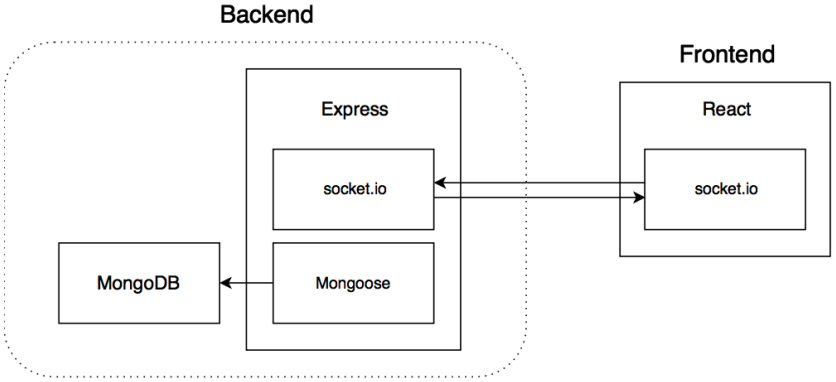
!Chatapp

Henrik Numé, Johanna Torbjörnsson, Henry Ly & Pontus Stjernström

!Chatapp (Uttalas *“Not Shut up”*) är en chatt-applikation byggt i MERN-stacken, vilket står får MongoDB, Express, React och NodeJS. Den övergripande arkitekturen syns i bilden nedan.

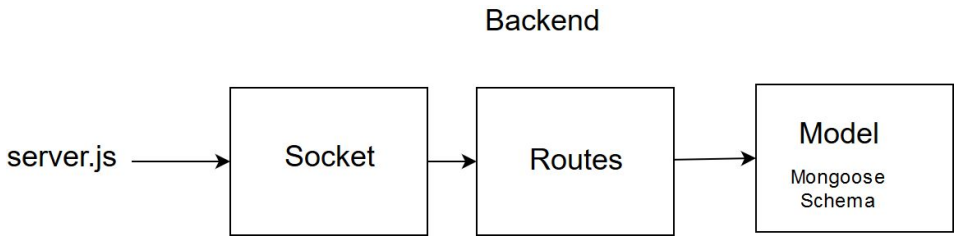


Backend och frontend kan köras på samma maskin men de måste ligga på två olika servrar. Frontend är en s.k “client side app” och består endast av statiska filer som kan förses från en vanlig HTTP-server. Backend kräver en Nodejs-miljö. Det är endast specifikt vår frontend som sätter upp anslutningar till vår backend och det är endast frontend som hämtas hem via en webbläsare av användaren. Mer ingående förklaringar av backend- och frontend-delarna och hur de kommunicerar ges nedan.

Arkitektur för backend

Vår Backend är baserad på Express-ramverket och använder en MongoDB databas med Mongoose som ORM. Det som kännetecknar MongoDB är att det är en NoSQL databas där all data sparas i Javascript-liknande objekt. Detta gör att hela applikations-stacken kan skrivas i endast Javascript.

Eftersom kommunikationen mellan klienterna ska ske i realtid måste det vara möjligt att pusha ny data till alla klienter. För att uppnå detta behöver vi använda websockets. Det socket API som har valts ut till projektet är Socket.io vilket är populärt och väldokumenterat. Backenden och databasen hanteras genom att skicka olika events som socket.io lyssnar på. Socket.io kommer därefter att broadcasta events till alla klienter som lyssnar på det eventet i *routes* paketet. Anslutningen av klienter till en server via socket sker i paketet *socket*.



server.js

Detta är startpunkten för backend-applikationen. Här startar vi en Express-server och upprättar en förbindelse till databasen med hjälp av Mongoose. Därefter startar vi även våra hanterare för websockets.

socket

Detta paket sätter up vår socket.io server m.h.a Express-applikationen. Socket:en börjar här lyssna på att klienter ska ansluta. När detta händer skapas en ny socket för klienten, denna laddas sedan med en uppsättning av events som den ska lyssna på. Dessa är definierade i paketet *routes*.

routes

Här hanteras alla events som servern tar emot från klienterna. Dessa events består av en ett event namn och en “body” med json-data. Event-namnen är strängar som är definierade som konstanter i *src/socket/events.js*. Dessa events används också av vår frontend app och är i princip hela vårt API.

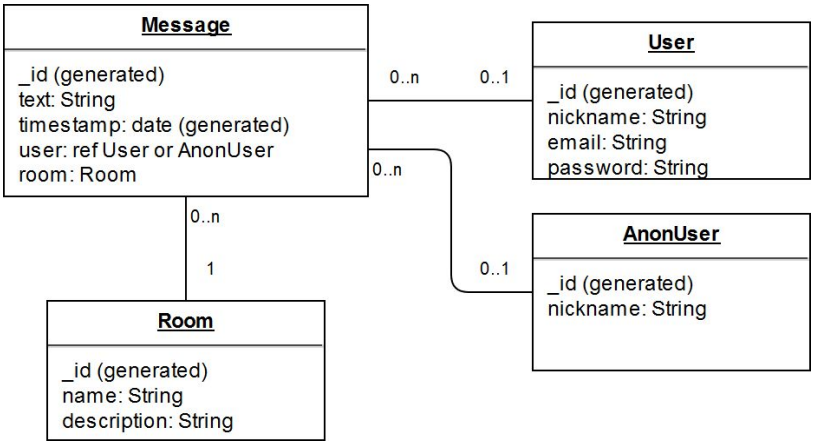
I detta paket sker även lite authentication och accesscontrol. När en User registreras (event: USER_REGISTER) så används bcrypt för att hasha lösenordet innan det sparas i databasen. Salt värden hanteras automatiskt av bcrypt och kräver därför inte att vår User modell sparar detta.

När en användare sedan försöker loggar in (event: USER_LOGIN) så jämförs en hash av det angivna lösenord med det hashade värdet i databasen. Om lösenordet stämmer så returneras en jwt-token innehållandes bl.a användarens id (userId).

När en användare sedan försöker göra en begränsad operation som att t.ex. uppdatera användarinformation (event: USER_UPDATE) så måste en giltig token skickas med i eventets body. I vårt fall kontrolleras både att den angivna token är giltig samt att den “User” som skall modifieras har samma id som det userId som finns sparat inuti token.

models

Modellerna som används är de följande: User, AnonUser, Messages och Rooms. Dessa är definierade i paketet *models* där varje entitet ärver ett Mongoose.schema. Detta gör att alla modeller från databasen är tillgängliga genom Mongoose vilket i sin tur tillhandahåller alla vanliga CRUD-operationer. Följande diagram visar alla modeller och deras relationer.



Arkitektur för frontend

Vår frontend är byggt i ramverket Reactjs som är ett modernt ramverk för frontendutveckling på webben baserat på Javascript. Hela applikationen byggs i ren Javascript där själva strukturen (HTML-liknande) skrivs i något som kallas JSX, som enligt React själva varken är strängar eller HTML. Rent syntaktiskt liknar det dock HTML väldigt mycket. I JSX kan man använda sig av “mustache”-liknande syntax för att köra JavaScript direkt i frontend.

```
function formatName(user) {
  return user.firstName + ' ' + user.lastName;
}

const user = {
  firstName: 'Harper',
  lastName: 'Perez'
};

const element = (
  <h1>
    Hello, {formatName(user)}!
  </h1>
);

ReactDOM.render(
  element,
  document.getElementById('root')
);
```

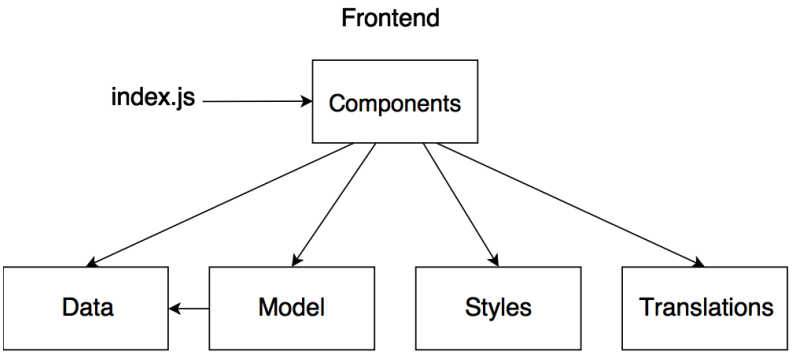
Exempel på JSX där användarens namn formateras av funktionen *formatUser.*
Källa: https://reactjs.org/docs/introducing-jsx.html

All kod skriven i Reactjs transpileras sedan ner till ett få statiska filer, som vilken annan hemsida som helst. Dessa kan sedan kommas åt från en vanlig http-server som Apache eller liknande. Detta skiljer sig alltså från backend, som är en körande applikation.

Strukturen i Reactjs bygger på att man använder komponenter snarare än mer klassiskt MVC. Varje komponent i gränssnittet ska kunna bytas ut till en annan enkelt och smidigt. Vi har byggt upp vårt gränssnitt i super- och subkomponenter så att t.ex. vår *frame-view* innehåller alla andra komponenter, vår *lobby-view* innehåller *chat-window-view* o.s.v.

Utöver de rent grafiska komponenterna, har vi fyra “paket” som heter *model, data, styles* och *translations*. I *model* finns *User* som är en singleton för att hantera användarens inloggning mellan olika sidbyten. Detta är möjligt då Reactjs skapar en single-page application som bara bygger om DOM-trädet hela tiden istället för att skicka en helt ny HTML-fil när användaren gör en förfrågan efter en ny sida. I *data* har vi våra websocketbaserade

anslutningar till vår backend. Detta blir ett slags API som de grafiska mer “dumma” komponenterna kan använda för att undvika att blanda in lågnivåkod i gränssnittet. I *styles* ligger css-filer som lägger till stilar på komponenterna, och i *translations* ligger json-filer med översättningar till engelska och svenska av alla ord som syns för användaren. Orden översätts till det språk som är valt i webbläsaren med hjälp av ett Reactjs-bibliotek.



Sockets och chattfunktionalitet

Vi valde att basera vår chattfunktion på de websocketbaserade biblioteket socket.io. Det är mycket enkelt att använda och det finns mycket dokumentation. Den största fördelen över t.ex. ett REST-API är att man slipper s.k. *polling*, d.v.s att klienter måste regelbundet göra nya förfrågningar för att se om det finns data att hämta. Med socket.io kan vi istället utsända (broadcast) ett meddelande till alla lyssnande klienter när ett meddelande tagits emot i ett chattrum. Detta blir både mer effektivt och lättare att förstå sig på.

Till en början använde vi endast socket.io för dessa operationer, alltså att skicka och ta emot meddelanden mellan klienter. Till andra operationer, såsom att skapa rum och hämta arkiverade meddelanden från databasen använde vi ett vanligt REST-API. Vi tog senare beslutet att ersätta alla REST-delar med socket.io p.g.a dess enkelhet samt för att vara konsekventa och inte använda två kommunikationsmedel mellan back- och frontend. Det visade sig sedan att det antagligen hade varit enklare att hålla oss till vårt initiala upplägg med både REST och socket.io, då detta verkar vara relativt vanligt i t.ex. chattapplikationer.

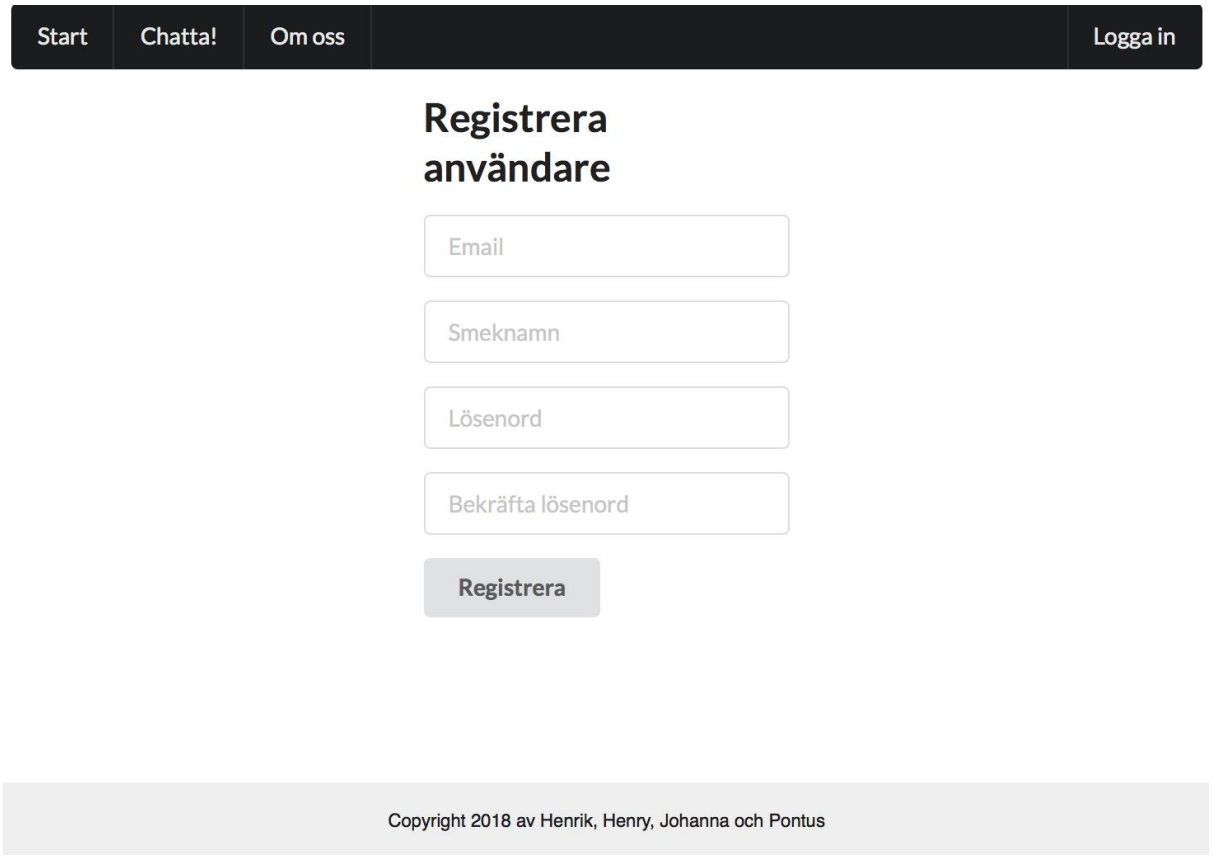
Use cases

Som en användare kan jag:

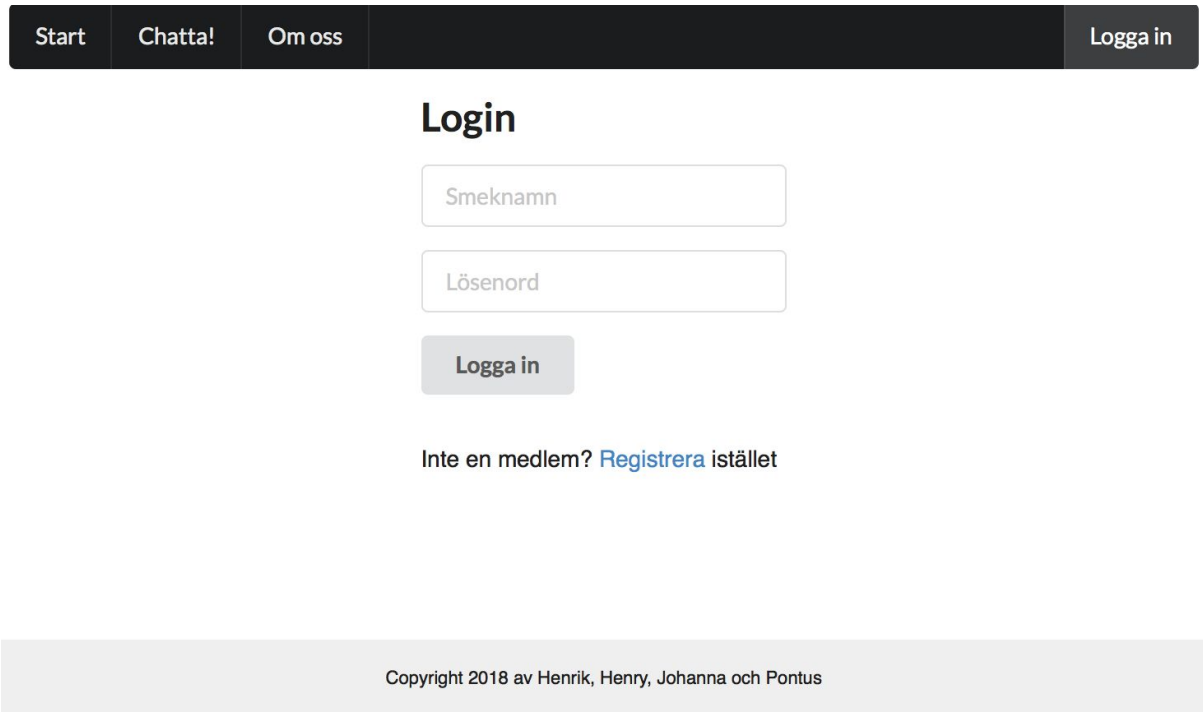
1. Skapa ett chattrum
2. Bli notifierad när andra användare skapar ett nytt chattrum
3. Skicka meddelanden i godtyckligt chattrum som en anonym användare
4. Ej skriva fula engelska ord
5. Ta emot meddelanden från godtyckligt chattrum
6. Skapa ett konto med ett unikt e-postkonto
7. Ej skapa ett konto om lösenorden ej överensstämmer
8. Logga in på mitt konto med mitt användarnamn och lösenord
9. Växla mellan olika användarskapade chattrum

10. Följa vilken användare som har skickat vad
11. Klicka på en användare och se deras detaljerade information
12. Ändra mina användarinställningar (nickname och about)
13. Ej ändra mina användarinställningar som utloggad
14. Generera en slumpmässig avatar, unik för mitt konto
15. Växla mellan svenska eller engelska, beroende på mina webbläsarinställningar
16. Logga ut från mitt konto

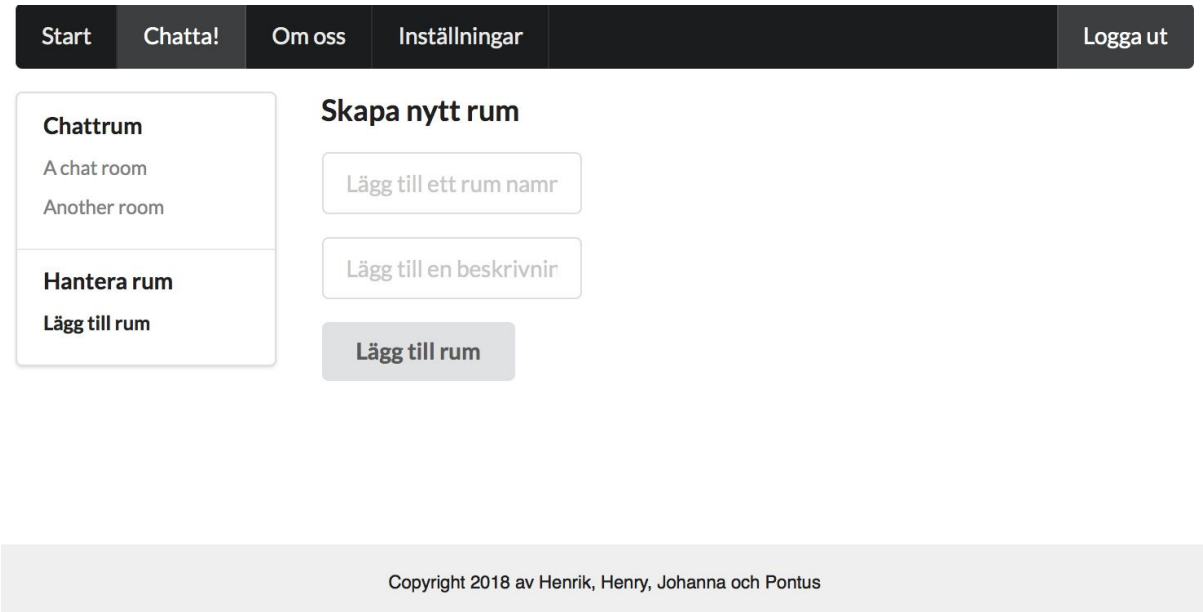
Screenshots



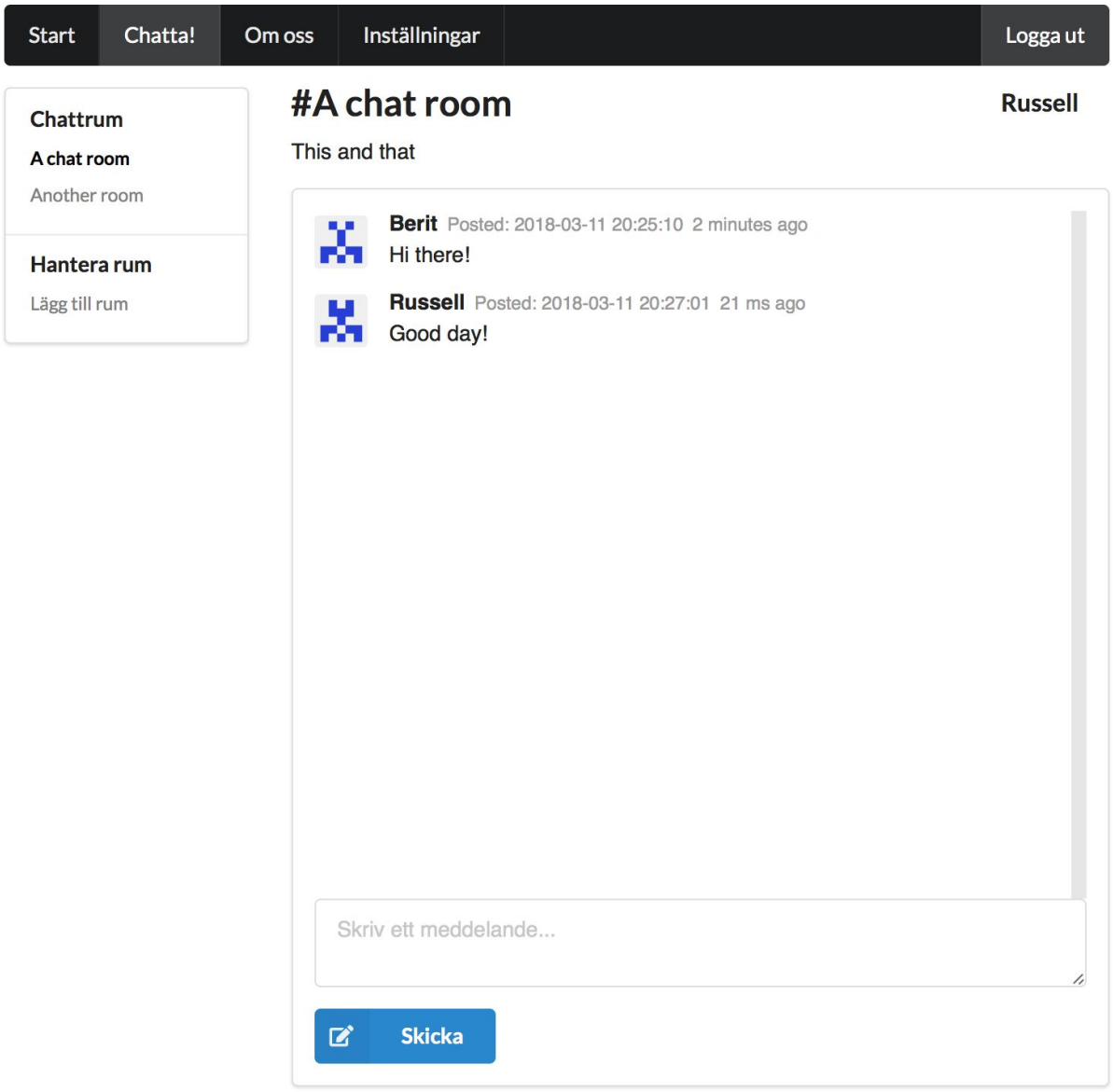
Skapa ett konto



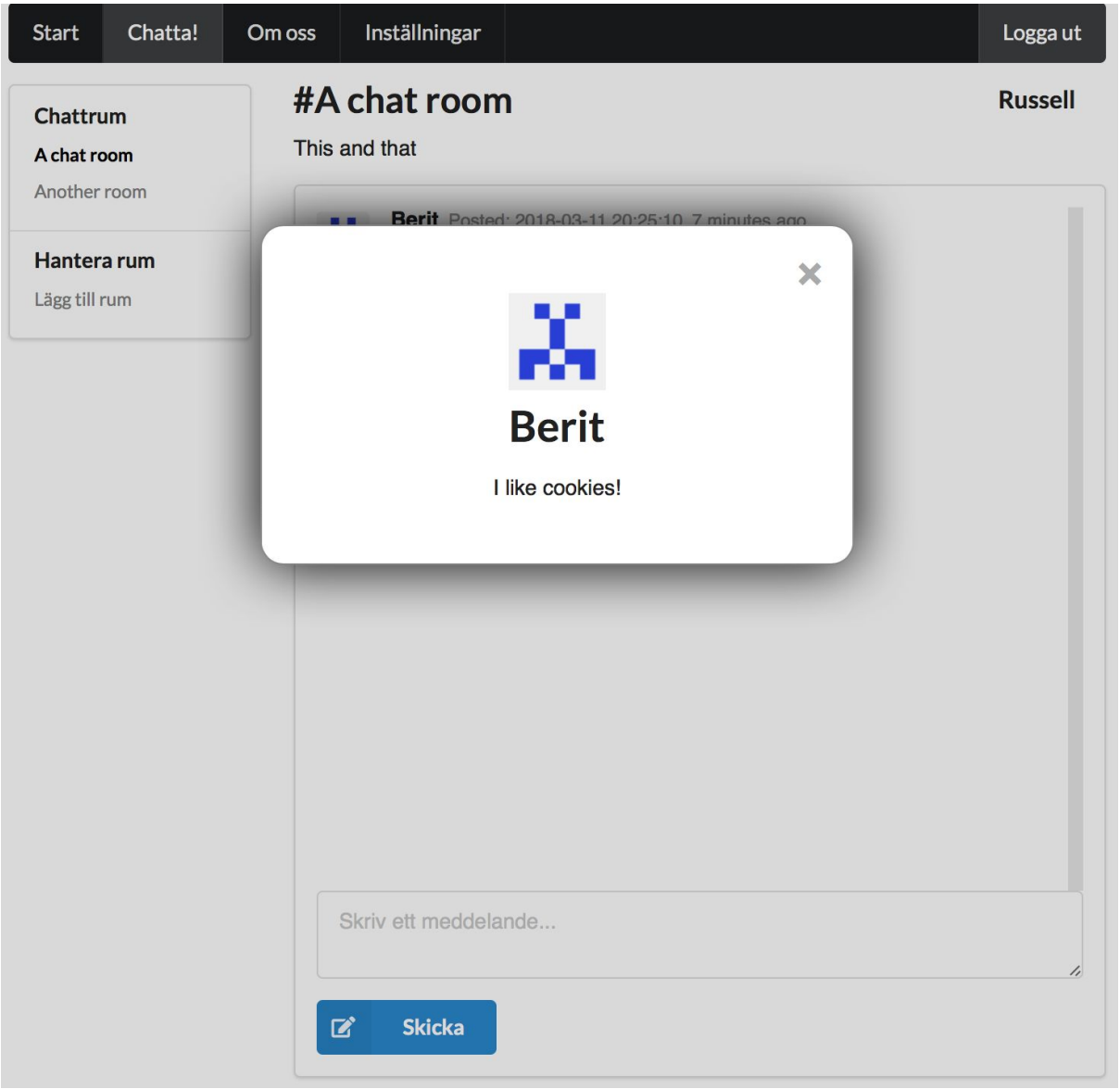
Logga in



Skapa ett chattrum



Skicka och ta emot meddelanden i chattrum



Klicka på en användare och se deras detaljerade information