CHALMERS

Spelifiering av processtyrning

En studie av ny teknik för webbutveckling

Examensarbete inom Högskoleingenjörsprogrammet i Datateknik

ROBIN BRAAF

Institutionen för data- och informationsteknik
CHALMERS TEKNISKA HÖGSKOLA
Göteborg, Sverige 2014

Förord

Utförandet av detta examensarbete har varit lärorikt praktiskt i fallstudien och akademiskt i sammanställandet av rapporten.

Jag önskar tacka Calle Arnesten som varit mentor under arbetet och bidragit med djupgående kunskap om mjukvaruutveckling utöver förväntningarna. Det var även Calle som gav mig möjlighet att arbeta med hans produkt Kanbanflow på CodeKick. Jag önskar även tacka Uno Holmer för hans stöd med handledande av att sammanställa ett resultat i denna rapporten.

Abstract

This report is the result of a case study of implementing gamification on a process management web service. Node.js which has been studied in the case is a relatively new technology for developing web applications. The report gives an overview about using Node.js as a developer coming from a background in Java and from a business point of view why to choose Node.js as the technology to develop a web application. The case study is material for discussion and more research about how other use Node.js and a comparison with Java EE. The report ends with suggestions on areas for further study and a conclusion about how the environment around Node.js along with its non blocking I/O which makes it a competitive technology.

Sammanfattning

Denna rapport är resultatet av en fallstudie av att implementera spelifiering på en webbtjänst för projekthantering. Node.js som studerats under fallet är en relativt ny teknik för att utveckla webbapplikationer. Rapporten ger en överblick av hur det är att använda Node.js som utvecklare när man kommer från en bakgrund inom Java, och även från ett affärsmässigt perspektiv varför Node.js ska väljas för att utveckla en webbapplikation. Fallstudien ligger som grund för diskussion och efterforskningar inom andras erfarenheter av Node.js och en jämförelse med Java EE. Rapporten avslutas med förslag på vidare studier och en slutsats om att det är miljön runt Node.js tillsammans med icke-blockerande I/O som gör det till en konkurrenskraftig teknik.

Innehållsförteckning

Terminologi

CSS	Cascading Style Sheet är en stilmall som beskriver hur ett strukturerat dokument ska ritas ut, användas mycket i samband med HTML [8].
HTML	Hyper Text Markup Language är ett uppmärkningsspråk för att beskriva struktur och innehåll på en hemsida [7].
HTTP	Hyper Text Transfer Protocol är ett protokoll för att skicka data av MIME-typ mellan en användare som interagerar med en server genom en webbläsare [2].
I/O	Input / Output. Syftar på ett dataflöde in och ut från en applikation eller process.
Java EE	Java Enterprise Edition, samling specifikationer för att utveckla webbapplikationer med Java.
JSON	JavaScript Object Notation, ett textbaserat format för att skicka data.
Kanbanbräde	Ett planeringsbräde som används i utveckling med kanbanmetoden.
MIME	Multi Purpose Mail Extenstions, är en standard för att deklarera vilken typ av information en fil har [5].
Node.js	Ett minimalistiskt ramverk för att utveckla webapplikationer.
npm	Pakethanterare för Node.js.
ORM	Object Relational Mapping, mjukvara som används för att översätta en tabeller från en databas till objekt i en applikation.
Pomodoro	En teknik för att arbeta effektivt. Går ut på att sätta en timer och arbeta intensivt tills den ringer för att sedan ta en kort rast.
REST	Representational State Transfer, är en arkitekturell stil för att dela resurser på en webserver [9].
Spelifiering	Användning av spelmekanik i områden där de traditionellt inte hör hemma.
SQL	Structured Query Language, ett språk som används för att fråga databas efter data.
UML	Unified Modeling Language är ett universellt sätt att modellera system.
URL	Uniform Resource Locator, en textrepresentation för en resurs tillgänglig på Internet [10].
XML	Extensible Markup Language är ett universellt uppmärkningsspråk.

1. Inledning

Webapplikationer idag är byggda på många sammansatta komponenter. Förutom att det behövs en server för att hantera användarens HTTP-request behövs funktionalitet för att hantera bland annat sessioner, säkerhet, data och HTML-rendering. För att inte behöva göra om alla grundfunktioner finns det en mängd ramverk att välja bland som erbjuder dessa funktioner.

1.1 Bakgrund

Node.js är en teknikplattform som är till för att underlätta utveckling av webbapplikationer. Node.js erbjuder inte all funktionalitet direkt som andra tekniker gör, utan är mer minimalistiskt och fokuserar på att lösa skalbarhetsproblem [1]. Skalbarhet är en viktig fråga för alla webapplikationer eftersom de används av flera användare samtidigt och måste kunna fungera när användarantalet stiger. Men skalbarhet är inte allt utan det finns flera funktioner som en teknikplattform för webbapplikationer ska kunna erbjuda.

Node.js har inte kommit ut med version 1 ännu, men stora företag som Ebay, LinkedIn [4] och Walmart [6] använder Node.js i produktion och delar med sig av sina framgångar med plattformen vilket gör Node.js till en intressant teknik att undersöka.

1.2 Problem

Det finns många teknikplattformar för utveckling av webbapplikationer idag och det kan vara svårt att välja vilken som ska användas för att utveckla nya applikationer. Node.js är en nykomlig bland teknikplattformarna så varför ska den tekniken ses som ett alternativ till äldre, mer beprövade tekniker?

1.3 Syfte

Syftet med detta arbetet är att få förståelse för hur Node.js löser problem i samband med utveckling av webbapplikationer, bland annat hur tekniken löser skalbarhetsproblem.

1.4 Avgränsningar

Då varje webapplikation har många komponenter kommer det inte finnas tid till att granska varje komponent på djupet.

En del av syftet med arbetet är att jämföra Node.js med äldre tekniker, vilket specifikt syftar på teknikplattformen JavaEE.

1.5 Rapportöversikt

Rapportens första kapitel ger en introduktion och beskrivning till arbetet. I det andra kapitlet ges en beskrivning av metoden som använts och i det tredje kapitlet beskrivs miljön för arbetet. Det fjärde kapitlet handlar om hur implementationen av spelifiering på Kanbanflow gjordes. Resultaten beskrivs i det femte kapitlet där de även diskuteras. Rapporten avslutas med en slutsats i det sjätte kapitlet.

2. Metod

För att kunna ge en en så objektiv bild som möjligt av Node.js har kontakt etablerats med företaget CodeKick för att få arbeta med deras produkt som är en webbapplikation byggd på Node.js. Förutom praktisk erfarenhet av att arbeta med CodeKicks produkt kommer en handledare från företaget att hjälpa till genom att delta i parprogrammering, granska kod för att ge kritik och lära ut teori för god praxis vid utveckling med Node.js.

Erfarenheten av att använda Node.js kommer sedan att ligga till grund för efterforskning om andras erfarenheter av att använda tekniken, då deras erfarenheter kan vara ett komplement till erfarenheten av arbetet hos CodeKick.

Arbetet avslutas med en teoretisk studie om hur Node.js skiljer sig mot Java EE för att sedan kunna diskutera och dra slutsatser om för- och nackdelar med Node.js.

3. Teknisk Bakgrund

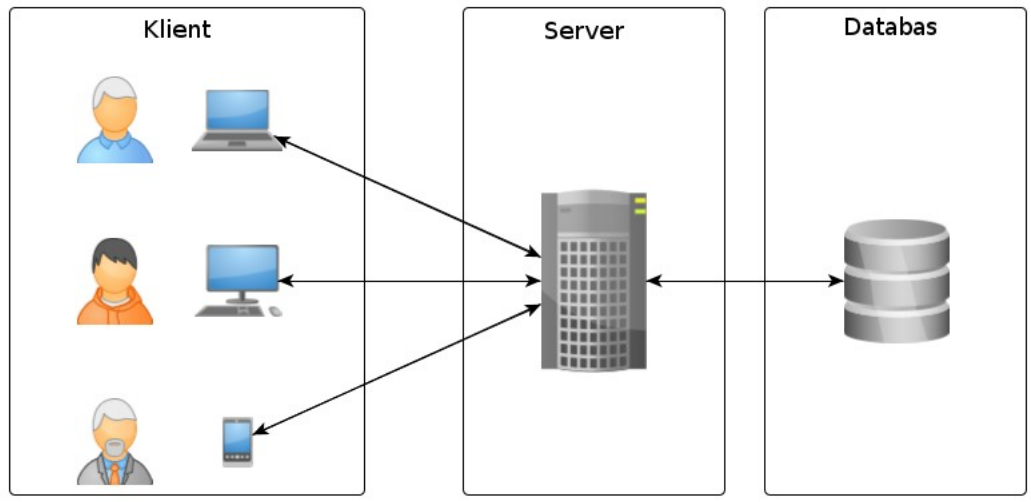
I detta kapitel ges först en översikt av miljön runt en webbapplikation och sedan specifika tekniker som använts i arbetet.

3.1 Webapplikationer

En webapplikation kan delas upp i flera lager som kan ses i figur 3.1. Applikationen används genom en klient som får användargränssnittet på begäran från servern. Flera klienter kan använda applikationen samtidigt och de kan även ha olika sorters hårdvara.

Webapplikationen körs på en server och behöver kunna kommunicera med både klienter och databas. Med skalbarhet hos en webapplikation menas att lägga till ytterligare hårdvara till servern för att kunna ta hand om flera klienter, och att den fortfarande fungerar som en enhet.

Databasen har hand om att lagra data för applikationen.



Figur 3.1: Översikt av en webapplikation

3.1.1 Klient

Klienten är en webbläsare som visar upp sidor uppbyggda av HTML och CSS, som den hämtar från servern. För att göra sidorna dynamiska för användaren, används JavaScript för att skriva om HTML- och CSS-koden när användaren interagerar med sidan.

Klienten är komponenten som en webapplikation har minst kontroll över. När webbläsaren har tagit emot en sida från servern finns det ingen kontroll för hur sidan används och det är därför viktigt att validera data som klienten skickar till servern.

3.1.2 Server

Servern är kärnan i en webapplikation. Servern har hand om att köra applikationen och

kommunicerar med klienter och databaser.
Funktionalitet som är vanlig på serversidan är följande [3]:

- Ta emot och svara på HTTP-förfrågningar.
- Känna till klientens tillstånd i applikationen genom sessionshantering.
- Koppla URLer till funktioner i applikationen.
- Validering av data från klienten.
- Generering av dynamiska sidor.
- Säkerhet för användarnas data.

Utöver denna funktionalitet är prestanda viktigt i applikationen. En webapplikation har oftast flera klienter som använder den samtidigt, därför är det viktigt att applikationen utnyttjar resurserna effektivt.
För att utveckla applikationen till servern finns det många teknikplattformar att bygga på för att slippa utveckla all funktionalitet från grunden. Bland dessa teknikplattformar finns Node.js och Java EE.

3.1.3 Databas

Databasen har hand om att lagra data som används i en webapplikation. Det finns olika kategorier av databaser att välja mellan och vilken som passar bäst beror på hur den ska användas. Bland de olika kategorierna finns bland annat relationsdatabaser och dokumentdatabaser.

3.2 Relationsdatabaser

Relationsdatabaser har sin grund redan från 1970 med Ted Codds dokument *A Relational Model of Data for Large Shared Data Banks*. Under ytan kan datastrukturen vara komplicerad medan datan presenteras som tabeller för programmeraren, eller användaren. Tabellerna kallas för relationer och defineras av scheman. Detta sättet att presentera data på har en matematisk grund i relationsalgebra som ger användaren stor flexibilitet i sina frågor till databasen. I början användes relationsalgebra direkt för att fråga efter data men numera används högnivåspråket SQL för att använda databasen [11].

3.2.1 Object Relational Mapping

Webapplikationer implementeras ofta med objektorienterade språk och presentationen av data i relationsmodellen passar inte helt med den objektorienterade modellen. För att komma runt detta finns det ORM-mjukvara som hjälper till med att översätta mellan de olika modellerna, exempelvis Hibernate till Java EE och Sequelize till Node.js.

3.3 Dokumentdatabaser

Dokumentdatabaser lagrar data i dokument istället för rader i tabeller och är en samling av nycklar med tillhörande värden. Dokumenten är inte definerade av något schema och är väldigt flexibla i hur de lagrar data. Hur dokumenten lagras och hur man frågar efter dokumenten skiljer sig mellan olika implementationer.

3.3.1 CouchDb

CouchDb är tillsammans med MongoDB den mest populära implementationen av en dokumentdatabas. CouchDb lagrar dokumenten i JSON-format som i exemplet på figur 3.2. Nycklarna *id* och *rev* är obligatoriska för alla dokument medan de andra nycklarna kan skilja sig mellan dokumenten. Värdena är flexibla och är inte av någon bestämd typ.

```
{
    id: "7c76bfa332706eba108efb2889000392",
    rev: "1-ebc11f778c81395b0bfd531ccf636d16",
    name: "Anders",
    notes: [ 1, 4, "seven"]
}
```

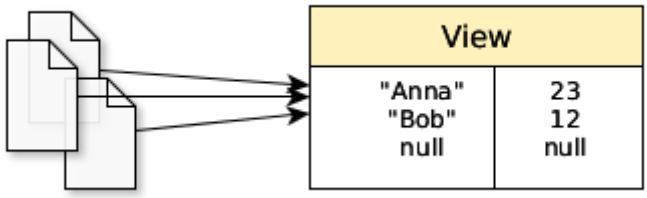
Figur 3.2: Exempel på ett dokument i CouchDb

För att fråga efter data används mapreduce-funktioner som sparas i speciella designdokument. Map-delen av funktionen körs separat och reduce-delen av funktionen utförs i samband med en fråga. Map-funktionen körs en gång för varje dokument i databasen och skapar en lista med nyckel-värdepar. Ett exempel på en map-funktion ses i figur 3.3 där värdet på *name* som en ny nyckel och värdet på *notes* som värde till den nya nyckeln. Resultatet av en map-funktion sparas och uppdateras när dokumenten ändras eftersom det är en krävande funktion att köra när det är större mängder dokument.

```
function(doc) {
    emit(doc.name, doc.notes);
}
```

Figur 3.3: En enkel map-funktion

För att illustrera map-funktionen så visar figur 3.4 hur den plockat ut värden från varje dokument och listat dem som nyckel-värdepar som är sorterade efter nyckeln. Dokument som saknar nycklar som map-funktionen använder ger ett null-värde i listan. Det är mot denna listan som användare ställer frågor för att hämta ut data från databasen.



Figur 3.4: Resultatet av en map-funktion på en samling dokument

3.4 JavaScript

JavaScript är ett objektorienterat skriptspråk som ofta används i webbläsare för att göra hemsidor dynamiska och interaktiva för klienter. Till skillnad från de flesta objektorienterade språk finns det inga klasser som definerar objekten, utan varje objekt har har en prototyp som det ärver egenskaper av när det skapas. Prototypen är också ett objekt som är länkat till en annan prototyp vilket bildar en kedja av prototyper ner till grundprototypen *Object.prototype* [13][14].

3.4.1 Funktioner

Funktioner i JavaScript är objekt som är länkade till *Function.prototype*, som i sin tur är länkad till *Object.prototype*. Som andra objekt kan funktioner skickas in som en parameter vid funktionsanrop. Funktioner som skickas in som parameter är inte garanterade att utföras synkront utan ursprungsfunktionen kan hinna köra klart sin kod först, vilket kan göra JavaScript-program komplexa. Vanligast är att funktioner som skickas in som parametrar ersätter *return* för att skicka tillbaks resultatet. Denna sorts funktion kallas för callback. Funktioner som anropar en funktion med en callback har sin resterande kod i callback-funktionen för att få ett synkront flöde i programmet som är lättare att följa [13][14].

3.4.2 Undantag

JavaScript har stöd för try-catch-satser som fungerar analogt med andra språk som Java för att hantera undantag. När callback-funktioner används kan undantag kastas efter try-catch satsen och det är därför en norm att första parametern i callback-funktionen alltid är ett felmeddelande eller null för att kunna hantera dessa undantag [13][14].

3.5 Node.js

Node.js är ett ramverk som är framtaget för att bygga skalbara webapplikationer. Ryan Dahl presenterade Node.js på JSConf 2009 [15]. Node.js har en händelse-driven arkitektur med en loop som hämtar händelser från en kö och skickar dem vidare till funktioner som hanterar händelserna. En grund i Node.js är att programmet inte ska låsas vid I/O så det finns bibliotek med funktioner som använder callback-funktioner för att hantera händelsen när I/O-

operationen är klar, så andra funktioner kan köra under tiden I/O-operationen pågår. För att exekvera JavaScript använder Node.js Googles V8-motor som kompilerar till maskinkod innan den kör koden.

3.5.1 Moduler

En applikation i Node.js byggs upp av moduler, som är en implementation av ett designmönster i Node.js. En modul är det som är publikt för utvecklaren i en Javascript-fil och är vanligen en funktion som gömmer detaljer om modulen men returnerar funktioner för att använda modulen. För att sedan använda en modul i programmet används funktionen *require* som i exemplet på figur 5, där en av Node.js inbyggda moduler hämtas.

```
1   var fs = require('fs');
```

Figur 3.5: Hämta in File System-modulen i Node.js

Moduler exporterar en funktion som körs när modulen hämtas med require, som i exempelfunktionen i figur 3.6. Exempelfunktionen returnerar ett objekt med en funktion som har tillgång till variabeln *msg*, även om den inte finns i objektet som returneras.

```
1   module.exports = function(message){
2
3       var msg = message;
4
5       return {
6           displayMessage: function(){
7               console.log(msg)
8           }
9       };
10  };
```

Figur 3.6: En modul i Node.js

För att använda modulen i figur 3.6 behöver den hämtas med *require*, vilket kan liknas med att importera en klass i Java. Det som är publikt för programmeraren som ska använda modulen finns i funktionen som returneras, därför anropas den direkt med exempelvis strängen ”*Hello World*” som parameter, och objektet som returneras lagras i en variabel som ses i exemplet på figur 3.7. För att skriva ut variabeln *msg* måste utvecklaren använda funktionen *displayMessage*. Variabeln *msg* är då privat för modulen och kan inte användas av programmeraren direkt.

```
1   var message = require('./message.js')('Hello World');
2   message.displayMessage();
```

Figur 3.7: Användning av modulen i figur 6

Att exportera funktioner i moduler ger fördelen att beroenden kan förmedlas till modulen som parametetrar. Detta förenklar enhetstestning genom att beroenden byts ut mot skenobjekt när

testet använder *require* för att hämta modulen. Skenobjekten kan skapas i testet som anonyma objekt för att fabricera data till modulen som testas.

3.5.2 Hantering av tredjepartsmoduler

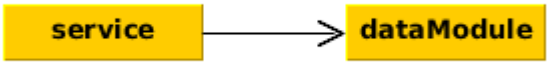
För att hantera tredjepartsmoduler kommer Node.js med en pakethanterare som kallas för npm. Vid skrivandes stund (maj 2014) finns det strax över 70 000 moduler i npm-registret vilket bidrar till att fylla ut med funktionalitet till Node.js. Tredjepartsmoduler som används av en Node.js-applikation skrivs in i en fil som döpts till *package.json* som ligger i rotmappen av applikationen. För att hämta och installera tredjepartsmodulerna används sedan *npm install* direkt i ett terminalfönster eller en kommandotolk.

3.5.3 Modellering av Node.js applikation

Avsaknaden av klasser gör att UML-diagram inte passar helt med en Node.js-applikation så i denna rapport används en liknande variant. Allting i JavaScript är objekt och de modelleras i rapporten som i figur 3.8. För objekt som är moduler är första bokstaven i namnet en gemen. För objekt som ska instansieras med *new* börjar namnet med en versal, för att visa att dessa objekt ska betraktas som Java-klasser.

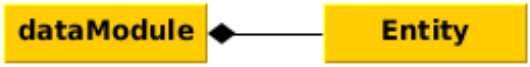
Figur 3.8: Modell för objekt

För att visa att en modul använder en annan modul dras en pil mellan modulerna som i figur 3.9 där *service* använder *dataModule*.



Figur 3.9: service använder sig av dataModule

När en modul lagrar ett eller flera objekt som har instansierats med *new* har den ägande modulen en ifylld diamant med ett streck till objektet som ägs av modulen.



Figur 3.10: dataModule har instansierat ett Entity objekt

3.6 Java EE

Java EE är en samling specifikationer för att utveckla webbapplikationer med Java och exekveras i en behållare på en applikationsserver som implementerar dessa specifikationer [9]. För att kompilera och paketera Java EE-applikationer används vanligen mjukvaran Maven. Maven tar även hand om tredjepartsbibliotek som applikationen behöver och har ett centralt register där de finns tillgängliga.

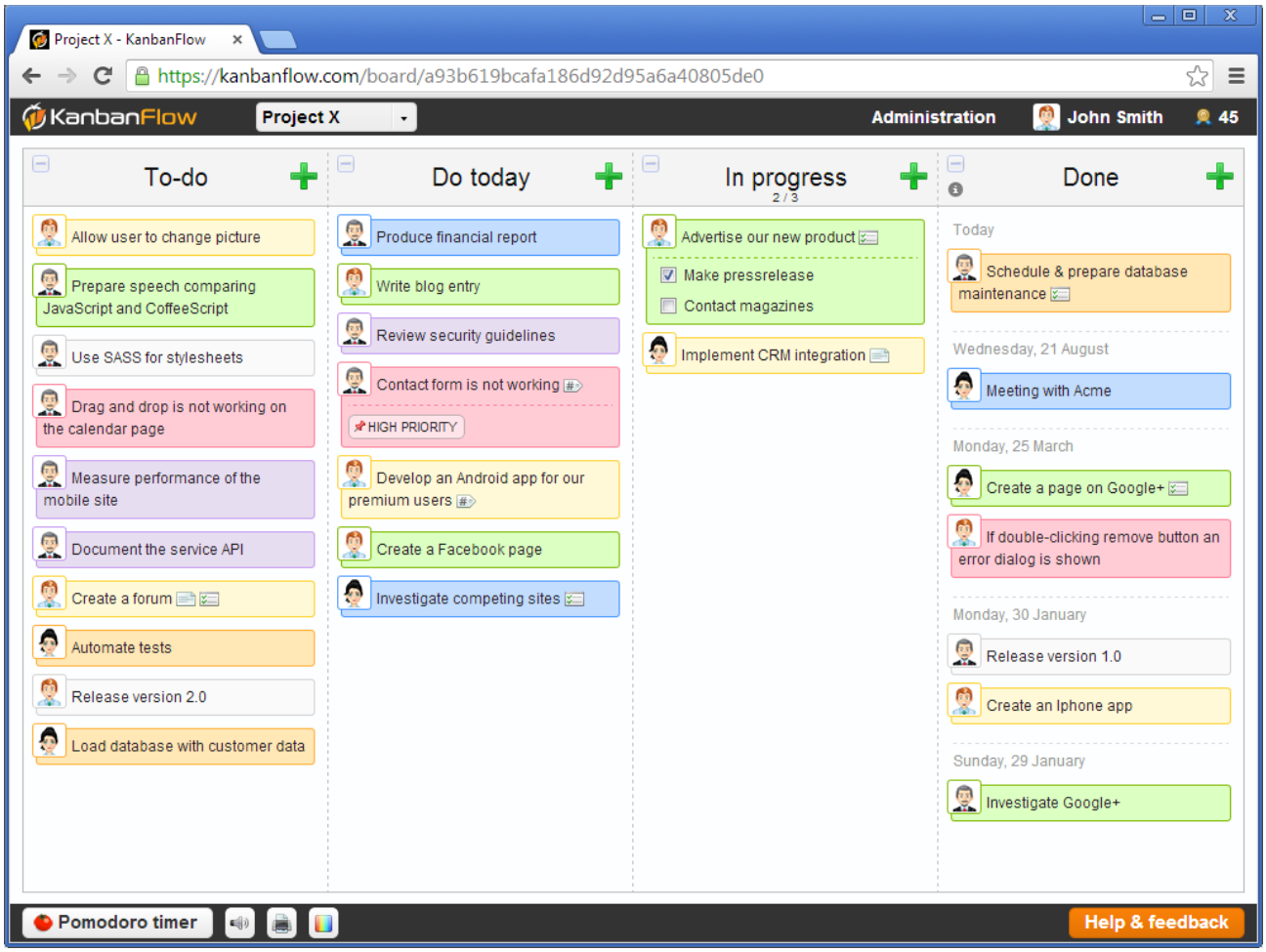
3.7 Kanbanflow

Kanbanflow är en webapplikation som är ett hjälpmedel för att hantera agila projekt. CodeKick började utveckla Kanbanflow under 2011 och har genom att erbjuda unika funktioner som swimlanes och pomodoro-timer fått uppmärksamhet i media, och kunnat växa till totalt 120 000 användare idag med toppar på 7000 användare samtidigt. Varje användare som är ansluten till applikationen använder en socket, alltså hanterar applikationen upp till 7000 sockets samtidigt vilket ställer hårda krav på prestanda.

3.7.1 Kanbanbräden

Huvudfunktionen på Kanbanflow är virtuella kanbanbräden. Flera användare kan samarbeta på ett bräde. När användare gör en ändring på ett bräde skickas ändringen ut i realtid med websockets så ändringen sker hos alla som tittar på det brädet.

Figur 3.11 visar en överblick av en grundläggande kanbanbräde på Kanbanflow. Dessa bräden används för att visualisera arbetet för ett projekt.

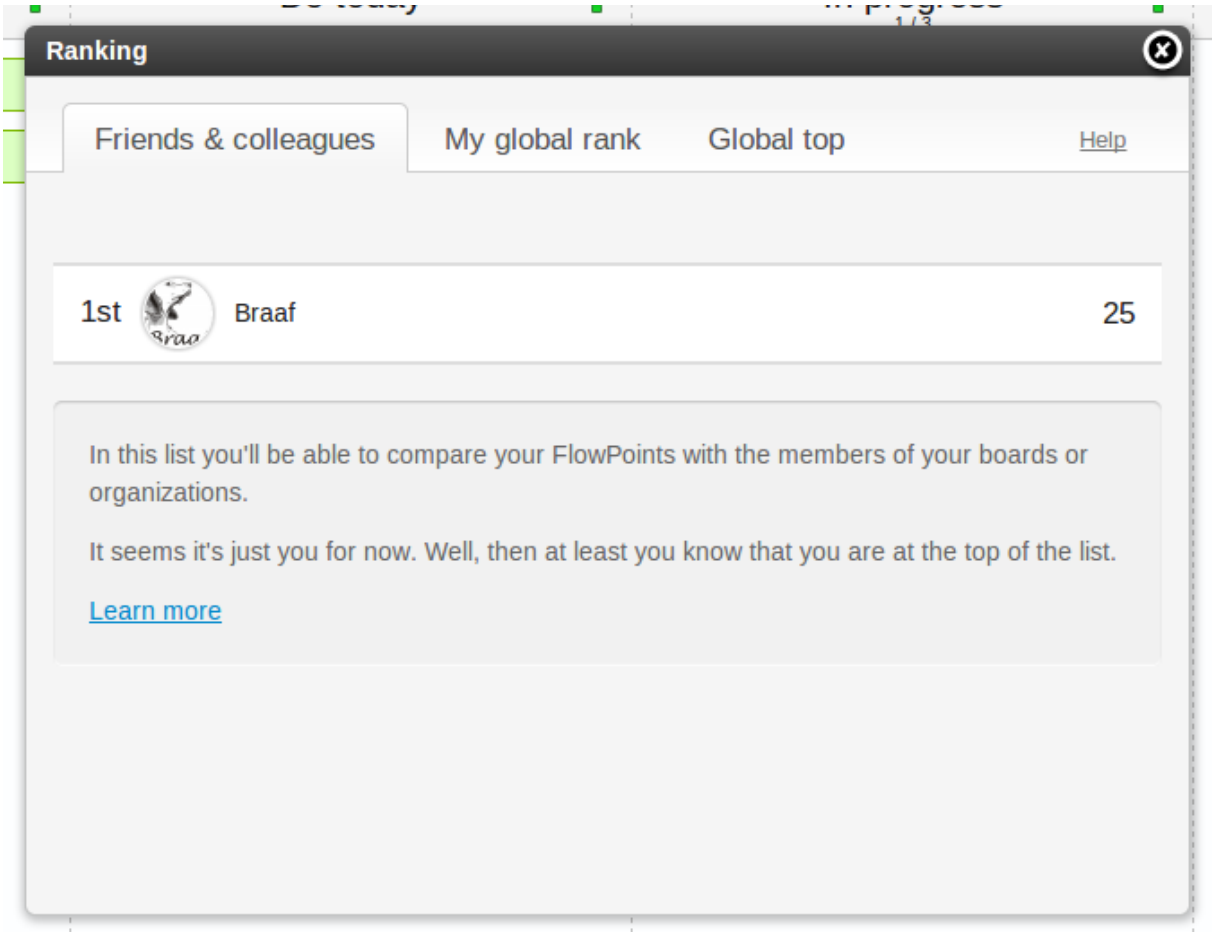


Figur 3.11: Ett kanbanbräde med några uppgifter. Figuren är hämtad från https://kanbanflow.com/press.

3.7.2 Pomodorotimer och poängräkning

Användare kan bland annat använda en timer för att registrera tid på en uppgift. Detta baseras på Pomodoro-metoden för att arbeta intensivt med en uppgift i ett mindre tidsintervall. När tiden på en startad timer är slut får användaren en notis om att pomodoro-sessionen är över och det är dags för en kort rast. Användaren får då poäng motsvarande antalet minuter som timern var inställd på. I högra övre hörnet kan användaren se totalt antal poäng som förtjänats genom pomodoro-timern, vilket är 45 poäng i figur 3.11.

Genom att trycka på poängen får användaren upp en dialog som visar rankningen för poängen. I figur 12 visas denna dialog och i toppen av dialogen finns tre flikar, en för varje sorts rankningslista som kan visas. Den första fliken *Friends & Colleagues* visar rankningen mellan andra användare som är bekanta genom att de samarbetar på en kanbantavla eller är medlemmar i samma organisation. Den andra fliken *My global rank* visar användarens totala rank och andra användare som är rankade strax över och under. *Global top* visar topplista över de högst rankade användarna.



Figur 3.12: Rankningsdialogen

3.7.3 Använda tekniker

Kanbanflow använder en del tekniker som behöver introduceras. Det första är modulen express som är ett mindre ramverk till Node.js. Det innehåller funktionalitet för att översätta URL-adresser till metoder och rendera HTML-mallar.
För enhetstestning används modulerna Buster.js och Sinon.js som innehåller funktionalitet för att testa asynkrona funktioner, stega i tid, och undersöka objekt och funktioner som används. Till integrationstesterna används Selenium genom modulen Soda. Selenium är ett testramverk för webbapplikationer och används för att simulera en användare på applikationen och kontrollera resultaten av användningen. Soda används för att programmera instruktioner till Selenium.

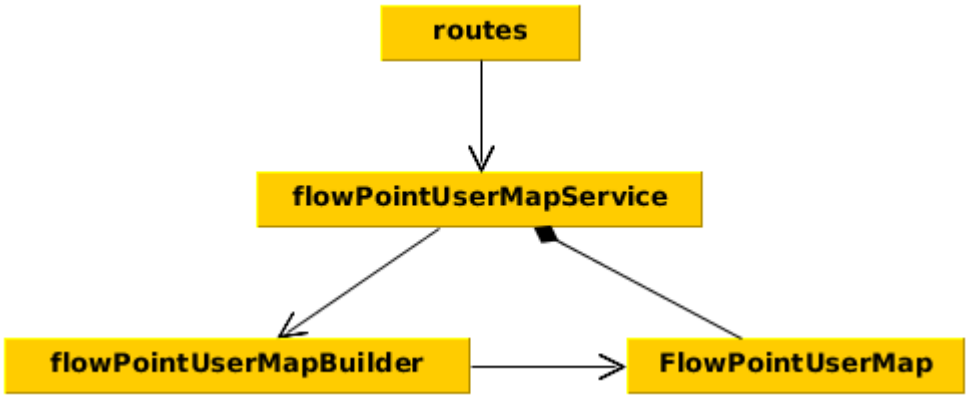
4. Fallstudie: Spelifiering i Kanbanflow

Fallstudien med Kanbanflow hos CodeKick är att vidareutveckla spelifieringen av pomodoro-timern. Första uppgiften är att utöka rankninglistorna till att visa rankning baserad på poäng intjänade under månadsintervall, för att låta nya användare komma in i spelifieringen lättare.

4.1 Analys av applikationen

Kanbanflow är uppdelad i flera små applikationer som sköter olika uppgifter. Alla uppgifter som har med spelifieringen att göra finns i en egen applikation som heter *flowPointCalculator.*

I *flowPointCalculator* finns det totalt fyra objekt, tre moduler och ett prototypobjekt. En översikt av kopplingarna mellan modulerna och prototypobjektet i *flowPointCalculator* visas i figur 4.1.



Figur 4.1: Översikt av flowPointCalculator

Det finns ett skript som startar applikationen och samlar moduler och beroenden i ett objekt som delas som inparameter till varje modul för att underlätta separat enhetstestning (se kap 3.5.1). I detta beroende-objektet finns inte *FlowPointUserMap* med, utan det instansieras i *flowPointUserMapService* och de har en hög koppling till varandra.

För att veta vilken funktionalitet som kan brytas ut gjordes en övergripande beskrivning av modulerna och objekten som läses i tabell 4.1.

Modul / Objekt	Beskrivning
routes	Består av metoder som hanterar HTTP förfrågningar på vissa URL:er. Svarar på förfrågningarna i JSON-format med data som hämtas från flowPointUserMapService.
flowPointUserMapService	Lagrar en rankningslista i en FlowPointUserMap och ser till att den uppdateras regelbundet. Har metoder för att hämta de olika listorna från FlowPointUserMap och dekorerar sedan objekten i listan med information om användaren som ska visas på klienten.
flowPointUserMapBuilder	Är som en fabrik för FlowPointUserMap. Innehåller metoder för att skapa och uppdatera FlowPointUserMap baserat på sparad data.
FlowPointUserMap	Ett objekt där prototypen är modifierad för att fungera som en rankningslista. Prototypen innehåller metoder för att hämta delar ur rankningslistan, där varje placering representeras av ett objekt. Dessa objekt innehåller förutom användar-id och poäng även information om användaren på placeringen är relaterad till användaren som frågar efter listan genom att de delar på en kanbantavla eller är medlemmar i samma företag.

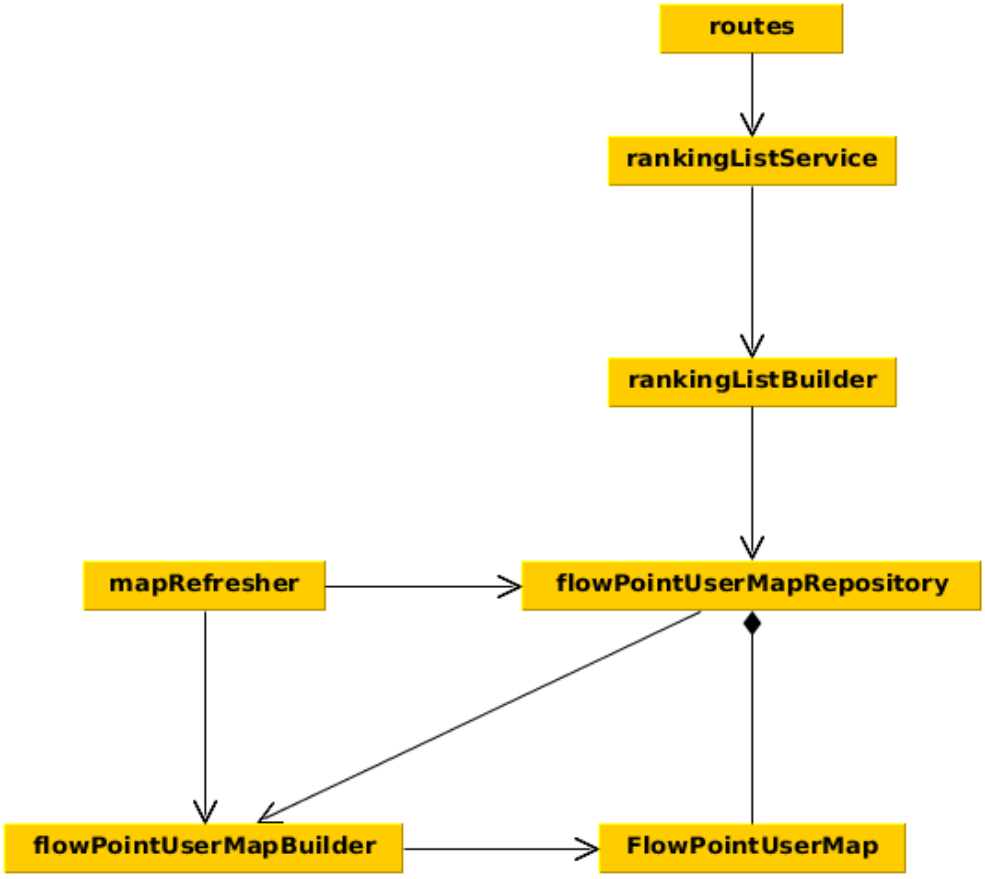
Tabell 4.1: Beskrivning av moduler och object i flowPointCalculator

4.2 Implementering av månadsrankning

Efter applikationen analyserats och teoristudier om Node.js och JavaScript genomförts, påbörjades implementeringen av månadsrankningen. I samband med studierna och analysen var detta det mest tidskrävande arbetet.

4.2.1 Omstrukturering av applikationen

För att implementera månadsrankningen behövde applikationen struktureras om. I samråd med extern handledare gjordes en ny struktur som visas på figur 4.2. Grundtanken bakom strukturen är att ha fler moduler som har mindre individuellt ansvar.



Figur 4.2: Översikt av flowPointCalculator efter planering av månadsrankning

I den nya strukturen ersätter *rankingListService* den gamla *flowPointUserMapService*, och den har som uppgift att dekorera listan som användaren frågat efter med information om användarna som finns i listan. Modulen *rankingListBuilder* hämtar listan som användaren har frågat efter, antingen från databasen eller från en *FlowPointUserMap*. Rankningslistorna som är lagrade i minnet hanteras av *flowPointUserMapRepository* där även framtida påbyggningar av applikationen kan få tillgång till akutella rankningslistor. Hanteringen av att uppdatera rankningslistor som ändras sköts nu av modulen *mapRefresher*. För att kunna lagra listor baserade på månadsintervall i minnet blir *flowPointUserMapBuilder* utbyggd med funktionalitet för att hantera det, annars ser den likadan ut som innan. Objektet *FlowPointUserMap* behålls i samma form men objekten som finns i listorna som hämtas ut förenklas till att endast innehålla rankningsnummer och användar-id.

4.2.2 Angreppssätt

För att implementera ändringarna angreps problemet uppifrån och ner med testdriven utveckling.

Innan en modul implementerades skrevs separata enhetstester för modulen. Eftersom beroenden skickas in när modulen instansieras i testet fejkades deras funktionalitet för att kunna använda moduler som inte ännu implementerats. Detta gjorde att arbetet gick nedåt i

hierarkin som syns i figur 4.2 eftersom utseendet på en modul var förutbestämt i testerna av modulen som implementerades innan. Undantaget är *mapRefresher* som skiljer sig något från resten av applikationen och används inte av någon annan modul.

4.2.3 Spara rankningslistor för gamla månader

Rankningslistor som gäller för tidigare månader behöver inte längre uppdateras och ska lagras i databasen för att inte ta upp plats i arbetsminnet. Först skulle *mapRefresher* ta hand om denna funktionen, men med flera instanser av applikationen igång samtidigt skulle det bli flera lagringar av rankningslistan. Tillsammans med hjälp av externa handledaren bestämdes det att lagringen av rankningslistorna skulle startas en gång i månaden med hjälp av operativsystemets schemaläggningsfunktion på servern.

Modulen som skapades för lagringen döptes till *rankingListPersistor* och utvecklades testdrivet som tidigare moduler. För att låta schemaläggaren lagra listorna skapades även ett nytt startskript som gav *rankingListPersistor* tillgång till beroenden på samma sätt som för skriptet som startar *flowPointCalculator,* men startade ingen server utan avslutades efter körning.

4.3 Brister i validering av data

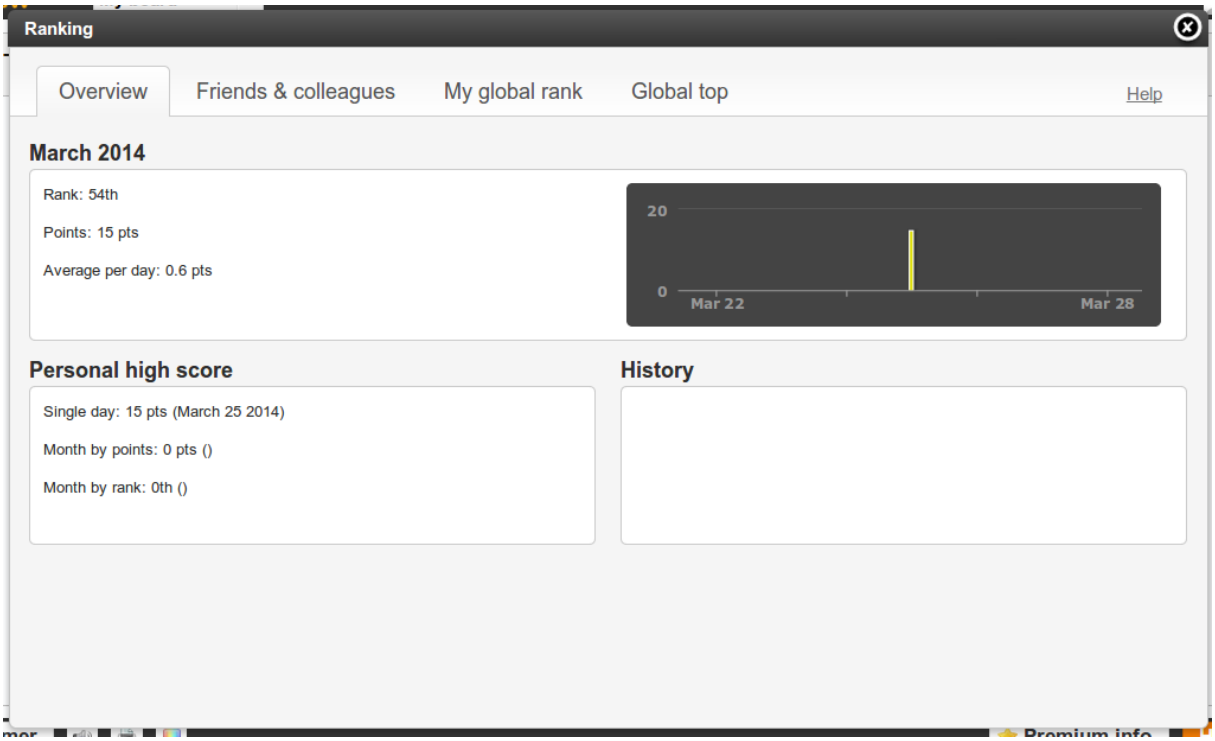
När måndadsrankningen var införd i applikationen framkom det att några användare hade tjänat mer poäng än vad som var möjligt under månaden. Genom att utnyttja en brist i hur servern validerade om en pomodoro-session var giltig kunde användare använda flera pomodoro-timers samtidigt och få mer poäng än tillåtet.

Bristen identifierades som att om en användare startar flera pomodoro-timers i en ordning och sedan stoppar dem i omvänd ordning kommer alla sessioner valideras som giltiga för att ge poäng. Logiken för valideringen finns i en enskild metod så ett nytt enhetstest bekräftade bristen innan den avhjälptes.

4.4 Implementation av statistikpanel

Då månadsrankningen tog längre tid än beräknat att införa, bestämdes det i samråd med företaget att en statistikpanel skulle implementeras för att låta användaren se statistik om sitt pomodoro-användande. Statistikpanelen består av två moduler som utvecklades testdrivet, analogt med månadsrankningen.

För att visa statistikpanelen för användaren uppdaterades även klientsidan av applikationen. Resultatet av arbetet med statistikpanelen ses på figur 4.3.



Figur 4.3: Statistikpanelen

4.5 Integrationstester

För att komplettera enhetstesterna skrevs en testsvit av integrationstester för rankningslistorna. Det fanns inga integrationstester för spelifieringsdelen av applikationen så testerna fick skrivas från grunden. Det som var intressant att testa och kunde utföras inom arbetets tidsram var följande:

- Sätta på spelifiering och se användaren i rankningslistan.
- Ge poäng till en användare och se till så att den rankas korrekt efter poäng.
- Skapa användare som delar på en kanbantavla och/eller är från samma företag och se att de syns tillsammans på rankningslistan för vänner och kollegor.

Testerna implementerades med npm-modulen Soda för att programmera ett beteende för Selenium.

5. Resultat och diskussion

I detta kapitel presenteras mina tankar om Node.js från spelifieringen på Kanbanflow och vad andra som använt det tycker om det. Lite fakta för att jämföra Node.js med Java EE finns med i avsnitt 5.3. Slutligen diskuteras resultatet och förslag på vidare studier ges.

5.1 Resultat från arbete

Det första som tar upp tid vid inlärning av Node.js är att förstå JavaScript kod. Eftersom koden har ett asynkront flöde, skiljer den sig en del från traditionell objektorienterad kod. Användningen av anonyma objekt som parametrar, gjorde att det tog längre tid att komma in i arbetet än väntat. Det är även viktigt att ha god kunskap om JavaScript för att använda Node.js, då det finns stora fallgropar i språket som Douglas Crockford ägnat två appendix-kapitel åt i sin bok *JavaScript: The Good Parts* [13].

Vid arbetet med Kanbanflow användes Node.js funktioner relativt lite. Förutom i skriptet som startade servern och i moduluppbyggnaden, användes endast tredjepartsbibliotek och egna moduler. Istället märktes grundtanken med Node.js om att lösa skalbarhetsproblem, genom att tvinga utvecklaren att tänka på det direkt. Detta syntes när rankningslistorna skulle lagras i databasen, om externa handledaren inte hade påpekat att applikationen körs i flera instanser hade flera kopior av samma lista sparats i databasen.

Testdriven utveckling är komfortabel i Node.js. Koden behöver inte kompileras utan testerna kör direkt utan någon väntetid. Men störst fördel är att det är enkelt att skapa anonyma objekt som kan användas för att fingera en kontext så att moduler kan testas enskilt.

5.2 Andras erfarenheter av Node.js

LinkedIn använder Node.js för sina mobilsidor och Shravya Garlapati skriver redan 2011 om vad de anser öka prestandan i en Node.js applikation. Hon tar upp tio punkter där en del kan verka triviala, medan andra, som att de inte använder Node.js för statiska filer som bilder och CSS, och att de inte använder sessioner inte är lika självklara [16].

Senthil Padmanabhan på Ebay skriver om deras första Node.js applikation. Han skriver om svårigheter att övertyga Java-utvecklare att använda JavaScript på serversidan och om hur de implementerat funktioner som det borde finnas en färdig lösning för, t.ex. en modul för loggning [17].

Även PayPal använder Node.js och Jeff Harrell skriver om det i en blogpost på deras ingenjörsblogg. Han beskriver deras erfarenhet som en saga där prestanda förbättrades marginellt jämfört med deras gamla Java-applikation och att utveckling gick mycket fortare. Det intressanta med hans inlägg är att deras motivation till att använda Node.js var att låta utvecklare arbeta med både klient och server istället för att ha separata utvecklarlag för varje del [18].

5.3 Node.js jämförelse mot Java EE

En jämförelse mellan vad teknikerna erbjuder finns i tabell 5.1.

	Node.js	**Java EE**
Arkitektur	Använder en enkeltrådig händelsedriven arkitektur. All kod exekveras asynkront och ingen I/O blockerar exekveringen. För att utnyttja flera processorkärnor behöver flera instanser startas och lastbalanseras.	Använder trådar för att hantera förfrågningar. I/O blockerar en tråd tills en förfrågan har hanterats klart. Utnyttjar flera processorkärnor automatiskt genom att använda fler trådar.
Server	Finns tillgänglig som en modul i standardbiblioteket som startas av applikationen. Konfigureras i programkoden direkt.	Det behövs en applikationsserver som Tomcat eller JBoss för att köra Java EE-applikationer. Servern kan konfiguras genom XML-filer i applikationen.
Infrastruktur	För att hantera tredjepartsmoduler finns verktyget npm. Över 70 000 moduler finns i skrivande stund och företag som bland annat PayPal och Walmart bidrar till moduler i registret.	Java EE har funnits länge och har större utbud av tredjepartsmjukvara. För att bygga och hantera tredjepartsmjukvara används vanligen Maven.

Tabell 5.1: Jämförelse mellan Node.js och Java EE

5.3.1 Test av icke-blockerande I/O

Icke-blockerande I/O är en grund för Node.js som behövde ett enkelt test. En dator med 4 kärnor fungerade som testdator som simulerar 100 samtidiga användare med programmet Apache JMeter. Som måldator användes en enkelkärning Raspberry Pi på samma nätverk som har en exekverande applikation åt gången. Varje applikation testades separat och som tabell 5.2 visar lyckades Node.js hantera fler förfrågningar när databasen användes än vad Java EE-applikationen klarade av.

	Java EE	Node.js
Returnerar texten ”Hello World!”.	276	274
Returnerar texten ”Hello World!” med ett värde hämtad från en MySQL-databas.	40	116

Tabell 5.2: Antal förfrågningar/sekund som hanteras av respektive teknik

5.4 Diskussion

Node.js lever upp till att vara ett minimalistiskt ramverk, vilket märkts genom att det nästan inte använts alls i fallstudien. Istället handlar resultatet mer om miljön runtom tekniken. Framförallt så har mycket av tiden gått till att programmera en applikation i JavaScript och att läsa om vilka metoder som fanns att använda i biblioteken som användes.

Vid en första titt på JavaScript är det ett bristfälligt språk om man har en bakgrund i strikt typade språk som Java. Språket liknar Java i syntaxen men fungerar annorlunda vilket kan orsaka problem. Saker som att variablers definitionsområde inte är inom objekt utan inom funktioner och att jämförelse görs med tre likhetstecken istället för två är något som kan vara problematiskt för en Javaprogrammerare. Men JavaScripts egenskaper har öppnat möjligheter för att skapa modulsystemet som Node.js använder och callback-rutiner som fungerar så bra med den händelsedrivna arkitekturen.

Node.js verkar ha en grundtanke om att många små bitar sätts ihop till en större. En modern server har flera processorkärnor och en Node.js-applikation behöver därför skalas direkt till flera instanser för att utnyttja dessa processorkärnor. Utvecklar tvingas utforma applikationen för att skalas tidigt i uvecklingen. Resultatet blir att applikationer körs i flera mindre instanser och Node.js fokuserar på att effektivisera dessa mindre instanser istället för att bestämma hur applikationen ska skalas.

5.4.1 Framtida studier

Webbutveckling är ett brett ämne och det har kommit upp flera saker att arbeta vidare med. Först så finns det gott om grafer på testdata som visar Node.js som överlägset andra tekniker i prestanda. Dessa grafer saknar dock källor och dokumentation så ett mer utförligt prestandatest hade varit intressant att göra där prestandan kan visas med dokumentation om testet.

Node.js tvingar utvecklare att skala applikationer tidigt och erbjuder en modul i standardbiblioteket för att göra detta över flera processorkärnor. Denna modulen är märkt som experimentell i skrivande stund så det hade varit intressant att utforska denna modulen och titta på andra alternativ för att skala en webbapplikation.

Händelsedriven arkitektur är inget nytt koncept men det har framgått under studien att detta tillsammans med icke-blockerande I/O är effektivt för att utveckla webbapplikationer. Det

skiljer sig från dagens webbapplikationer där trådar används för att hantera nya förfrågningar. Att vidare undersöka vilken effekt detta har på webbappliktioner och att titta på alternativ till Node.js för icke-blockerande I/O är en intressant fråga för framtiden.

5.4.2 Node.js miljö & samhälle

Miljöpåverkan av mjukvara är vanligen att resurser spenderas på virtuella varor istället för fysiska varor, därför anses miljöpåverkan av att välja Node.js framför någon annan teknik försumbar.

Node.js har påvisat nog med stabilitet hos större företag för att tekniken ska kunna användas inom samhället. Ett tänkbart scenario är att använda Node.js för att utveckla tjänster som kan komplettera media som TV och radio för att skicka information till invånare. En sådan tjänst skulle låta invånare få information genom andra medium, exempelvis en mobiltelefon. För att undvika att invadera på invånares privatliv är det lämpligt att låta invånare själva välja att prenumerera på relevant information och vilket medium som ska användas.

6. Slutsats

Det finns anledningar till att föredra Node.js framför andra ramverk. En är hur ramverket löser skalbarhetsproblem, nämligen genom att tvinga utvecklarna till att tänka på skalbarhet direkt från starten. Node.js erbjuder ett standardbibliotek för att underlätta I/O funktioner så dom inte blockerar event-loopen vilket gör att ramverket blir bra på att hantera kommunikation mellan olika källor som klienten och databasen. Eftersom många webbapplikationer idag inte är mer än kommunikation mellan klienten och databasen, finns det ett behov av tekniker som Node.js.

Men den främsta anledningen till att använda Node.js är på grund av tredjepartsmodulerna som finns tillgängliga genom pakethanteraren npm. Utan *moment* för att hantera tid och tidszoner, och *underscore's* många hjälpfunktioner hade arbetet med Kanbanflow handlat om att återuppfinna hjulet istället för att implementera spelifiering. På ett år har antalet moduler som finns i registret ökat från ca 30 000 till 70 0000 [19] och det finns nu nästan lika mycket artefakter som i Maven Central. Med en så livlig öppen-källkodsgrupp runt Node.js finns det gott om användare för att säkra att ramverket kommer leva kvar en lång tid framöver.

Referenser

[1] About Node.js, acc Februari 2014.
http://nodejs.org/about/

[2] Mozilla Developer Network, HTTP, acc Februari 2014
https://developer.mozilla.org/en-US/docs/HTTP/

[3] DocForge, Web Application Framework, acc Mars 2014
http://docforge.com/wiki/Web_application_framework

[4] Node.js, Industry, acc Mars 2014
http://nodejs.org/industry/

[5] Boutell, WWW FAQs : What are MIME types?, acc Mars 2014
http://www.boutell.com/newfaq/definitions/mimetype.html

[6] Node.js, video, acc Mars 2014
http://nodejs.org/video/

[7] Mozilla Developer Network, HTML, acc Mars 2014
https://developer.mozilla.org/en-US/docs/Web/HTML

[8] Mozilla Developer Network, CSS, acc Mars 2014
https://developer.mozilla.org/en-US/docs/Web/CSS

[9] Antonio Goncalves, Beginning Java EE 6 Platform with GlassFish 3
ISBN: 978-1-4302-2889-9

[10] Ian Jacobs, W3C och Norman Walsh, Sun Microsystems, Architecture of the World Wide Web, acc Mars 2014
http://www.w3.org/TR/webarch/#identification

[11] Hector Garcia-Molina, Jeffrey D. Ullman, Jennifer Widom, DATABASE SYSTEMS The Complete Book
ISBN: 978-0-13-135428-9

[12] Eric Redmond, Jim R.Wilson, Seven Databases in Seven Weeks
ISBN: 978-1-93435-692-0

[13] Douglas Crockford, JavaScript: The Good Parts
ISBN: 978-0-596-51774-8

[14] Christian Johansen, Test-Driven JavaScript Development
ISBN: 978-0-321-68391-5

[15] Ryan Dahl, Slides from JSConf 2009, acc May 2014
http://s3.amazonaws.com/four.livejournal/20091117/jsconf.pdf

[16] Shravya Garlapati, Blazing fast node.js, acc june 2014
http://engineering.linkedin.com/nodejs/blazing-fast-nodejs-10-performance-tips-linkedin-mobile

[17] Senthil Padmanabhan, How We Built Ebay's First Node.js Application, acc june 2014
http://www.ebaytechblog.com/2013/05/17/how-we-built-ebays-first-node-js-application/

[18] Jeff Harell, Node.js at PayPal, acc june 2014
https://www.paypal-engineering.com/2013/11/22/node-js-at-paypal/

[19] Module Counts, acc June 2014
http://modulecounts.com/