ss. 93, 95).

2.6.2 Säkerhetsproblem

I varje autentiseringssteg finns problem och risker. Om ett lösenord används som identifiering är idealet att lösenordet är hemligt, det vill säga att endast användaren känner till det. Trots detta finns det inkräktare som kan lyckas knäcka lösenord genom att utföra olika attacker. Det vanligaste scenariot är att inkräktarens dator lyckas gissa lösenordet på grund av att användaren har valt ett lösenord som inte är tillräckligt starkt (ibid., s. 95).

Ett annat problem är överföring av autentiseringsinformation. Internet är ett osäkert medium och det kan finnas individer som lyssnar på trafiken mellan klienten och servern (Kurose och Ross 2013, s. 698). Till följd av detta bör endast klienten och servern förstå informationen som skickas mellan dem. Dessutom måste kommunikationen ske med rätt klient eller rätt server. Om dessa egenskaper uppfylls har en *säker kommunikation* skapats. För att förhindra avlyssning kan meddelanden krypteras.

I verifikationssteget finns också vissa risker. Information behöver lagras i en databas för att kunna jämföras med autentiseringsinformationen. Informationen i databasen är känslig och kan därför vara ett mål för inkräktare. I händelsen att en inkräktare kommer åt information som finns i databasen är idealet att informationen skall vara oanvändbar. Genom att kryptera datan kan det göras svårare för inkräktare att tolka informationen (Stallings och Brown 2015, s. 86).

2.7 Bokföring

Enligt lag är varje enskild näringsidkare, handelsbolag och aktiebolag skyldiga att bokföra (BFN 2011, s. 15). En bokföring består av en mängd verifikationer som beskriver de olika transaktioner som skett inom ett företag. Varje inkomst och utgift skall redovisas med en verifikation. Dessa verifikationer skall i sin tur kunna redogöras för, i form av ett kvitto, en faktura eller någon annan form av fysiskt eller elektroniskt verifikat. Verifikationerna sammanställs vid varje årsslut i olika rapporter: resultatrapport och balansrapport. Det är dessa rapporter som utgör företagets årsbokslut.

2.7.1 Verifikation

En verifikation består av ett antal olika poster (minst två stycken), där varje post tillhör ett konto (ibid., s. 15). Varje post har två tillhörande kolumner: en för debet och en för kredit. Beroende på vilket konto som är associerat med posten kommer debet och kredit innebära olika saker. Summan av debet- och kreditkolumnerna skall i slutändan vara densamma; detta kallas för dubbel bokföring och görs för att säkerställa att bokföringen utförts på rätt sätt.

I den vanligaste kontoplanen, som kallas för BAS-kontoplanen, finns det totalt åtta olika kontoklasser. Majoriteten av dessa kan delas in i fyra olika kategorier (BAS 2015): kontoklass 1 är konton för företagets tillgångar, kontoklass 2 är för skulder, kontoklass 3 är intäkter, och kontoklass 4-8 är för olika kostnader. Beroende på vad som har hänt i företaget bokförs, som tidigare nämnt, posterna på antingen debet- eller kreditsidan. Också här kan kontoklasserna delas in i samma fyra grupper. I tabell 2 nedan följer en specifikation på i vilken kolumn ökning och minskning skall anges vid bokföring i de olika kontoklasserna.

Tabell 2: Exempel på en verifikation.

Kontoklass	**Debet**	**Kredit**
1	+	-
2	-	+
3	-	+
4-8	+	-

2.7.2 Bokföringsrapporter

Årsbokslutet består enligt lag av en resultatrapport, en balansrapport, noter och en förvaltningsberättelse (Riksdag 1995). Resultatrapporten skall omfatta företagets kostnader och intäkter vilket i BAS-kontoplanen innebär kontoklass 3 till 8. Balansrapporten i sin tur skall omfatta de resterande kontona som redovisar tillgångar och skulder, kontoklass 1 till 2. Det finns ingen fast mall att följa när dessa rapporter skall upprättas, men varje komplett bokföringsprogram har stöd för dem och utformar dem på liknande sätt. Gemensamma nämnare för balansrapporter är att de redovisar ingående och utgående balans för den aktuella perioden för varje konto som har använts i verifikationerna, samt att de redovisar det totala resultatet efter att kontona kvittas mot varandra. Resultatrapporter i sin tur redogör för företagets omsättning under den givna perioden.

2.7.3 Fakturor

En faktura är en elektronisk eller fysisk räkning som vanligen skickas till ett företags kunder för att informera dem om hur mycket de skall betala för en tjänst eller vara (Skatteverket 2010). En faktura skall enligt lag innehålla ett flertal saker. De viktigaste är: fakturans datum, datum för utfärdande, unikt fakturanummer, kundens och fordringsägarens namn, specifikation på antal och art av det fakturan avser, betalningssumma, båda parters registreringsnummer, båda parters adresser, momsskatt och moms som skall betalas (Riksdag 1994). Fakturor är en utav de handlingar som kan ligga till grund för en verifikation i bokförföringen och skall därför sparas i minst 7 år.

3 Metod

I detta avsnitt beskrivs hur arbetet genomfördes, samt vilka verktyg som användes för att åstadkomma resultatet och varför dessa valdes.

3.1 Arbetsmetod

För att systemet skulle bli korrekt utformat inleddes projektet med att grundläggande bokföringskunskaper inhämtades. Detta skedde genom en omfattande litteraturstudie av diverse bokföringsmaterial som samlades in under projektets början. Studiens främsta syfte var att ge korrekt struktur till systemet, varför materialet i första hand innehöll diverse bokföringsguider. Ytterligare relevant litteratur hämtades även från Skatteverkets och Sveriges Riksdags hemsidor för att undvika att lagar och regler skulle förbises. Om dessa bestämmelser inte beaktades kunde programmet bli opålitligt för användarna.

Parallellt med litteraturstudien undersöktes existerande bokföringsprogram för att få en bättre förståelse för svårigheter i att bokföra med dem. Bland annat testades bokföringsfunktionaliteten i program såsom Zervant och Visma eAccounting. Deras styrkor och svagheter diskuterades för att förstå hur bokföringsproceduren kunde förenklas. Detta gjordes eftersom det primära utvecklingsmålet, att göra ett användarvänligt program, senare skulle kunna realiseras.

Utöver detta utfördes även en enkätundersökning som delades ut till individer med begränsad erfarenhet av bokföring. Individerna valdes för att de ansågs representera möjliga användare av produkten. Resultatet användes sedan för att på bästa möjliga sätt kunna förstå användarens behov. Genom deras respons kunde gruppen även få reda på vilka svårigheter befintliga användare i dåläget hade med bokföring och anpassa programmet efter detta.

Gruppen delades upp i två arbetslag i ett försök att åstadkomma en tydlig ansvarsuppdelning. Det ena laget fokuserade på att göra efterforskningar om vad som behövdes i back-enden, medan det i andra laget genomförde liknande undersökningar för front-enden. Kunskaperna i respektive lag användes sedan för att konstruera och programmera systemet på bästa sätt. Meningen med att ha två arbetslag var att lättare kunna sätta gruppmedlemmarna i arbete utan att det blev för många personer som jobbade med samma problem, vilket lätt hade kunnat medföra oordning.

3.2 Front-end

JavaScript-ramverket AngularJS användes för att programmera front-enden. Angular valdes främst eftersom den erbjuder ett unikt sätt att uppdatera data i statiska HTML-sidor. Tack vare detta behöver inte mycket tid läggas på att skriva kod som manipulerar DOM:en, vilket kan vara svårt och krångligt. Applikationen delades

upp i flera moduler där varje modul hade en egen Angular-komponent, det vill säga att varje Service och Controller hade en egen modul.

För att undvika repeterande kod i HTML-sidor användes en mall. Varje innehållssida (partial) använde endast element som var unika för just den sidan, resten inkluderades från mallen. För att få bättre struktur på applikationen användes RequireJS som fungerade som en länk mellan HTML-filen och alla JavaScript-filer. Vid exekvering av RequireJS-filen initialiserades alla JavaScript-filer.

Att på egen hand utveckla en webbapplikation tar mycket tid, varför det under detta projekt beslutades att försöka spara in tid på att skapa designen genom att använda Twitters ramverk Bootstrap. Detta ramverk innehåller filer som är ämnade för att underlätta utvecklarens jobb; filer som skapar en designmall som är klara att användas. Mallen består bland annat av CSS-filer som användes för att definiera hur HTML-element ska se ut och bete sig. Utöver att använda ramverkets CSS skapades även en egen CSS-fil, `default.css`, som finjusterade några av sidans element och dess helhetsutseende.

3.3 Back-end

I detta avsnitt beskrivs först de ramverk och bibliotek som bygger upp back-enden, varför de valts och hur de används. Därefter redogörs kort för hur testningen i back-enden har genomförts.

3.3.1 Spring Framework

Back-enden använde sig av Spring Framework. Spring valdes först och främst eftersom det är enkelt att använda efter att det konfigurerats på ett tillfredsställande vis. Utöver enkelheten finns det i Spring många komponenter som kunde användas till att koppla ihop olika delar av systemet. Vissa komponenter utnyttjades mer än andra, såsom Spring Web MVC (Spring u.å.[b]), Spring Data JPA (Gierke, Darimont och Strobl 2015) och Spring IoC (Spring u.å.[a]).

En av de störst bidragande faktorerna till varför Spring valdes var att det är det mest populära ramverket för att utveckla webbapplikationer med Java (ZeroTurnaround 2015). Detta medför att det finns rikligt med dokumentation samt bra guider för att jobba med de olika komponenterna. Vidare hade vissa gruppmedlemmar redan erfarenhet med Spring, vilket det gjorde det lättare att komma igång i början; mer fokus kunde då läggas på att göra applikationen så användarvänlig som möjligt istället för att jobba med ramverket.

3.3.2 GlassFish

GlassFish version 4.1 är applikationsservern som valdes för att köra applikationens back-end. Den största anledningen till att GlassFish valdes var för att tidigare erfarenhet av programmet fanns, vilket minskade inlärningstiden.

3.3.3 Export av dokument

För att ett bokföringsprogram skall vara komplett krävs att det kan generera bokföringsrapporter. De bokföringsrapporter som krävs vid ett årsbokslut är, som tidigare nämnts, en balansrapport och en resultatrapport. Utöver dessa dokument är fakturor en central del både inom bokföring och i applikationen. Det bestämdes att dessa dokument lämpligast exporteras i PDF-format, för att göra de så portabla som möjligt. HTML-hanteraren Jsoup i kombination med PDF-skapararen Flying Saucer valdes för att skapa dessa.

Eftersom att Java och HTML redan användes i projektet ansågs det vara en smidig lösning. Av PDF-generatorerna till Java så var det Flying Saucer som valdes för inriktningen på just XHTML till PDF. För HTML-hanteraren så var Jsoup den mest kompletta av dem som övervägdes.

Första steget var att skapa olika mallar för faktura, balansrapport och resultatrapport och spara dessa som XHTML-filer. Mallarna innehöll en basstruktur samt utbytbara element som senare skulle ersättas med data från användaren. Därefter användes Jsoup för att läsa in mallarna i Javaapplikationen. Data från databasen laddades in och skrevs in där den behövdes, varefter Jsoup modifierade en lokal kopia av mallen i Java. Sista steget var att använda Flying Saucer för att exportera den lokala kopian till en PDF-fil.

3.3.4 Testning

För programmets testning utfördes en blandning av enhets- och integrationstester med hjälp av det inbyggda stödet för *JUnit* i Spring. Målet med testningen var att uppnå minst 75% kodtäckning, för att till en hög grad kunna programmatiskt säkerställa att systemet fungerade som det förväntades. Utöver den programmatiska testningen utfördes också användartester, beskrivna i avsnitt 3.7. Tillsammans förväntades de två typerna av testning kunna utgöra en god bas för att bedöma applikationens tillförlitlighet.

3.4 Säkerhet

För att uppnå en hög säkerhetsnivå för systemet inleddes arbetet med inlärning om vad säkerhet är och hur ett säkert system programmeras. Ett säkert system karaktäriseras av att det ofta utnyttjar en teknik som kallas defensiv programmering. En viktig regel i defensiv programmering är att aldrig anta något om hur systemet uppför sig; istället skall alla antaganden och feltillstånd kunna hanteras av systemet (Stallings och Brown 2015, s. 398). Ofta brukar programmerare lägga fokus på de stegen som genererar normal exekvering av programmet, med andra ord lyckade resultat (ibid., s. 398). För att undvika detta lades mycket fokus på att testa data som ledde till misslyckad exekvering av programmet, för att enklare hitta eventuella fel och buggar.

En strategi som användes var att aldrig lita på något som kommer från användaren, utan att istället validera all data innan den skickas vidare till systemet. För att göra arbetet mer effektivt lades mycket tid på inlärning om potentiella risker och hot, tekniker som används av inkräktare och vad som behövde göras för att minimera dessa risker. Testerna som genomfördes bestod av slumpvald indata eller vanliga misstag som kan komma från användaren.

3.5 Verktyg

För att underlätta den dagliga hanteringen av de två viktigaste resurserna i projektet, källkod och dokumentation, användes en rad olika verktyg. Motivering till varför de viktigaste av dessa valdes ges nedan.

3.5.1 Git

För att hantering av källkod skulle förenklas användes versionshanteringsverktyget *Git*. Samtidigt som det fanns flera fullgoda alternativ såsom CVS och SVN, valdes Git på grund av gruppens familjäritet med det samt att det av gruppen upplevdes som mycket enkelt att använda. Utöver detta tog det inte längre än en halvtimme att lära ut grundläggande operationer till de gruppmedlemmar som inte var vana vid Git, vilket var en stor fördel.

3.5.2 NetBeans

Att programmera större system i Java utan en *Integrated Development Environment* (IDE) blir onekligen i längden ohållbart. Enkla arbetsuppgifter, såsom att byta namn på en klass, hade med en vanlig textredigerare tagit överväldigande mycket längre tid och kraft än om en IDE används. Detta på grund av att programmeraren hade tvingats att hitta och ändra i alla referenser till den klassen som byttes namn på. Andra fördelar med att använda en IDE inkluderar generering av testrapporter, avlusningsstöd, syntaxrättning i realtid, med mera.

Med ovanstående anledningar i åtanke användes därför *NetBeans 8.0*, som är den senaste versionen av Oracles IDE för Java-applikationer. Just NetBeans valdes framför dess största konkurrenter Eclipse och IntelliJ IDEA på grund av dess väl utvecklade stöd för Java EE-applikationer. Utöver detta finns i NetBeans inbyggt stöd för GlassFish 4.0, vilket var den applikationsservern som ansågs enklast att arbeta med.

3.6 Design

För ett projekt med fokus på användarvänlighet är det viktigt med en tydlig definition på vad användarvänlighet är och hur den kan kvantifieras och mätas. Definitionen som används i rapporten är som följer: *”Användarvänlighet är den grad till vilken en användare kan förstå och använda en produkt för att nå sina mål på ett*

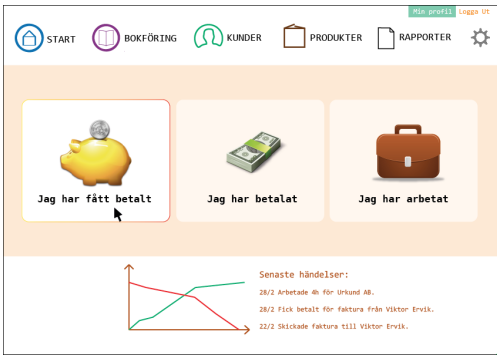
effektivt och tillfredsställande sätt”. Denna definition skiljer sig från ISO:s definition av användbarhet som nämns i avsnitt 2.5 genom att inkludera den grad till vilken en användare kan förstå produkten.

3.6.1 Processen

Arbetet med designen bestod till en början av ett designdokument där applikationens tänkta funktioner skrevs ut. Här låg fokuset på att få ner övergripande funktionalitet på papper, men inte gå in för djupt i detaljer. Designdokumentet bifogas som bilaga C i denna rapport.

En enkätundersökning utfördes för att få förståelse för de svårigheter som i nuläget finns för personer som bokför, och svaren användes för att leda arbetet i rätt riktning. Samtidigt undersöktes befintliga program på marknaden och en bedömning gjordes på dessa programs användarvänlighet och design, med syftet att använda deras bra idéer och samtidigt undvika deras misstag. Under arbetets och designprocessens gång fanns hela tiden fokus på den målgrupp som projektet önskade nå ut till.

Skisser på den tänkta grafiska designen skapades. Dessa skisser gjordes för hand på papper och gav front-endteamet en riktning att gå i vid implementationen. För programmets startsida skapades en prototyp i Adobe Illustrator där de viktigaste funktionerna tydligt specificerades, se figur 5.



Figur 5: Prototyp av startsidan.

Parallellt med denna designprocess implementerades funktionerna som beskrivits i designdokumentet. Allteftersom implementationen färdigställdes i front-enden stöttes det på saker som ännu inte tydliggjorts i designdokumentet. Det handlade till exempel om hur drop-downmenyer skulle bete sig och vad som skulle finnas under en inställningsmeny. När dessa oklarheter dök upp fördes en kort diskussion och ett beslut togs för att fastställa hur gränssnittselementen skulle se ut. När systemet började ta form blev vissa nya funktioner självklara, och andra behövde ändras. Genom denna metod arbetades en färdig design successivt fram.

3.6.2 Enkätundersökning

Som syftet lyder så var målet med projektet att skapa ett användarvänligt bokföringsprogram för småföretag. För att kunna säkerställa att programmet blev an-

vändarvänligt var det viktigt att veta vad användarna efterfrågar av programmet, alltså vilka funktioner som är vanligast och hur enkelt det är att genomföra dem. En enkätundersökning gjordes för att svara på dessa frågor och för att få ett underlag vid programmets design. Undersökningen riktade sig till personer med en tidigare grundläggande kunskap inom bokföring. Enkätundersökningen och svaren finns i bilaga D. Undersökningen skapades i Google Docs och skickades till personer som gruppmedlemmarna visste hade bokfört inom mindre företag tidigare.

3.7 Användartester

Mätningen av användarvänlighet av programmet skedde genom att intervjua programtestare i slutskedet för att få deras subjektiva bedömning, samt genom kvantitativa tester. Dessa tester såg ut på följande vis:

- Hur många steg/klick krävs för att utföra de vanligaste funktionerna i programmet?
- Hur lång tid tar det för en användare att lära sig att använda de vanligaste funktionerna i programmet?
- Hur svårt var detta att genomföra? (Skala 1-5)

Dessa tester gjordes också för bokföringsprogrammet Zervant, varefter resultaten jämfördes för att få en bild av graden av användarvänlighet. Programmet Zervant valdes då det är ett webbaserat bokföringsprogram med liknande funktioner som SuperBooky. Användartestet i dess helhet återfinns i bilaga E.

4 Implementation

I detta avsnitt beskrivs hur applikationen har konstruerats rent programmeringsmässigt: hur systemet ser ut, vilka beslut som har tagits, samt motivation till dessa. Avsnittet inleds med en beskrivning av själva systemet, hur det är tänkt att användas och hur det realiserar problemet som är tänkt att lösas - att bygga ett bokföringssystem. Därefter beskrivs själva arkitekturen, det vill säga hur applikationen har delats upp i lager - i detta fall i tre lager enligt trelagrad arkitektur. Vidare beskrivs även programmeringen bakom front-enden, samt hur kommunikation mellan front-enden och back-enden fungerar. Slutligen redogörs för hur säkerheten i systemet har implementerats.

4.1 Systembeskrivning

Nästan all bokföring kretsar kring verifikationer. En lista över alla verifikationer är även en lista över alla affärshändelser inom ett företag, vilka utgör kärnan i ett företags bokföring. Med detta i åtanke är det rimligt att anta att även de flesta datoriserade system för bokföring kretsar kring begreppet verifikation. Så är fallet även i detta system; många av de stora koncept som systemet innefattar är hjälpmedel till eller bara andra former av verifikationer. Utöver dessa koncept är alla de rapporter som bokföring kräver, såsom balansrapporter, årsbokslut och resultatrapporter i någon mån endast olika sätt att representera verifikationernas data.

Eftersom att systemet är webbaserat är det naturligt att det i centrum står en inloggad användare som brukar systemet, både ur en säkerhets- och en användarvänlighetssynpunkt. Till användaren finns kopplat allt som denne producerar i systemet: skapade verifikationer, produkter, tidsrapporter, bokföringsrapporter med mera.

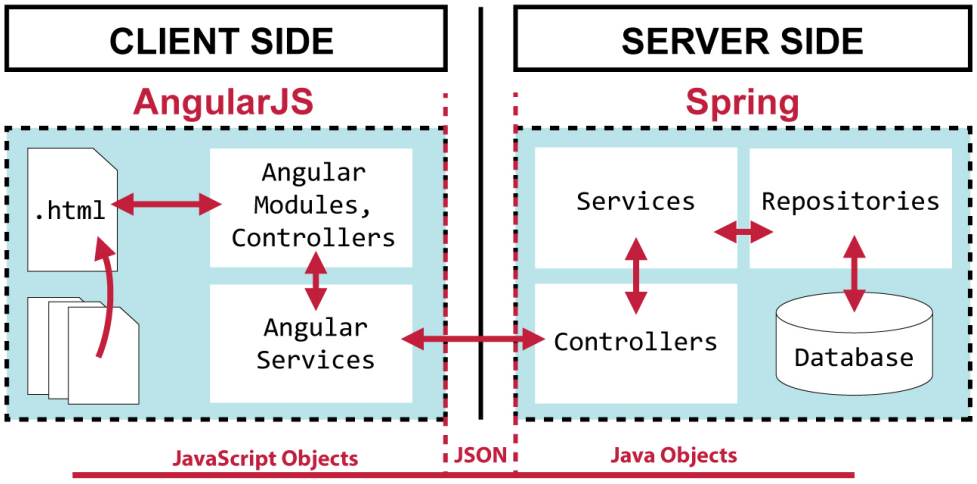
4.2 Back-end

Applikationen använder sig av trelagrad arkitektur. I stora drag är de tre delarna presentation, logik och persistens, och beskrivs nedan. En grov avbildning av dessa kan ses i figur 6, under den del med rubriken “Server side”.

4.2.1 Presentationslagret

Varje sökväg som webbapplikationen specifierar har en motsvarande Controller-klass taggad med annotationen `@Controller`, som tillhandahålls av Spring Web MVC. Detta innebär att till exempel sökvägen `/bookkeeping` har sin egen Controller, där all hantering av den manuella bokföringen sker.

Syftet med Controller-klasserna är att ta emot förfrågningar från användaren, validera dem, skicka vidare till logiklagret, och sedan skicka tillbaka resultatet till front-enden. Valideringen som sker i Controller-klasserna är främst till för att verifiera att användaren har rätt privilegier för att komma åt den resurs som begärts.



Figur 6: Översikt över systemarkitekturen.

Utöver denna verifiering fångas här triviala misstag från användaren upp, såsom att denne skrivit in bokstäver i ett inmatningsfält ämnat för telefonnummer. Mer komplicerad validering, som att bestämma huruvida en verifikation har giltiga värden eller ej, sker istället i andra delar av back-enden. Därför måste Controllern delegera vidare verifiering till logiklagret.

4.2.2 Logiklagret

Logiklagret i applikationen har hand om att manipulera och hämta information ur domänmodellen, den modell som beskriver domänen. Så kallade managerklasser används för att utföra mer komplicerade operationer på domänklasserna än det som kan hanteras av enkla metoder för åtkomst och modifikation av instansvariabler; exempelvis att räkna ut balansen för ett konto för en specifik användare. Till detta hör även operationer som kräver att flera olika domänklasser, som inte har någon relation till varandra, samarbetar för att producera ett resultat.

Domänmodellen i sig består av enkla entity-klasser som tillsammans beskriver domänen. Dessa klasser är tänkta att vara dumma, dvs att de inte har någon inneboende logik i sig, förutom att hämta och spara data. Logiken hanteras istället av managerklasserna, vilket gör domänmodellen till en så kallad *anemic domain model*. Fördelen med detta är att logiken helt separeras från datan, vilket både gör entityklasserna enklare att programmera och att de följaktligen uppfyller *Single Responsibility Principle*, alltså att varje klass har ett specifikt ansvar. I entityklassernas fall blir deras ansvar att hålla reda på sin data, medan managerklasserna håller reda på de möjliga operationerna på datan.

Logiklagret känner till konceptet att spara datan som domänmodellen innehåller, men det vet inte till vilket lagringsmedium datan sparas. Istället finns det ett gränssnitt till persistenslagret, som tillhandahåller metoder för att hämta data och spara data till något slags lagringsmedium. I detta fall är detta lagringsmediumet en databas.

4.2.3 Persistenslagret

Detta lager har hand om att spara data till databasen samt tillhandahålla metoder för att hämta denna datan. Detta åstadkoms genom att så kallade serviceklasser agerar som ett gränssnitt mellan logiklagret och persistenslagret, vilket gör att logiklagret inte behöver känna till vilket lagringsmedium som används. Det enda logiklagret vet när det begär att spara eller hämta data är att datan finns undansparad någonstans. Fördelen med denna uppdelning är att databasen enkelt kan bytas ut utan att något annat lager påverkas.

Närmast databasen finns repositoryklasser, som är en del av Spring Data JPA. Serviceklasserna, som tillhandahåller gränssnittet mot andra lager, gör i sin tur anrop till korrekt repository för att kunna uppfylla sina gränssnitt. Varje entity i logiklagret behöver sitt eget repository på grund av begränsningar i Spring Data JPA, vilket kräver att databasförfrågningsmetoder i en repositoryklass endast kan returnera objekt av en enda typ, deklarerat som en generisk parameter i klassignaturen.

4.3 Front-end

Angular-applikationen har två huvudfiler: `main.js` och `app.js`. Filen `main.js` använder RequireJS och innehåller destinationen till samtliga Javascript-filer samt huvud-modulen för applikationen. Detta kan även anges i en HTML-fil men denna metod valdes för att få en bättre struktur på systemet. Filen `app.js` innehåller huvud-modulen. Här angavs bland annat vilka Controllers och Services (komponentmoduler) som skall inkluderas i applikationen samt konfigurationer. En viktig konfiguration i modulen är att den kopplar ihop sökvägar med HTML-filer (partials) och Controllers. Listing 4 beskriver hur en sådan koppling programmeras.

Listing 4: Ett exempel på hur en sökväg sammankopplas till en HTML-fil samt en Controller.

```
when('/login', {
    templateUrl: 'login.html',
    controller: 'LoginCtrl'
}).
```

Varje HTML-sida har en unik Controller som huvudsakligen innehåller affärslogik som används av HTML-sidan.

En Service används främst som ett hjälpmedel och riktar sig åt alla angular-komponenter i applikationen. Dessutom kopplas en Service ihop med varje Controller som finns i back-end. Dessa Services innehåller främst förfrågningar, de vanligaste förfrågningarna är create, get, delete och update.

4.4 Kommunikation mellan front- och back-end

För att kommunicera mellan klienten och servern används *JavaScript Object Notation* (JSON). JSON är ett textbaserat språk som består av strukturerad data och

kan användas till exempel för överföring av data mellan klient och server (Crockford 2006). För att undvika att konstruera ett specifikt JSON-objekt för varje förfrågan av klienten skapades Java-objekt som representerar samtliga tabeller i databasen. Java-objekten översätts automatiskt till JSON-objekt när de skickas till klienten och vice versa, detta med hjälp av annotationerna `@ResponseBody` och `@RequestBody` som sätts före returtypen respektive parametern.

Det första servern gör när den mottar data från klienten är att validera den mottagna datan. Om ett fel hittas avbryts förfrågan direkt och ett JSON-meddelande skickas tillbaka till klienten. Om alla tester genererar positiva resultat skickas förfrågan vidare till en Service eller en Manager. Innan svaret skickas tillbaka till klienten kontrolleras även svaret från Servicen och Managern.

En HTTP-förfrågan från servern kan antingen lyckas eller misslyckas. Vid en misslyckad förfrågan kommer ett felmeddelande att skrivas ut i webbkonsolen. Om förfrågan istället lyckas kommer JSON-meddelandet att sparas som en variabel i ett Scope. Detta leder till att DOM kommer uppdateras och nya datan kommer att visas för användaren.

4.5 Spring Framework

Integralt i applikationen är användandet av Spring. Genom Spring Web MVC erhålls funktionalitet för att kommunicera med webbservern och front-enden, genom Spring Data JPA erhålls kommunikation med databasen, och genom den inbyggda IoC-funktionaliteten i Spring erhålls dependency injection och därmed också lösa kopplingar mellan systemets komponenter.

4.6 Säkerhet

I detta avsnitt beskrivs hur autentisering och kryptering av data har implementerats.

4.6.1 Autentisering

För att kunna logga in och börja bokföra måste användaren först gå igenom en autentiseringsprocess. För att kunna identifiera en användare måste denne tillhandahålla autentiseringsinformation som består utav en e-postadress samt ett lösenord. När autentiseringsinformationen skickas till servern jämförs den med informationen som finns i databasen.

Ett lösenord väljs utav användaren när denne registrerar sig. Det enda kravet på lösenordet är att det inte får ha färre än åtta tecken, detta för att förhindra korta lösenord som lättare kan testas fram. För att upplysa användaren om att det är viktigt att välja ett starkt lösenord får användaren en indikation på hur starkt det angivna lösenordet anses vara och kan baserat på den informationen förbättra sitt lösenord.

När användaren är inloggad sparas inloggningsinformationen i en session. Informationen består av användarens e-postadress, användarens användarnivå och ett sessions-id. Sessions-id tilldelas när användaren loggar in och sparas förutom i sessionen även i databasen. Varje gång en förfrågan skickas till servern valideras sessionen. Detta görs genom att jämföra sessions-id med det som finns i databasen. Om valideringen misslyckas töms all data från sessionen, varpå användaren loggas ut och nekas tillträde. Varje klient sparar en lokal kopia av sessionen, detta för att kunna ge eller neka tillträde till vissa sökvägar. Varje sökväg har därför konfigurerats med variabeln `auth` och en funktion `AuthHandler` för de sidor som kräver auktorisering. Ett exempel på hur sådan kod ser ut finns i Listing 5.

Listing 5: Ett exempel på hur autentisering kan läggas till en sökväg.

```
when('/edituser', {
    templateUrl: 'private/edituser.html',
    controller: 'EditUserCtrl',
    auth: USER_LEVELS.user,
    resolve: {
        auth: ['AuthHandler', function(AuthHandler) {
                return AuthHandler.promise();
        }]
    }
}).
```

Variabeln `auth` anger vilken användarnivå användaren måste ha för att kunna se en sida. Det finns tre typer av användarnivåer i klient-sidan: `all`, `guest` och `user`. Sidor som har användarnivån `all` kan nås av alla sorters användare, sidor med användarnivån `guest` kan endast nås av användare som inte är inloggade, medan `user` kräver inloggning.

Processen för att komma åt en sökväg sker i två steg. Det första steget jämför om den lokala kopian av sessionen har samma användarnivå som den nya sökvägens `auth`. Om jämförelsen inte stämmer nekas tillträde och användaren omdirigeras till en lämplig sida. I det andra steget försöker applikationen läsa in den angivna sökvägen. Sidor som kräver auktorisering använder funktionen `AuthHandler` som har placerats inne i en `resolve`, vilket förhindrar att sidan visas tills funktionen har exekverats. Syftet med `AuthHandler` är att se till att den lokala kopian av sessionen är giltig. När `AuthHandler` exekveras kommer en förfrågan skickas till servern efter en ny kopia av sessionen. Den nya kopian kommer att jämföras med den gamla, och baserat på resultatet av jämförelsen kan användaren antingen beviljas eller nekas tillträde till sidan.

4.6.2 Kryptering av data

Applikationen stödjer två kommunikationsprotokoll: HTTP och HTTPS. Vid användning av HTTP visas en varningsruta på inloggningssidan som notifierar användaren om vilka protokoll som finns att användas. Användaren kan själv avgöra om den vill kommunicera med servern över en säker kanal eller inte.

Känslig data i databasen såsom lösenord krypteras. Datan krypteras med en hash-funktion som är en enkelriktad algoritm. *Secure Hash Algorithm* (SHA) är samling av hash-funktioner som anses vara tillräckligt säkra, speciellt SHA-256 och SHA-512 (Stallings och Brown 2015, s. 73-74). Därför används SHA-256 i applikationen.

5 Resultat

I detta kapitel redovisas projektets slutgiltiga resultat. Först ges en sammanställning av enkätundersökningen och användartesterna, sedan visas det hur väl koden testats, och till sist visas applikationens användargränssnitt.

5.1 Enkätundersökning

Tio personer svarade på enkäten, vissa med mycket utförliga svar och andra med kortare svar. Tio personer tros inte vara tillräckligt för att dra några stora slutsatser om vad marknaden som helhet kräver, men det ger åtminstone en riktning att gå i som inte baseras på gissningar.

De som svarade på enkäten var uteslutande i åldern 18-35 och hade i 8 fall av 10 arbetat med bokföring inom en ideell förening i 12-18 månader. Det vanligast förekommande programmet var Visma eAccounting med 5 användare.

I undersökningen ställdes frågan: Vilka tre funktioner är viktigast i ett bokföringsprogram? Svaren rangordnas nedan:

1. Det ska vara enkelt att använda.
2. Man ska lätt kunna se över företagets ekonomi och de olika kontonas balans.
3. Man ska enkelt kunna korrigera fel i sin bokföring.

Dessa svar ger legitimitet åt projektets fokus på användarvänlighet och enkelhet i design. Många som svarade skrev att det program de använde inte hade någon bra funktion för kontoanalys, dvs förmågan att se över samtliga kontons in- och utgifter, varför denna funktion implementerats i arbetet. Genom att skapa och evaluera denna enkät blev de potentiella användarnas mål tydligare.

5.2 Användartester

För att avgöra hur väl projektet uppnått målet om ett användarvänligt program utfördes användartester. Användartesterna utfördes med flertalet personer som alla hade varierande bokföringskunskaper. Resultatet av testerna återfinns i bilaga E.

Genom att be personer som tidigare bokfört att utföra ett visst antal bestämda testfall i SuperBooky och Zervant gavs en bra och realistisk återkoppling. Under testningens gång iakttogs testarna noggrannt, vilket gjorde att det fanns stor möjlighet för direkt feedback och därmed chans för utvecklarna att bilda nya tankar kring programmet. Enligt (Cooper m. fl. 2014) är det viktigt att utvecklare är medvetna om de potentiella användarnas lång- och kortsiktiga mål för att kunna skapa en service som täcker användarnas krav.

Det utfördes 9 olika testfall på respektive applikation (enligt avsnitt 3.7). Dessa tester utfördes av 7 testpersoner som alla tidigare sysslat med någon form av bokföring. Testerna även eventuella kommentarer från testarna dokumenterades noga. Genom

att sitta bredvid och se utomstående personer testa applikationen var det lätt att se om det fanns brister i funktionaliteten.

För att få en överblick över resultatet sammanställdes alla testpersoners resultat och medelvärden togs fram över varje testfall för båda applikationerna. Det blev på detta vis lättare att se de olika testkriterierna: via vilken applikation testaren kunde utföra testfallen snabbast, vilken applikation som krävde minst antal klick för att utföra uppgiften och vilken applikation de slutligen gav bäst betyg. De olika testfallen visade att SuperBooky fungerade bättre än Zervant på 6 av 9 testfall. En generell kommentar som gavs efter avslutade tester var att det verkade som att SuperBooky var utformat för att vara enkelt, men att applikationen kanske inte räcker till för företag som kräver mer avancerade bokföringsfunktioner.

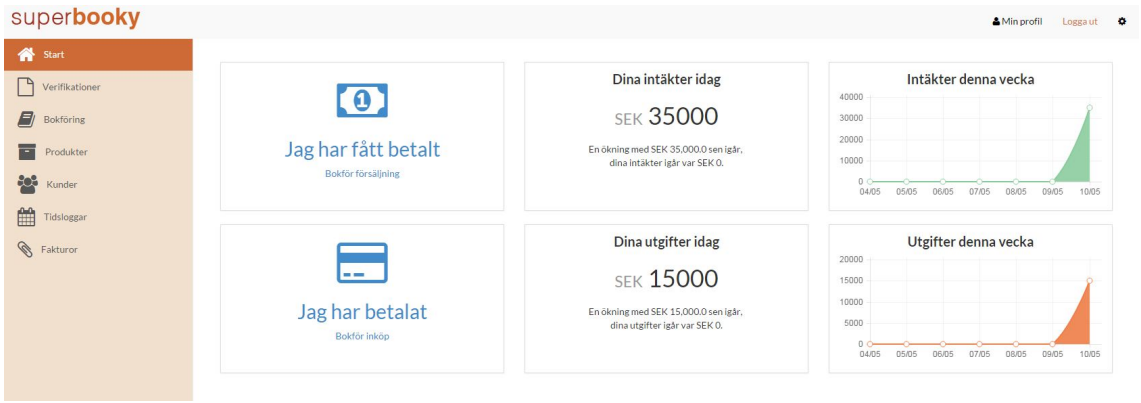
5.3 Testning av kod

För att testa koden sattes 75% kodtäckning som ett mål att uppnå för väl testad kod, vilket lyckades. Just 75% valdes då det enligt Steve Cornell är onödigt tidskrävande att sträva efter kodtäckning över 70-80% (Cornell 2013). Resultaten av testerna som genomfördes med hjälp av *Java Code Coverage Library* (JaCoCo).

I körningen av JaCoCo för kodtäckningen valdes att exkludera en rad av paketen, då de ansågs vara irrelevanta för ändamålet. Tanken bakom att ta reda på systemets kodtäckning var att få ut ett mått på systemets tillförlitlighet, varför klasser såsom de i domänmodellen inte var meningsfulla att testa specifikt eftersom de enbart innehöll metoder för att komma åt och modifiera instansvariabler. Vidare exkluderades paket såsom de för konfigurationen av Spring, då dessa testades tillräckligt genom att exekveringen utfördes utan felmeddelanden.

5.4 Användargränssnitt

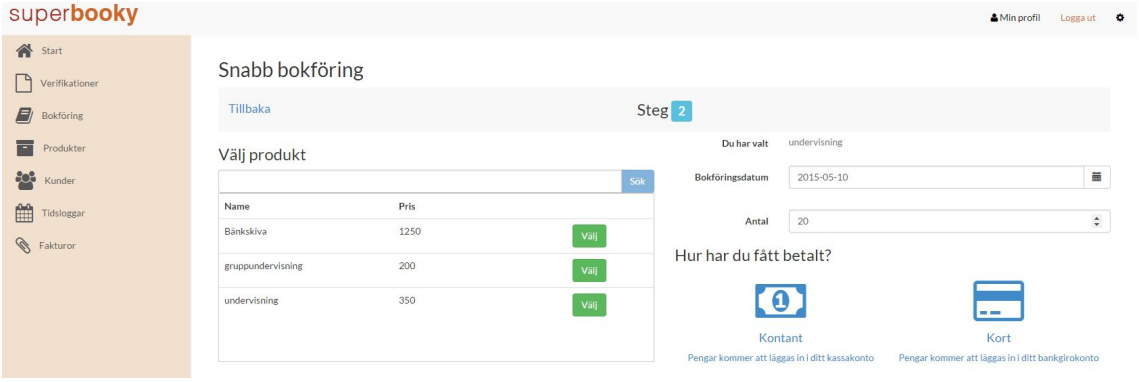
Programmets nuvarande användargränssnitt och den första prototypen som tidigare visats i figur 5 ser vid en första anblick väldigt olika ut, men många av funktionerna kvarstår. Menyn har flyttats från överst på skärmen till vänster och har nu även knappar för “Fakturor”, “Verifikationer” och “Tidsloggar”.



Figur 7: Startsidan

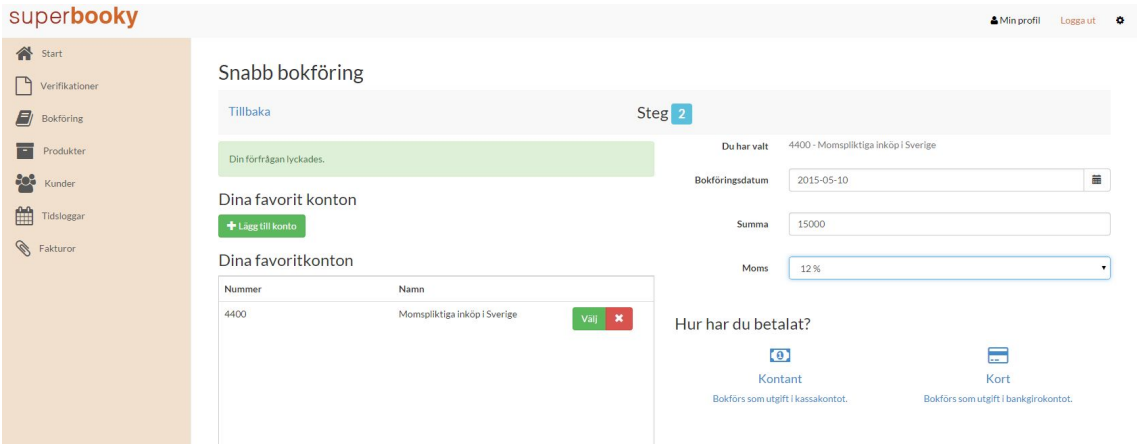
Snabbknappen “Jag har arbetat” har ersatts med sidan “Tidsloggar” i menyn till vänster. Denna sida innehåller den funktionalitet som som knappen "Jag har arbetat"var tänkt att ha, och ytterligare funktioner för hantering av tidsloggar.

Funktionen “Senaste händelser” har ersatts av de två sektionerna “Dina intäkter idag” och “Dina utgifter idag”. Diagrammet har i linje med föregående ändring delats upp i två diagram för att visa intäkter och utgifter för denna vecka.



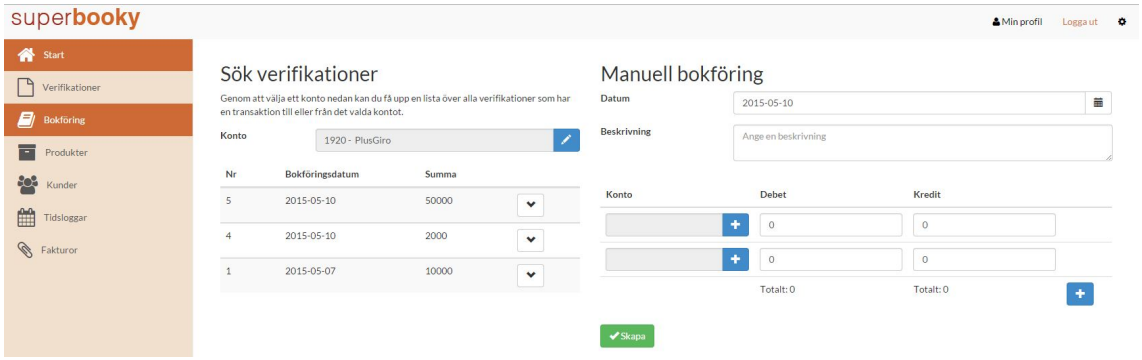
Figur 8: Snabb bokföring kopplad till knappen ”Jag har fått betalt”

Knapparna “Jag har fått betalt” och “Jag har betalat” i figur 7 leder in användaren till respektive sida för snabb bokföring. Om användaren exempelvis trycker på “Jag har fått betalt” hamnar användaren på sidan i figur 8. Här får användaren välja en produkt och antal enheter av den produkten som denne fått betalt för, vilket datum pengarna togs emot och om betalningen skedde med kort eller kontant. När användaren lagt in all relevant information tas användaren till en kontrollsida där den genererade bokföringsposten får godkännas innan den bokförs.



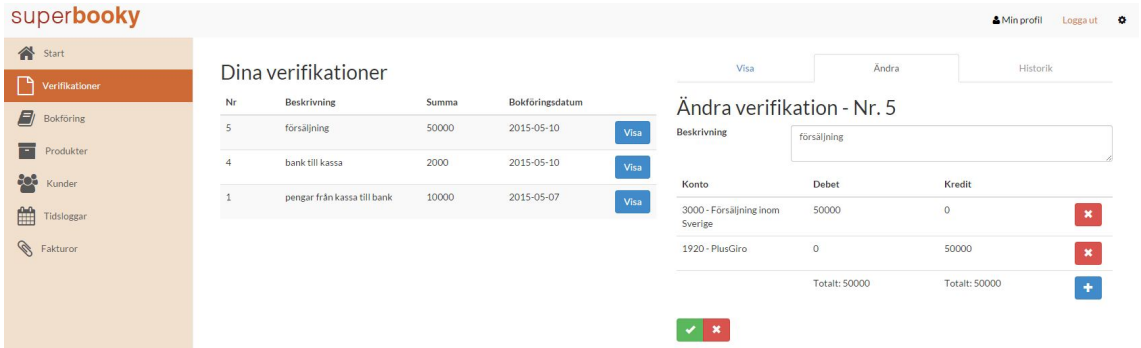
Figur 9: Snabb bokföring kopplad till knappen “Jag har betalat”

Om användaren trycker på knappen “Jag har betalat” leds användaren till sidan som visas i figur 9. På denna sida finns en lista med användarens “favoritkonton” som användaren själv lägger till med hjälp av knappen “Lägg till konto”. När användaren sedan valt det berörda kontot får användaren välja bokföringsdatum, summa, momssats och betalningssätt. Därefter tas användaren till en kontrollsida där den genererade bokföringsposten får godkännas innan den bokförs.

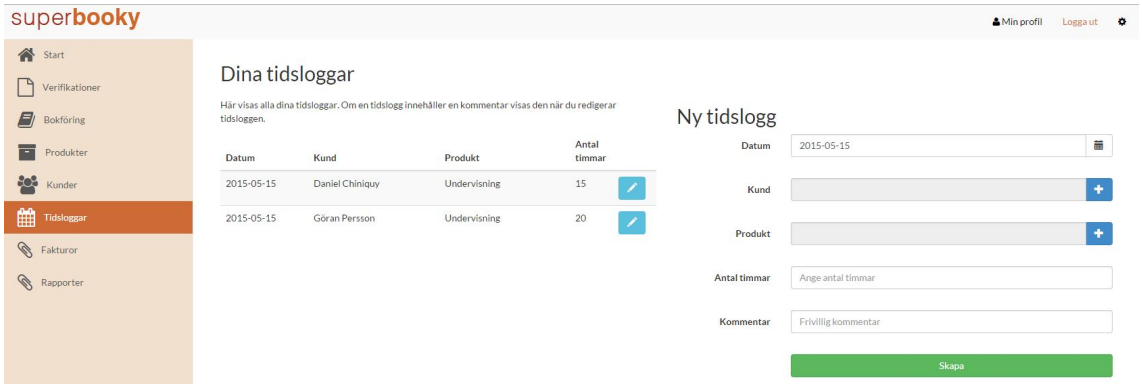


Figur 10: Bokföring

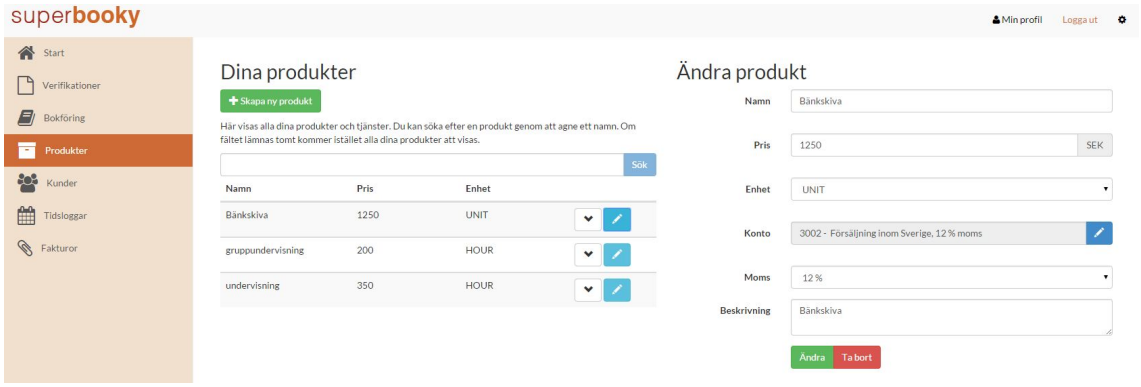
Om användaren vill bokföra annat än det som ingår i snabbknapparna "Jag har betalat" och "Jag har fått betalt" kan hon använda sidan "Bokföring" som visas i figur 10. Här kan användaren manuellt göra nya verifikationer genom att ange konto, datum samt debet- och kreditposter. Till vänster på sidan kan användaren ange ett konto, varefter hon visas verifikationer som rör det kontot ordnat efter datum nedanför.



Figur 11: Verifikationer



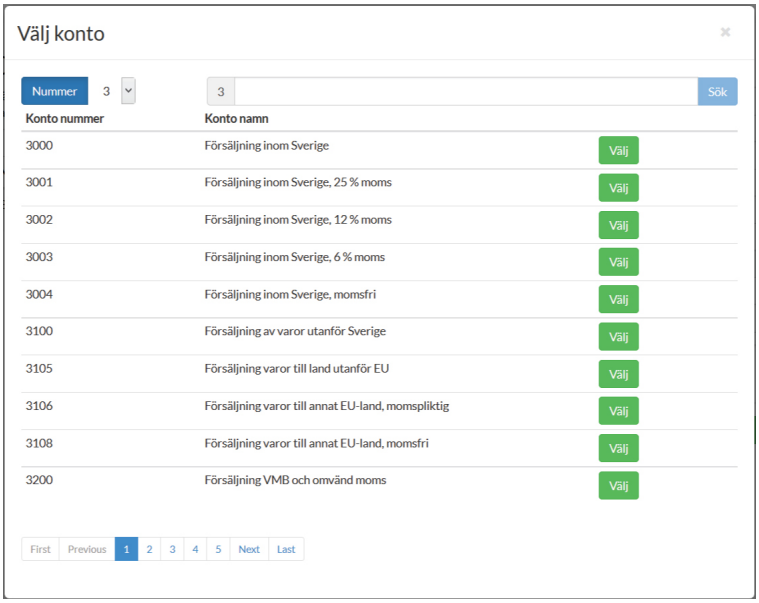
Figur 12: Tidsloggar



Figur 13: Produkter

Sidan “Verifikationer” som visas i figur 11 är en sida med en tabell på vänster sida som visar samtliga verifikationer för användaren med kort information om dessa. När knappen “Visa” trycks visas en mer detaljerad beskrivning till höger på skärmen, med flikar för att ändra verifikationen och för att visa ändringshistoriken för verifikationen.

Sidorna “Kunder”, “Tidsloggar”, “Produkter” och “Fakturor” beter sig alla på liknande sätt och visas ovan i figur 13. Sidorna är tvådelade med dels en tabell till vänster där de befintliga objekten visas, och dels en sida till höger med funktioner för att lägga till, redigera eller ta bort objekt. När en användare behöver lägga till ett konto för en produkt dyker en meny (Figur 14) upp mitt i skärmen, där en lista med samtliga konton tillsammans med en sökfunktion visas.

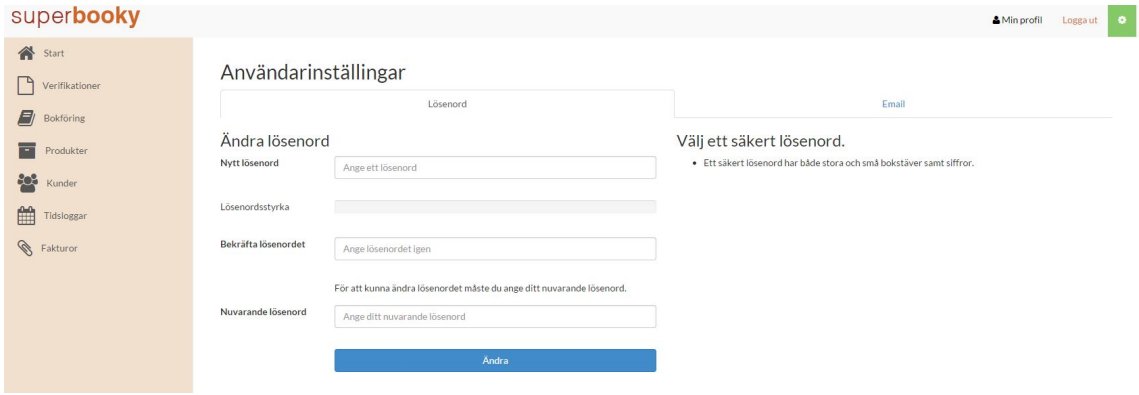


Figur 14: Meny vid kontoval

Längst upp till höger i applikationen finns en kugghjuls-ikon som tar användaren till sidan som visas i figur 15. Här kan användaren ändra lösenord och e-postadress. Lösenordsstyrkan för användarens nya lösenord visas med en förloppsmätare.

Vid byte av e-postadress tillämpas tvåvägsautentisering. När användaren har angivit en ny e-postadress, angivit sitt lösenord och klickat på knappen "Ändra"kommer en

ny ruta att öppnas. Rutan informerar användaren om att en kod har skickats till användarens gamla e-postadress. När användaren fyller i den korrekta koden kommer e-postadressen att ändras.



Figur 15: Användarinställningar

Hela applikationen kan i skrivande stund ses på `http://alexandralazic.se:8080/bokforing/`.

6 Diskussion

De delar som hittills täckts i rapporten har främst behandlat implementation och dess resultat. Utöver detta finns en mängd beslut som tagits, exempelvis angående säkerhet, struktur och design. I detta kapitel följer därför en reflektion av vad som gjorts, vad som valts bort samt vad som skulle kunna gjorts annorlunda.

6.1 Förarbete

Innan projektet påbörjades hade gruppen ingen eller relativt lite kunskap om bokföring. Därför var de tidiga domänmodellerna orealistiska och ohållbara. De flesta av de koncept som introducerades då slopades eller förändrades drastiskt vid ett senare skede.

6.2 Implementation

Den resulterande implementationen av back-enden blev i stora drag som planerad. Tanken var från början att back-enden skulle innehålla den funktionalitet som front-enden krävde, samt att den skulle vara säker och tillförlitlig. Kraven på säkerhet uppfylldes delvis av det inbyggda säkerhetsstödet i Spring, och delvis av defensiv programmering. Tillförlitlighet föll ut som en produkt av de rigorösa och omfattande tester som systemet innehåller. Dessa tester säkerställer att systemets mest kritiska delar fungerar som förväntat. Med detta sagt kunde naturligtvis fler och mer ingående tester ha implementerats för att ytterligare öka systemets tillförlitlighet.

Då fokus i systemet låg på de tidigare nämnda kriterierna har vissa kodprinciper ej prioriterats. Exempel på en sådan kodprincip är en väldefinierad paketstruktur som gömmer undan funktionalitet för moduler som inte bör ha tillgång till den, vilket i Java kan åstadkommas genom att använda paketsynlighet på klasser. Istället bygger paketstrukturen endast på att sortera klasser så att de är enkla att hitta för programmeraren. Denna princip bortprioriterades, utöver tidigare nämnda anledningar, på grund av systemets ringa storlek och gruppens nära samarbete, vilket minimerade risken att moduler användes på fel sätt.

6.3 Användargränssnitt

Användargränssnittet som projektet resulterade i ser annorlunda ut än hur det var tänkt att designats ifrån början. Många av de funktioner som var tilltänkta finns fortfarande kvar men har ändrat utseende och även funktionalitet allt eftersom arbetet fortskridit. Den första och mest fastställda tanken kring gränssnittet var att det skulle vara användarvänligt även för användare som ännu inte är vana att bokföra. Det viktigaste för projektgruppen var att det skulle vara enkelt för användaren att förstå vad som ska bokföras och var, utan att överväldiga användaren med information. För att åstadkomma detta skapades en idé, som nämnts i tidigare kapitel, om

att ha knappar på startsidan som täcker ”Snabb bokföring” (enligt figur 5 i kapitel 3.6). Genom att låta användaren tänka på om hon har betalat eller fått betalt kommer det, enligt projektgruppen, bli lättare för nybörjare att komma igång med bokföringen.

Under arbetet med applikationen insågs dock att vissa funktioner behövde utvecklas ytterligare för att programmet skulle platsa inom ramen för vad som förväntas av bokföringsprogram. Några ändringarna som behövde göras gällde huvudmenyn. Den första tanken var att ha en väldigt simpel topp-meny bestående av alla de sidor som ämnades läggas in. Idén bakom detta var att låta användaren få en överblick samtidigt som hon kan se vart hon navigerat och inte tappa bort sig i ett hav av länkar. Slutresultatet som utformades innehåller vissa ändringar i jämförelse med den ursprungliga idén, varav de mest signifikanta är att huvudmenyn flyttades till vänster sida, samt att den snabba bokföringen reducerades till två knappar istället för tre. Att huvudmenyn flyttades berodde främst på att det under implementationen insågs att det skulle behövas fler länkar än de som var tänkte att ha i topp-menyn. Om topp-menyn inte flyttats hade skärmutrymmet blivit ihoptryckt, vilket hade kunnat orsaka problem framförallt på mindre skärmar.

Den slutgiltiga versionen av den snabba bokföringen implementerades också med en del ändringar, jämfört med urspringsidén. Den största ändringen var gällande funktionaliteten och genomfördes eftersom projektgruppen insåg att det inte var nödvändigt att låta knappen ”Jag har arbetat” vara en så stor del av startsidan. Detta då det kändes mer intuitivt att koppla ihop denna funktionen direkt till ”Tidsloggar” och låta användaren lägga till sina arbetade timmar där. På detta sätt kunde startsidans fokus ytterligare riktas mot det som var ett utav huvudsyftena med projektet: att inte ge användaren för mycket information så fort hon loggar in i applikationen. Den första prototypen, som kan ses i figur 5, hade även ett översiktsdiagram för att vissa senaste veckans intäkter och utgifter. Att ha ett gemensamt diagram med en intäkt- och utgiftgraf på visade sig dock under implementationen vara otydligt. Det insågs att om en användare har samma intäkter och utgifter vid ett specifikt datum kommer graferna ligga precis över varandra. Det blir då svårt för användaren att få en tydlig översikt över företagets ekonomi, vilket var syftet med denna funktion.

6.4 Användartester

Att användartesterna (avsnitt 5.2) visar att SuperBookys funktioner fungerar bra var ett positivt resultat för projektet. Insikten att förstagångsanvändare lyckas utföra de flesta testfallen utan problem visar på att ett av syftena med projektet, att skapa ett användarvänligt bokföringssystem, har uppnåtts. Även kommentarerna som vissa av testpersonerna gav då de nämnde att SuperBooky verkade utformat för att vara enkelt, men att applikationen möjligtvis inte räcker till för större företags användningsområden tolkades som positivt då applikationen designats med syfte att främst vara för mindre företag.

Trots de positiva resultaten är projektgruppen medveten om att utförligare tester

och förbättringar av applikationen krävs för att säkerställa att produkten är genomarbetad och dess möjliga brister eliminerade. Dessa tester skulle behöva täcka en större del av applikationen samt utföras med fler personer för ett mer kvalitativt resultat. Ytterligare tester minskar också risken att bias påverkar resultatet.

6.5 Omvärldsanalys

De ekonomiska vinstmöjligheterna för bokföringsprogram på marknaden är i dagsläget begränsade. Detta beror dels på att applikationen skulle kunna plagieras, men även på att många bokföringsprogram redan finns ute på marknaden. Men om det förutsätts att rätt nisch har valts på applikationen och om korrekta antaganden har gjorts om att användare med begränsad bokföringskunskap saknar ett lättanvänt bokföringssystem, finns det ändå goda möjligheter att fylla ett befintligt behov på marknaden. Detta i enlighet med syftet där automatiseringen av funktioner antas locka fler nybörjaranvändare att välja ett program som SuperBooky vilket i sin tur medför vinstmöjligheter.

Det antas även att programmet kan medföra miljömässiga fördelar gentemot att bokföra för hand, eftersom pappersåtgången minskar vid användning av ett elektroniskt program. Med fler internetbaserade system finns möjligheten att helt eliminera pappersanvändande allteftersom ytterligare myndigheter och företag börjar stödja, och så småningom övergå helt till, internetbaserade tjänster. I och med införandet av elektroniska fakturor, som både SuperBooky och andra bokföringsprogram använder sig av, kan även dessa skickas och betalas över internet. Men med ett ökat internet- och datoranvändande kommer även energiåtgången att öka vilket skulle kräva renare energi för att vara helt gynnsamt för miljön. Intresset, både för internetbaserade tjänster och för miljömässiga produkter, tros gynna SuperBooky.

6.6 Framtida arbete

Med tanke på den begränsade tidsramen som projektet ligger inom finns det delar av applikationen som skulle behöva mer arbete innan de blir helt fullständiga. För att applikationen ska kunna lanseras som ett bokföringssystem krävs att utförliga säkerhetstester görs både för att försäkra att användaren kan använda applikationen på ett säkert sätt men även för att bekräfta att applikationen inte har några som helst brister vad gäller de bokföringsregler som finns för företag baserade i Sverige. Det finns även några funktioner som back-enden har stöd för men som inte implementerats till fullo i front-enden. Ett exempel på detta är bokföringsrapporterna som går att återfinna i bilaga F och G.

Projektets huvudfokus var, som nämnts tidigare, att skapa ett användarvänligt bokföringssystem då många av de som finns på marknaden har funktioner som är överflödiga för småföretag. Med detta i åtanke hade projektet gynnats av ytterligare användartestning samt förenklande av de funktioner som finns i applikationen. Även formgivningen och egenskaperna av funktioner i systemet hade kunnat förbättras med ytterligare testning.

7 Slutsats

En välfungerande applikation som täcker grundläggande bokföringskrav har skapats. Applikationen och användartesterna som utförts visar på att bokföring för småföretag inte behöver vara särskilt komplext för att vara fungerande.

Användartesterna som utfördes visar på goda resultat. Samtliga funktioner i applikationen var enklare att använda än motsvarande i Zervant. Utöver detta tog 6 av 9 av funktionerna kortare tid att använda än i Zervant, vilket upplevdes som mycket positivt av användarna. Funktionen ”Snabb bokföring” var speciellt uppskattad av testarna. Den upplevdes som både snabb och intuitiv, och föredrogs därför av testarna framför manuell bokföring. Projektgruppen anser även att funktionaliteten bakom ”Snabb bokföring” i SuperBooky kan användas av personer utan större kunskap inom bokföring, då den är presenterad på ett simpelt sätt.

Projektets lösning på problemformuleringen visar att det går att effektivisera bokföringsprocessen. Genom att underlätta denna process tror projektgruppen att nyföretagare med begränsad bokföringserfarenhet enklare kommer kunna sätta sig in i grundläggande bokföring.

Referenser

AngularJS (u.å.[a]). *AngularJS: Developer Guide: Conceptual Overview*. URL: https://docs.angularjs.org/guide/concepts (hämtad 2015-05-12).

— (u.å.[b]). *AngularJS: Developer Guide: Directives*. URL: https://docs.angularjs.org/guide/directive (hämtad 2015-05-12).

— (u.å.[c]). *AngularJS: Developer Guide: Introduction*. URL: https://docs.angularjs.org/guide/introduction (hämtad 2015-05-12).

— (u.å.[d]). *AngularJS: Developer Guide: Modules*. URL: https://docs.angularjs.org/guide/module (hämtad 2015-05-12).

— (u.å.[e]). *AngularJS: Developer Guide: Templates*. URL: https://docs.angularjs.org/guide/templates (hämtad 2015-05-12).

Bartledan (2009). *Overview of a three-tier application*. URL: http://commons.wikimedia.org/wiki/File:Overview_of_a_three-tier_application_vectorVersion.svg.

BAS (2015). *Kontoplan BAS 2015*. URL: http://www.bas.se/documents/2015/Kontoplan_Normal_2015_ver1.pdf (hämtad 2015-05-14).

BFN (2011). *Att föra bok*. URL: http://www.bfn.se/info/attforabok-12.pdf (hämtad 2015-05-14).

Cooper, Alan m. fl. (2014). *About Face: The Essentials of Interaction Design*. 4. uppl. John Wiley & Sons. ISBN: 978-1-118-76657-6.

Cornell, S. (2013). *Minimum Acceptable Code Coverage*. URL: http://www.bullseye.com/minimum.html (hämtad 2015-05-13).

Crockford, Douglas (2006). *The application/json Media Type for JavaScript Object Notation (JSON)*. RFC 4627.

Fielding, R. m. fl. (1999). *Hypertext Transfer Protocol – HTTP/1.1*. RFC 2616.

Flanagan, David (2002). *JavaScript: The Definitive Guide*. 4. uppl. Sebastopol: O'Reilly Media, Inc. ISBN: 978-0-596-00048-6.

Flying Saucer (2013). URL: https://code.google.com/p/flying-saucer/ (hämtad 2015-05-13).

Fowler, M. (2004). *Inversion of control containers and the dependency injection pattern*.

Gehrke, Dave och Efraim Turban (1999). "Determinants of successful website design: relative importance and recommendations for effectiveness". I: *Systems Sciences, 1999. HICSS-32. Proceedings of the 32nd Annual Hawaii International Conference on*. (5–8 jan. 1999). IEEE. Maui, HI, 8–pp.

Gierke, O., T. Darimont och C. Strobl (2015). *Spring Data JPA*. URL: http://docs.spring.io/spring-data/jpa/docs/current/reference/html/ (hämtad 2015-04-09).

Goodman, Danny, Michael Morrison och Brendan Eich (2007). *JavaScript Bible*. 6. uppl. Indianapolis: John Wiley & Sons. ISBN: 978-0-470-06916-5.

ISO (2010). *Ergonomics of Human-system Interaction: Part 210: Human-centred Design for Interactive Systems*. ISO.

Jendrock, E. m. fl. (2014). *The Java EE 7 Tutorial, Volume 1*. 5. uppl. Addison-Wesley Professional.

Krasner, G. E. och S. T. Pope (1988). ”A description of the model-view-controller user interface paradigm in the smalltalk-80 system.” I: *Journal of object oriented programming* 1.3, s. 26–49.

Kurose, James F. och Keith W. Ross (2013). *Computer Networking, A Top-Down Approach*. 6. uppl. Essex: Pearson Education Limited. ISBN: 978-0-273-76896-8.

Martin, R. C. (2000). ”Design principles and design patterns”. I: *Object Mentor* 1, s. 1–34.

Meyer, Marc H och Peter H Webb (2005). ”Modular, layered architecture: the necessary foundation for effective mass customisation in software”. I: *International Journal of Mass Customisation* 1.1, s. 14–36.

MSDN (1999). *Understanding Interface-Based Programming*. URL: https://msdn.microsoft.com/en-us/library/aa260635(v=vs.60).aspx (hämtad 2015-04-02).

NNGroup (2012). *Usability 101: Introduction to Usability*. URL: http://www.nngroup.com/articles/usability-101-introduction-to-usability/ (hämtad 2015-05-16).

RequireJS (u.å.[a]). *RequireJS*. URL: http://requirejs.org/ (hämtad 2015-05-11).

— (u.å.[b]). *Why Web Modules?* URL: http://requirejs.org/docs/why.html (hämtad 2015-05-11).

Riksdag, Sveriges (1962). *Brottsbalk (1962:700)*. URL: http://www.riksdagen.se/sv/Dokument-Lagar/Lagar/Svenskforfattningssamling/Brottsbalk-1962700_sfs-1962-700/%5C#K1 (hämtad 2015-04-02).

— (1994). *Mervärdesskattelag (1994:200)*. URL: http://www.riksdagen.se/sv/Dokument-Lagar/Lagar/Svenskforfattningssamling/Mervardesskattelag-1994200_sfs-1994-200/ (hämtad 2015-05-16).

— (1995). *Årsredovisningslag (1995:1554)*. URL: http://www.riksdagen.se/sv/Dokument-Lagar/Lagar/Svenskforfattningssamling/rsredovisningslag-19951554_sfs-1995-1554/ (hämtad 2015-05-16).

— (1999). *Bokföringslag (1999:1078)*. URL: http://www.riksdagen.se/sv/Dokument-Lagar/Lagar/Svenskforfattningssamling/Bokforingslag-19991078_sfs-1999-1078/%5C#K5 (hämtad 2015-04-01).

Skatteverket (2010). *Bokföring - vad kräver lagen?* URL: http://www.skatteverket.se/foretagorganisationer/foretagare/bokforingbokslut/bokforingvadkraverlagen.4.18e1b10334ebe8bc80005195.html (hämtad 2015-05-16).

Spring (u.å.[a]). *The IoC Container*. URL: http://docs.spring.io/spring/docs/current/spring-framework-reference/html/beans.html (hämtad 2015-04-09).

— (u.å.[b]). *Web MVC Framework*. URL: http://docs.spring.io/spring/docs/current/spring-framework-reference/html/mvc.html (hämtad 2015-04-09).

— (u.å.[c]). *Web MVC framework*. URL: http://docs.spring.io/spring/docs/current/spring-framework-reference/html/mvc.html%5C#mvc-ann-requestbody (hämtad 2015-05-11).

Stallings, William och Lawrie Brown (2015). *Computer Security, Principles and Practice*. 3. uppl. Essex: Pearson Education Limited. ISBN: 978-1-292-06617-2.

Strannered (2010). *Model-view-controller*. URL: http://commons.wikimedia.org/wiki/File:ModelViewControllerDiagram2.svg.

W3C (2010). *Document Object Model (DOM) Level 1 Specification*. URL: http://www.w3.org/TR/2000/WD-DOM-Level-1-20000929/introduction.html (hämtad 2015-05-18).

ZeroTurnaround (2015). *Top 4 Java Web Frameworks Revealed: Real Life Usage Data of Spring MVC, Vaadin, GWT and JSF*. URL: http://zeroturnaround.com/rebellabs/top-4-java-web-fr%20ameworks-revealed-real-life-usage-data-of-spring-mvc-vaadin-gwt-and-jsf/ (hämtad 2015-06-01).

Zervant (u.å.). *Faktureringsprogram och bokföring för små företag och enskild firma*. URL: http://www.zervant.com/se/index.php (hämtad 2015-06-01).

A Projektbeskrivning

Modernt webbaserat bokföringsprogram för småföretag

Projektkod: DATX02-15-02

För ett nystartat företag är det idag mycket krångligt att komma igång med bokföring. De bättre programmen som finns i Sverige idag är ofta byggda för större företag och för personer som har tidigare erfarenhet av bokföring. Vi tror att det går att bygga ett bättre sätt för nystartade företag att bokföra!

För många utvecklare kan det tyckas torrt att välja ett kandidatarbete med bokföring i namnet. Ekonomidelen i projektet tänker vi dock kan få olika stort utrymme beroende på intresse och erfarenheter i gruppen. Det huvudsakliga målet är att bygga en modern webbapplikation som löser ett problem bättre än existerande alternativ.

Projekt- /problembeskrivning

I kandidatarbetet vill vi från grunden tänka ut hur ett bokföringsprogram för 2015 bör se ut och fungera. Arbetet tänker vi därför handlar om framförallt följande tre delar. Dels bygga en modern webbapplikation med ett bra gränssnitt. Dels en säker och stabil back-end som ska hålla oavsett vilka bad guys som knackar på dörren. Sist ekonomidelen och domänanalysen om hur ett bokföringsprogram faktiskt behöver fungera. Eventuellt vill vi även se vilka ekonomiska möjligheter det finns att starta ett företag baserat på projektet, antingen under eller efter våren.

Om något av följande passar in på dig tycker vi att du ska söka!

- Du vill lära dig mer om webbutveckling (front-end eller back-end).
- Du vill undersöka och bygga ett komplett system för att lösa ett väldefinierat och verklighetsbaserat problem.
- Du är intresserad av säkerhet och vill se till att applikationen är tillförlitlig.
- Du vill göra något som idag är komplicerat enkelt med god användarvänlighet.
- Du är intresserad eller vill lära dig mer om småföretag eller ekonomi.
- Du vill undersöka och testa marknaden för ett visst projekt.

Att bygga ett komplett bokföringsprogram är såklart inget vi kommer att hinna under ett kandidatarbete. Vi tänker oss att vi tidigt under projektet bestämmer oss för de viktigaste användningsområdena och sedan satsar på dessa enligt MVP-principen.

B Intervjumall

Ålder:

Vad jobbar du med?

Hur länge har du bokfört?

Använder du dig av ett bokföringsprogram? Om ja, vilket?

Vad anser du är detta programmets styrkor/svagheter?

Vilka är, enligt dig, de tre viktigaste funktionerna i ett bokföringsprogram?

Vad är de vanligaste bokföringssysslorna för dig?

Är dessa sysslor lätta att genomföra i ditt program?

Vad hade du svårt för när du började bokföra?

Har du råkat ut för affärshändelser du inte vet hur du ska bokföra? Om ja, vad?

C Design

Home page:

- Vi har fått betalt (händelse)
  - kund
  - produkt (antal)
  - typ av betalning
- Vi har betalat (händelse)
  - vad har vi köpt?
    - ange moms eller stående moms
- Vi har arbetat
  - Skapa faktura
  - Ange kund
  - Ange produkt
- Skapa faktura
- Undermeny
  - Kort status/info om företaget
  - Översikt

Header:

- Kunder
  - Lista
  - Lägg till
    - företag/privatkund för att veta om exkl eller inkl moms, namn,
  - Redigera/radera
- Produkter
  - Produkter + Tjänster
  - Lista
  - Lägg till
    - produkt, beskrivning, enhet, bas för prissättning, moms, pris/enhet
  - Redigera/radera
- Bokföring
  - Lista (likt banksida)
  - Sidomeny med alla konton så man kan hoppa mellan
  - “Skapa kontorapport”
- Rapporter
  - Skapa rapport
  - Men ska även komma hit om man går från annan sida
  - Balans
  - Resultat
  - Moms
  - Tids
- Inställningar

D Intervjusvar

Din ålder	18-25	25-35
Vad jobbar du med?	Sektion på Chalmers	förening
Hur länge har du bokfört?	Ca 10 månader	1 år
Använder du dig av ett bokföringsprogram?	Visma	visma
Vad anser du är detta programmets styrkor/svagheter?	VIsmas styrkor ligger i att allt finns på ett och samma ställa, man kan lätt se över budget, göra verifikat samt fakturor i samma program. Dess svagheter kommer när man vill ändra fel i bokföringen, specifikt fel datum - då krävs att man gör ett nytt verifikat. Det faktum att man är bunden till den dator där bokföringen finns är även ett problem, mer distribution skulle vara bra.	Det är lite oklart att visa knappar funkar olika beroende på om du är i verifikat-mode eller om du är i konto-mode. Vissa knappar är inte nödvändiga att ha framme i standardmode, man kanske skulle kunna specificera hur mycket funktioner som ska vara lättillgängliga.
Vilka är, enligt dig, de tre viktigaste funktionerna i ett bokföringsprogram?	Enkelt att använda, lättåtkomligt samt bra uppföljningsmöjligheter ifall fel upptäcks.	1. Tydligt vilka ikoner som gör vilken funktion. 2. Möjligheten att bokföra utgifter/intäkter kring ett visst projekt. 3. Att ha en sammanställd lista för varje konto hur balansen ser ut.
Vilka är, för dig, de vanligaste bokföringssysslorna?	Betalning av utlägg, betala och skicka fakturor, inköpt av mat och Kontors-, städ- och byggmaterial.	Betalning av fakturor, föra in verifikat.
Är dessa sysslor lätta att genomföra i det program du använder/har använt?	Ja	Efter att ha lärt sig den hårda vägen vilka symboler som gör vad så funkar det relativt smidigt. Mycket ligger nog dock i att när jag började hade jag aldrig bokfört innan, och endast gjorde det ett år.
Vad hade du svårt för när du började bokföra?	Främst hur bokföring fungerar i isg (kredit/debit etc.)	Jag hade bara tillgång till en windows version, men satt själv på Mac OS, vilket gjorde att jag behövde ha en virtuell desktop för att kunna använda visma. Det gjorde upplevelsen ännu värre, men det är möjligt att nyare versioner av programmet funkar på Mac.
Har du råkat ut för affärshändelser du inte vet hur du ska bokföra?	Inköp av fordon, då det inte räknas som en direkt utgift utan en tillgång som avskrivs. Vissa händelser som jag inte anser har passande konton.	

Din ålder	18-25	18-25
Vad jobbar du med?	Förening	förening
Hur länge har du bokfört?	1 år	1 år
Använder du dig av ett bokföringsprogram?	ja, kommer tyvärr inte ihåg	visma
Vad anser du är detta programmets styrkor/svagheter?	Spåra konton fungerade smidigt och projekt (man unde katagorisera verifikat i olika grupper och få ut balans/resultat rapporter för enskilda projekt).	lite mycket aktivitetsval
Vilka är, enligt dig, de tre viktigaste funktionerna i ett bokföringsprogram?	Smidigt sätt att skapa och skriva ut verifikat. Lätt sätt att felsöka bland olika konton när man märker att nått inte stämmer. Lagra konton så man slipper komma ihåg och skriva in nummer varje gång.	kunna ändra konton få en bra översikt tips och bra sökmotor/hjälpmotor
Vilka är, för dig, de vanligaste bokföringssysslorna?	Skapa verifikat, ändra verifikat. Titta på resultat/balansrapporter, vet inte riktigt om det är en syssla annars skulle jag säga skapa fakturor.	betalning fakturor inköp varor intäkter
Är dessa sysslor lätta att genomföra i det program du använder/har använt?	I regel va programmet lite för avanserat för vad jag behövde så det fanns ofta massa fält jag inte fyllde i eller flikar jag inte öppnade. Annars kändes det smidigt.	ja
Vad hade du svårt för när du började bokföra?	Komma ihåg olika konton och hitta rätt, även fast det fanns en lista med namn på alla inlagt i programmet. Kredit och debit va lite förvirrande.	att man ska göra så mycket transaktioner fram och tillbaka mellan olika konton (om man har fler än 1 kort & konto)
Har du råkat ut för affärshändelser du inte vet hur du ska bokföra?	Mest varit osäker på vilket konto som passade till olika saker.	över åren för att det ska "hamna" bokföringsmässigt på rätt år. löste de tillslut men va lite krångligare än de andra

Din ålder	18-25	18-25
Vad jobbar du med?	Ideell förening	Eget företag
Hur länge har du bokfört?	18 månader	3 år
Använder du dig av ett bokföringsprogram?	Visma	Kapitas
Vad anser du är detta programmets styrkor/svagheter?	Går snabbt att lära sig grunderna. Finns också mycket avancerade funktioner och tillägg att utforska. Dessutom bra onlineguider för användande av programmet! Finns många väldigt nyttiga och givande utskrifter man kan göra, samt exportera till Excel för att enkelt kontrollera sin bokföring. Skulle vilja ha öppet två bokföringsår samtidigt! Och framförallt kunna ha upp flera utskrifter på bildskärmen samtidigt samt inte behöva stänga ner utskriften för att gå tillbaka till programmet.	Väldigt enkelt att komma igång och gratis så länge man klarar sig utan att få den ingående balansen automatisk ifylld baserad på föregående år. Det finns en färdig kontoplan med alla standardkoder där man kan aktivera de konton man behöver i sin egen bokföring. Man kan göra egenprogrammerade händelser som "betald kundfaktura" så man själv kan skräddarsy vilka konton som ska användas, super smidigt om man har väldigt många affärshändelser av en specifik typ. När man väl slutfört ett verifikat så går det att ta bort eller ändra utan att behöva göra ett motverifikat så länge det gäller senast gjorda verifikatet. Skit bra grej!
Vilka är, enligt dig, de tre viktigaste funktionerna i ett bokföringsprogram?	- Diverse utskrifter (Balansrapport, Resultaträkning, Kontoanalys) - Enkelt och användarvänligt vid "inknappande" av verifikat, dvs att man enkelt kan lägga in många verifikat av "bara farten" genom kommandon och autofyll etc. - Tydligt och enkelt användande av kontoplan och verifikatlista, dvs det som krävs av amatören	Möjligheten att göra egenprogrammerade snabbkommandon för konton. Färdig kontoplan som standardmall Kunna ta bort felgjorde verifikat utan att det blir plottrigt med motverifikat.
Vilka är, för dig, de vanligaste bokföringssysslorna?	Verifiering av transaktioner, dvs fakturor, inköp intäkter osv. Avstämning av konton vid bokslut.	Kundbetalning av faktura Köp av domännamn
Är dessa sysslor lätta att genomföra i det program du använder/har använt?	Ja! Med lite knep så!	Ja
Vad hade du svårt för när du började bokföra?	Jag hittade inte i programmet och vågade inte klicka omkring i rädsla för att lägga in något jag inte kunde ta bort eller ändra.	Vilka konton som ska användas men detta löser man med en enkel googling.
Har du råkat ut för affärshändelser du inte vet hur du ska bokföra?	Ja, men det har mest berott på okunskap kring kontoplanen och hur man skall kategorisera ett inköp.	Ja, köp av tjänst utanför EU var klurigt att bokföra.

Din ålder	25-35	25-35
Vad jobbar du med?	Enskild firma	Student
Hur länge har du bokfört?	1 år	Under ett år som kassör ideell organisation.
Använder du dig av ett bokföringsprogram?	Nej	Fort nox
Vad anser du är detta programmets styrkor/svagheter?		Väldigt användarvänligt. Perfekt för bokföring i föreningar där man har mandatperioder på 1 år. Bokföringsprogram för dummies. Svagheterna är att det inte är gjort för stora organisationer. De har dessutom riktigt hunnit anpassa sin verksamhet till större verksamheter.
Vilka är, enligt dig, de tre viktigaste funktionerna i ett bokföringsprogram?	Kunna bokföra kvitton på ett smidigt sätt, t.ex. snabbt från mobiltelefonen Snabbt se en sammanställning av viktiga nyckeltal för min ekonomi.	Bokföringen och skicka fakturor som enkelt bokförs automatiskt.
Vilka är, för dig, de vanligaste bokföringssysslorna?	Representationsluncher och förbrukningsinventarier. Bokföring av inkomst	Bokföring av utbetalningar, betala fakturor.
Är dessa sysslor lätta att genomföra i det program du använder/har använt?		Ja, fortnox har en intuitivt gränssnitt där man bland annat ser det nya värdet på ett konto innan man bokfört det.
Vad hade du svårt för när du började bokföra?	Att lära sig de olika begreppen och bokföra de olika posterna.	Minns ej.
Har du råkat ut för affärshändelser du inte vet hur du ska bokföra?	Fakturor som löper över årsskiftet (ex. kundfordringar). Fakturor för tjänster jag köpte in utanför Sverige. RUT-intäkter kombinerat med tillhörande faktura.	Nej.

Din ålder	25-35	18-25
Vad jobbar du med?	konsult, företagare	Förening
Hur länge har du bokfört?	1 år	sedan gymnasiet
Använder du dig av ett bokföringsprogram?	Zenconomy	Visma Online
Vad anser du är detta programmets styrkor/svagheter?	styrka: lätt att komma igång. svaghet: stelt, dåliga konfigurationsmöjligheter, ej flexibelt dimensionsantal	Bra smidigt men inte super kraftigt.
Vilka är, enligt dig, de tre viktigaste funktionerna i ett bokföringsprogram?		Enkelt, bra interface, bra översikter
Vilka är, för dig, de vanligaste bokföringssysslorna?	skapa kundfakturor	faktura betalning och faktura utskick
Är dessa sysslor lätta att genomföra i det program du använder/har använt?	ja	ja
Vad hade du svårt för när du började bokföra?		Periodisering
Har du råkat ut för affärshändelser du inte vet hur du ska bokföra?	utlandsköp, klurig moms, tex leasing. avdragsgilla(?) kostnader	nej

E Användartester

SuperBooky

	Skapa användarkonto	Skapa kund	Skapa produkt
Tid (sekunder)	25,86	26,43	32,86
Antal click	4,57	4,43	7,00
Bedömd svårighet (1-5)	1,29	1,29	1,43

Zervant

	Skapa användarkonto	Skapa kund	Skapa produkt
Tid (sekunder)	33,00	67,86	98,29
Antal click	5,14	5,71	20,14
Bedömd svårighet (1-5)	1,29	2,00	3,86

SuperBooky

	Bokföra betald faktura	Bokföra inkommen faktura (kund bet)	Bokföra pengar från kassa till bank
Tid (sekunder)	41,43	20,00	60,67
Antal click	8,57	5,86	10,17
Bedömd svårighet (1-5)	2,29	1,43	3,17

Zervant

	Bokföra betald faktura	Bokföra inkommen faktura (kund bet)	Bokföra pengar från kassa till bank
Tid (sekunder)	37,86	34,57	61,67
Antal click	7,86	7,00	9,50
Bedömd svårighet (1-5)	2,43	2,57	3,33

SuperBooky

	Skapa tidslogg	Skapa faktura	Åtgärda bokföringsfel
Tid (sekunder)	45,43	35,29	49,29
Antal click	9,57	7,43	8,00
Bedömd svårighet (1-5)	2,29	1,57	1,71

Zervant

	Skapa tidslogg	Skapa faktura	Åtgärda bokföringsfel
Tid (sekunder)	82,43	23,86	20,14
Antal click	16,57	5,86	5,29
Bedömd svårighet (1-5)	3,29	1,43	1,29

F Resultatrapport

Resultatrapport

Företagsnamn	Jakob
Utskriftsdatum	18/05/2015
Räkenskapsår	1000
Period	10/11/1000 - 01/02/3000

Senaste verifikationsnummret: 7375

Intäkter	**Balans**
3055Någon inkomst	0.0

Kostnader	**Balans**
4055Någon utgift	300.0

Resultat
300.0

G Balansrapport

Balansrapport

Företagsnamn	Jakob
Utskriftsdatum	18/05/2015
Räkenskapsår	1000
Period	10/11/1000 - 01/02/3000

Senaste verifikationsnummret: 7388

Tillgångar	**Ingående balans**	**Utgående balans**
1001 Tillgång 3	0.0	-100.0
1010 Kassa	-300.0	-300.0
1055 Saker	-100.0	200.0
1056 Postgiro	0.0	-100.0
1057 Tillgång	0.0	-100.0
1065 Tillgång 2	0.0	-100.0
1275 Tillgång 5	0.0	-100.0
1375 Tillgång 6	0.0	-100.0
1475 Tillgång 7	0.0	-100.0
1575 Tillgång 8	0.0	-100.0
1675 Tillgång 9	0.0	-100.0
1775 Tillgång 10	0.0	-300.0
1875 Tillgång 11	0.0	300.0

Eget kapital och skulder	**Ingående balans**	**Utgående balans**
2003 Tillgång 4	0.0	-100.0
2018 Egna insättningar	0.0	300.0
2035 Skuld 3	0.0	-100.0
2055 Skuld	0.0	-100.0
2065 Skuld 2	0.0	-100.0
2075 Skuld 4	-100.0	-100.0
2085 Skuld 5	-300.0	-300.0
2095 Skuld 5	0.0	-100.0
2096 Skuld 6	0.0	-100.0
2097 Skuld 7	0.0	-100.0
2098 Skuld 8	0.0	-100.0

2099 Skuld 9	0.0	-100.0
2999 Skuld 10	0.0	-300.0

Ingående resultat	**Resultat**
-800.0	-3100.0