Projekt – Email Service

Nätverksprogrammering, ID1212

Evan Saboo
saboo@kth.se

2018-01-08

https://github.com/ZpeedX/Network-Programming/tree/master/Project/MailService

1 Introduktion

Målet med projektet är att skapa en större applikation som följer MVC arkitekturen och använder ett kommunikationsparadigm som inte har tagits upp i kursen.

Min applikation är en enkel klient-server mail tjänst. En användare kan registrera sig, logga in, logga ut, skicka ny mail till en annan användare, lista alla meddelanden och ta bort meddelanden.
En inloggad användare blir notifierad om ett nytt mail har inkommit.

2 Litteraturstudie

Uppgiften utfördes med hjälp av dessa källor:

* Jag gick till oracles hemisda för att lära mig om hur man kan skapa en enkel websocket applikation med java EE och JavaScript.[1]

* Jag använde StackOverflow hemsidan för att få hjälp med problem och för att få snabb information på några java syntaxer. [2]

* Eftersom jag använde jQurey för att implementera JavaScript funktionaliteter använde jag mig av jQureys API dokumentation för att hitta jQurey syntaxer.[3]

* Java EE dokumentationshemsidan var också hjälpsam för att hitta Java EE syntaxförklaringar.[4]

* Applikation Server föreläsningarna hjälpte mig förstå Java EE som helhet och dess funktionaliteter.

[1] http://www.oracle.com/webfolder/technetwork/tutorials/obe/java/HomeWebsocket/WebsocketHome.html
[2] https://stackoverflow.com/
[3] http://api.jquery.com/
[4] https://javaee.github.io/javaee-spec/javadocs/

3 Metod

För att komma igång med projektet gick jag först genom hur websockets fungerade och hur man implementerar den i java EE. Det var ganska lätt att förstå hur man skickade och tog emot data via websockets men hanteringen av sessioner var lite komplicerad.
Jag valde websockets eftersom det är ett nytt kommunikationsparadim som jag inte har lärt mig och den passar perfekt till applikationen jag har utvecklat.
Med websockets kan klienten och servern när som helst skicka meddelanden till varandra via en öppen anslutning, och de kan när som helst stänga anslutningen. Klienter kopplar vanligtvis bara till en server medan serven accepterar anslutningar från flera klienter.

HTML och CSS var inget stort problem för mig då jag har tidigare erfarenheter med dem men JavaScript behövde jag gå igenom eftersom vi inte har gått igenom språket ganska mycket.
För att förenkla HTML-, DOM- och CSS-modifikation användes JavaScript-biblioteket jQurey.
Den tar många vanliga uppgifter som kräver många rader med JavaScript-kod för att utföra, och omsluter dem till metoder som man kan kalla med en enda rad kod.

Jag använde också front-end-ramverket Bootstrap för att designa min hemsida med deras HTML- och CSS-baserade designmallar för vanliga användargränssnittskomponenter tex. Former, Knappar, Navigationer, Dropdowns, Alerts, Modals och många andra komponenter. Med Bootstrap kunde jag implementera responsiv layout mycket lättare än vanligt för att applikationen ska kunna anpassa sig till flera plattformar.

4 Resultat

Min applikation går ut på att kunna logga in som användare och skicka meddelanden till andra registrerade användare.
Figur 1 visar användargränssnittet för inloggningen. När man klickar på login knappen skickas användarnamnet och lösenordet som JSON format till serven via websockets.
Serven skickar sedan tillbaka ett svar till klienten som berättar (genom boolean variabel) om inloggningen lyckades eller inte. Om inloggningen misslyckades visas en förklaring (felmeddelande) på misslyckandet, annars om inloggningen lyckas skickas man vidare till inkorgsidan.
Registreringssidan har samma design som inloggningssidan men istället för två fält finns det tre, en för användarnamnet, en för lösenordet och sista för att skriva in samma lösenord igen.

I Figur 2 visas inkorgsidan. Där kan man se alla meddelanden man har fått och alla man har skickat. Om man trycker på ett specifikt meddelande kommer en modal ruta upp som visar: ämnet på meddelandet, datum den skickades, avsändares-, mottagarens användarnamn och meddelandets innehåll (Figur 3).
I inkorgsidan kan man också ta bort eller svara på ett meddelande. Vid borttagning av ett meddelande skickas meddelandets id till serven genom websockets. Serven kollar först om meddelandet verkligen tillhör användaren innan den tas bort.
Bara en instans av meddelandet finns i databasen och för att både avsändare och mottagare ska kunna hantera meddelandet så finns det två till kolumner (boolean) i databasen som berättar om meddelandet ska visas för dem.
Vid borttagning av meddelandet ändras en av kolumnens värde till ”false” vilket säger att meddelandet inte ska visas till en av ägarna längre. När både kolumner blir ”False” kommer meddelandet tas bort från databasen.

För att skicka ett nytt meddelande klickar man först på ”compose” knappen så att en modal ruta upp (Figur 4) där man kan fylla i de nödvändiga fälten för att skicka meddelandet. När serven tar emot det nya meddelandet kollas först om mottagaren är en registrerad användare innan meddelandet läggs in i databasen. Serven skickar sedan tillbaka bekräftelse till användaren om meddelandet skickades eller inte.

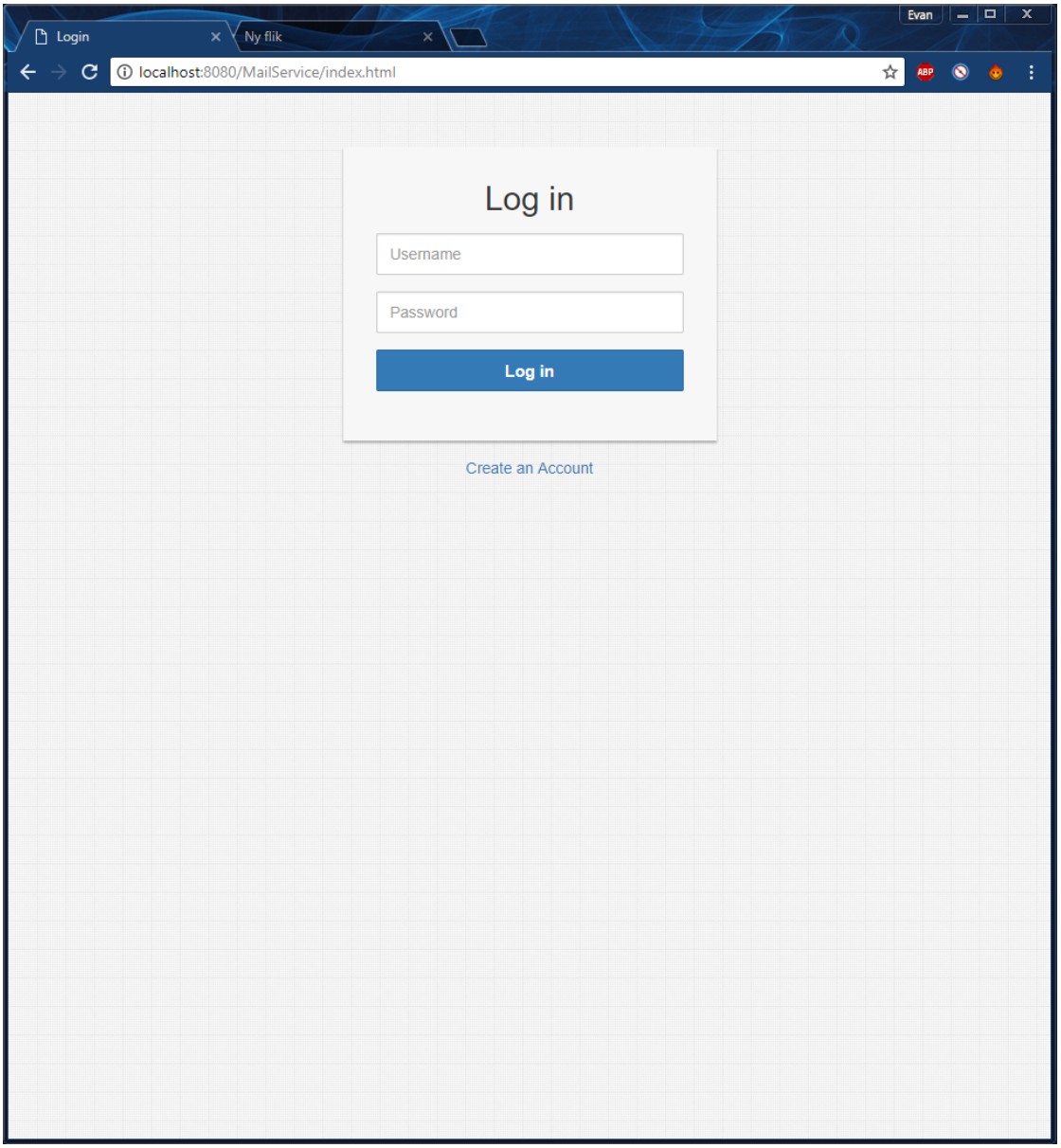
I figur 2 kan man också sen en liten notis som berättar att ett nytt meddelande har tagits emot från en annan användare. Detta görs när serven tar emot ett nytt meddelande, då skickas en notis till mottagaren om hen är inloggad.
När mottagaren får notis från serven uppdateras hens inkorg automatisk vilket gör så

att det nya meddelandet hämtas till inkorgen utan att mottagaren behöver uppdatera manuellt.
För att kunna skicka notisen från serven till klienten sparas alla aktiva sessioner som användaren kommunicerar med. På så sett kan notisen skickas till alla enheter användaren är inloggad och aktiv på.

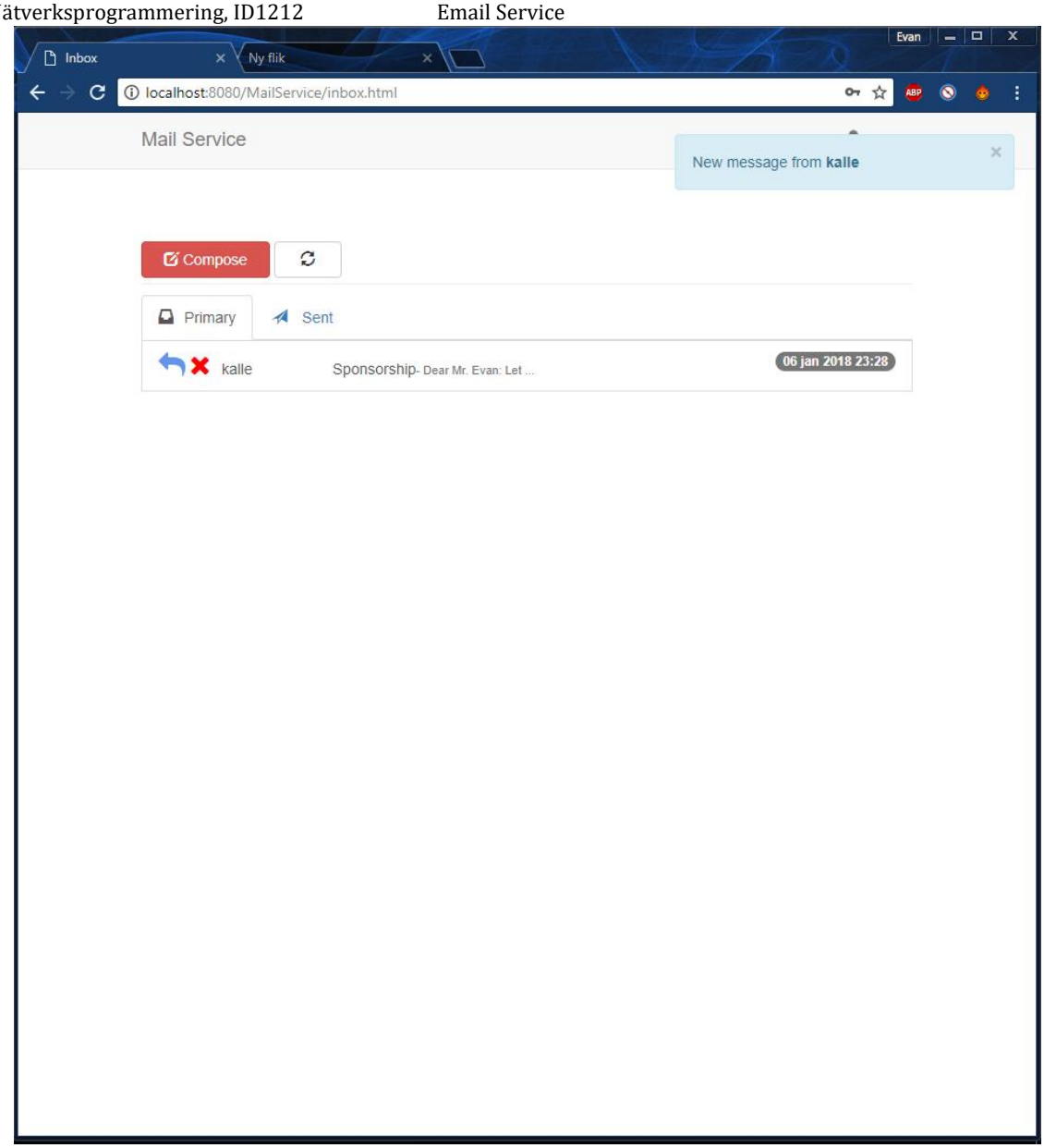
Arkitektur
Serven har fyra lager, net, controller, model och integration.
Net lagret hanterar kommunikation med alla klienter och håller koll på vilka användare som är inloggade. Om Net lagret vill ha något från databasen måste den först kalla på metoder från controller som sedan kallar på metoder från integration för att komma åt databasen. Model lagret har två klasser som används för att spara objekt i databasen genom integration lagret.

Klienten har inte en riktigt strukturerad arkitektur då det mesta av klientkoden hanterar användargränssnittet, men klienten har en Net lager som innehåller kod för att koppla och skicka data till serven.

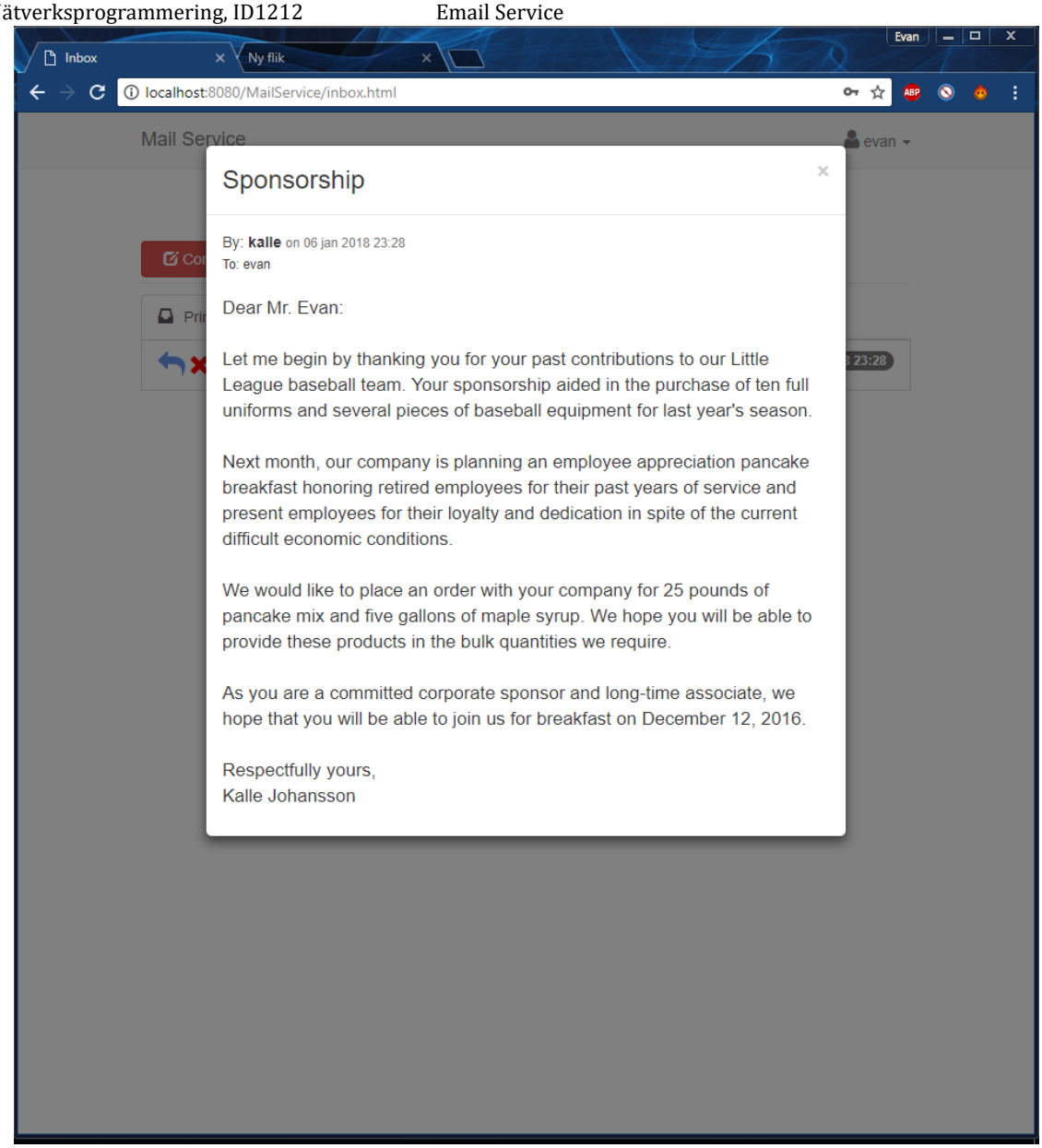
Felhantering
Applikationen har en html sida för att visa orsaken till ett fel som kan hända (Figur 5).
Applikationen hanterar flera problem såsom anslutningsfel, användarinmatningsfel, Interna server fel och skadlig html manipulering.
Det finns också felhantering för att kunna komma åt en sida som man egentligen inte ska kunna komma åt tex. om man inte är inloggad men man försöker ändå gå till inkorgsidan.
Om kommunaktionen mellan serven och klienten avbryts kommer användaren att bli notifierad om det.



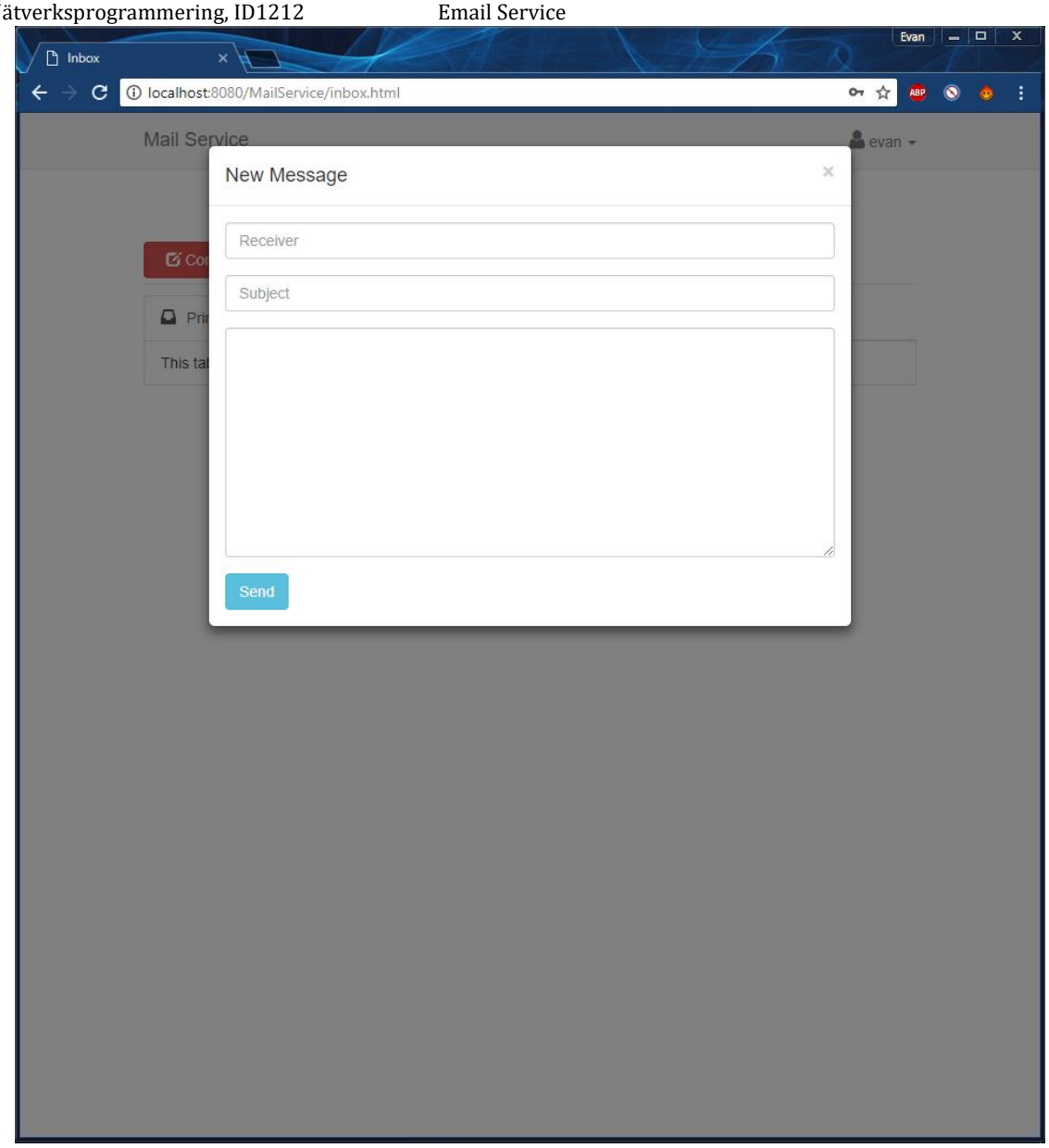
Figur 1: Inloggningssidan med felmeddelande



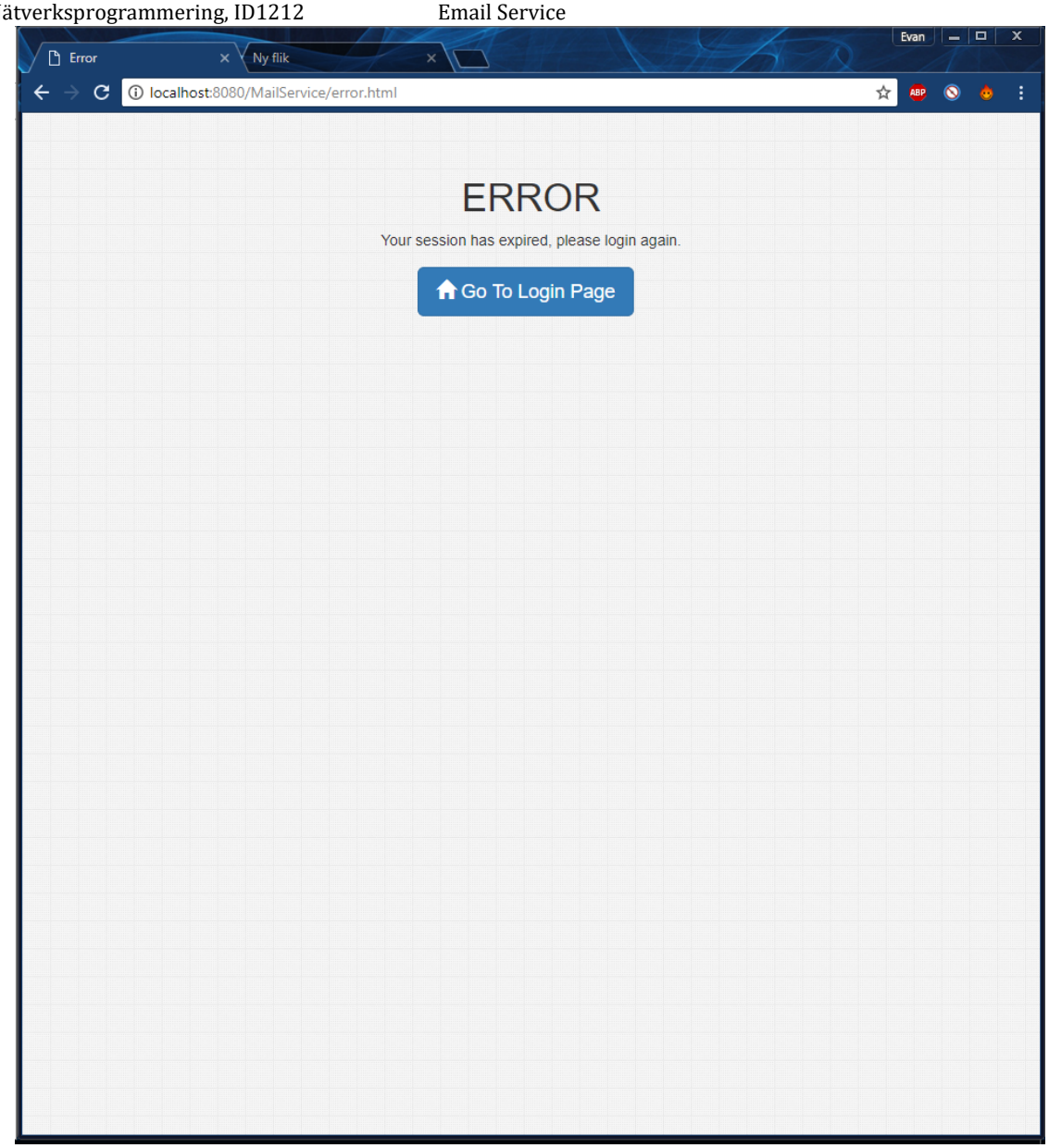
Figur 2: Inkorgens användargränssnitt



Figur 3: Ett popupfönster som visar hela innehållet i ett meddelande.



Figur 4: Modal ruta för att skicka nytt meddelande.



Figur 5: Om något fel oväntat inträffar blir man omdirigerad till ”error” sidan.

5 Diskussion

Kraven nedan uppfylldes:

- Serversidan följer MVC arkitekturen för bättre struktur över logiken och användargränssnittet.
- Jag använde websockets för att serven och klienten ska kunna kommunicera med varandra.
- Serven tar hand om all data och logik medan klienten hanterar användargränssnittet.
- Användargränssnittet acceptabel design och är informativ.
- Jag har försökt göra felhantering så informativt som möjligt genom att visa till användaren vad felet om något fel eller kommunikationsfel sker.

Projektet var inte svår att utföra, det vara bara felsökande av JavaScript fel som var jobbig och tog mest tid.
Problemet med JavaScript är att den inte ger bra förklaring till varför något inte fungerar när man testar den. Ett exempel var när jag ville lägga text i en div element men av någon anledning fungerade inte det med den metoden jag använde. Det var svårt att hitta orsaken till felet eftersom webbläsarens konsol gav inga konkreta förklaringar så jag var tvungen att googla problemet tills jag hittade något vettigt som kunde lösa den.

Problem som kom upp i Java EE och paraya serven var också svårt att hitta orsaken till. Man kunde inte debugga med Netbeans och ibland visste jag inte vart felet inträffades eftersom paraya gav ingen bra förklaring till problemet.

Implementeringen av websockets var ganska lätt att utföra. Man behövde inte mycket kunskap om websockets för att förstå hur det fungerade. Jag hade inte stora problem med att implementera websockets eftersom kommunikationen mellan serven och klienten fungerade hela tiden när jag testade den.

Som helhet känner jag mig nöjd med applikationen. Jag kunde ha implementerat flera funktioner men jag tycker att funktionerna som finns i applikationen borde vara nog för att bevisa att jag har en bra förståelse om Java EE, JavaScript och Websockets.

6 Kommentar om kursen

Det tog 45–50 timmar att utveckla applikationen. Rapporten tog ungefär 4 timmar att skriva.