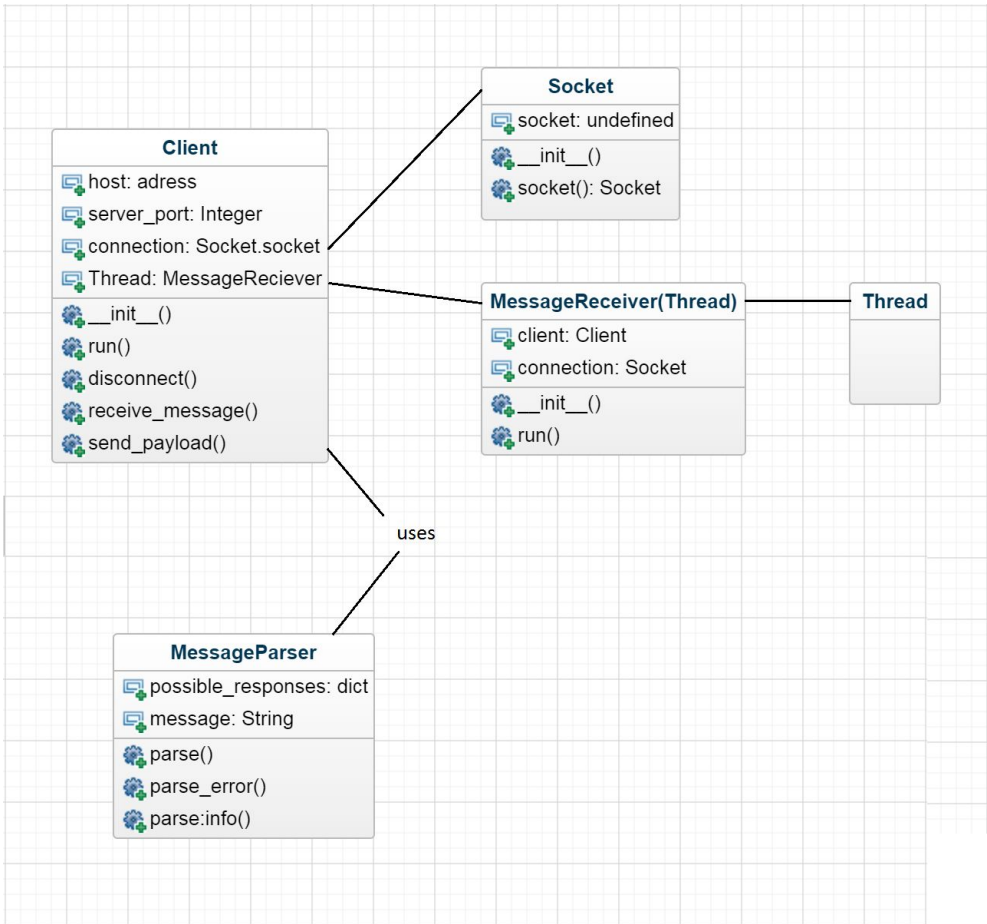
TTM4100 Prosjekt

Vi har valgt å tolke denne oppgaven slik at vi lager ett chatterom. Altså blir alt som sendes fra en person til serveren, send til alle andre koblet på samme server. Idéen er altså at alle som kobler seg opp mot serveren, vil kunne delta i en felles samtale, frem til man kobler seg av. Serveren skal altså fungere som et enkelt chatterom.

Programmet skal oppfylle enkelte krav. Det skal kunne håndteres av hvem som helst, derfor må vi gjøre det så enkelt og oversiktlig som mulig, så både en med og uten programmeringserfaring kan bruke chatterommet. Siden vi skal kunne snakke med flere personer samtidig, må serveren vår kunne håndtere flere klienter om gangen. Vi må også sortere meldingene etter tidspunktet de ble sendt, slik at alle beskjedene kommer i riktig rekkefølge.
For at vi skal ha en sikker kobling med et 3-way-handshake, velger vi å bruke TCP. TCP setter først opp en forbindelse, før den sender beskjeden, i motsetning til UDP som bare sender ut beskjeden med en gang. Med TCP sender også serveren ACKs tilbake, slik at klienten vet beskjeden at har kommet frem.

Klient:

Klienten vår skal kunne koble seg til serveren, klienten sender en string som brukernavn, hvor serven bruker dette til å identifisere klienten. Som en klient må vi kjøre to objektklasser samtidig: MessageReciever, som lytter til de beskjedene serveren sender, og Klientklassen selv, som lytter etter bruker-input. Vi har altså to forskjellige tråder, en for å sende beskjeder og en for å motta. Under har vi laget et klassediagram på hvordan klassen vår skal fungere:



Klassene til klienten fungerer på følgende måte:

Client:
Vår hovedklasse. Klassen lagrer informasjon om serveren og håndterer alle handlinger opp mot denne.

Thread:
Lar oss bruke multithreading, slik at vi kan håndtere både brukerinput og beskjeder fra serveren samtidig.

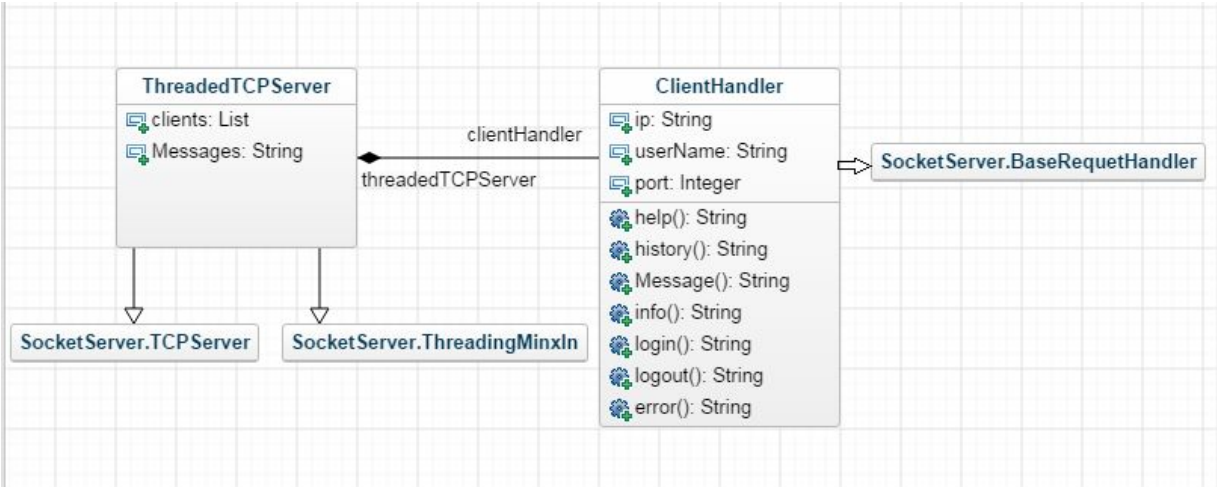
MessageReciever:
Denne utvider *Thread* klassen, den kan derfor kjøres på samme måte som en *Thread.*
Denne klassen lytter etter beskjeder fra serveren og håndterer disse ved å sende den til klienten.

MessageParser:
Denne klassen håndterer beskjeder som skal gå fra klienten til serveren.

Socket:
Benyttes til å lage en TCP connection til serveren.

Server:

Serveren sitter alltid og venter på klienter som kobler seg til. Når den får en ny klient vil den starte en ny ClientHandler for den klienten og legge til handleren i en liste med alle handlerene. ClientHandleren lagrer informasjonen til klienten og utfører funksjoner ut ifra hva som står i meldinger fra klienten. Hver gang handleren får en melding som skal sendes videre blir den sendt til serveren som traverserer listen av handlere og sender meldingen videre til de andre klientene som er tilkoblet.



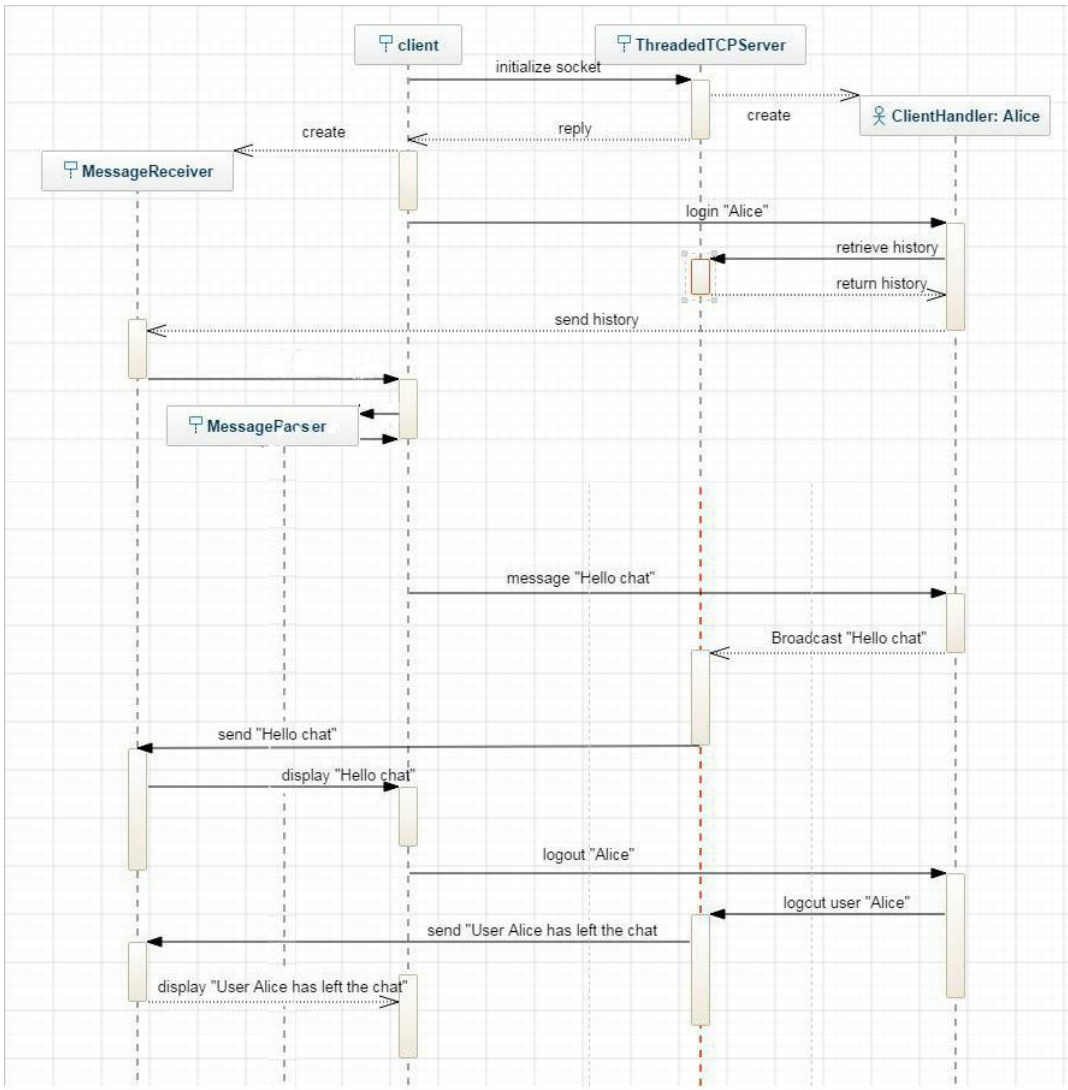
Klassene til serveren fungerer på følgende måte:

ThreadedTCPServer:
Utvider server, som finnes i python biblioteket. Denne klassen håndterer alle oppkoblinger mellom serveren og klientene. Klassen lytter konstant, og oppretter/kjører ett nytt ClientHandler objekt for hver nye oppkobling. Lagrer også en liste med alle klientene.

ClientHandler:
Denne klassen håndterer all kommunikasjon i mellom en gitt klient og serveren. Den håndterer også logikk som skal utføres med grunnlag i klientens beskjeder.

Litt om JSON:
JSON er et tekstformat som er praktisk å bruke fordi det er uavhengig av programmeringsspråk og det er et format som både mennesker og maskiner forstår.

Sekvensdiagram:



I oppgaven vil vi benytte følgende verktøy:

PyCharm/Eclipse:
Verktøyene som vi bruker til programmering.

Trello:
Benyttes for å håndtere TODOs på tevers av alle i gruppen. Brukes for å fordele arbeid, i stedet for layering.

GIT:
Benyttes slik at vi kan sammarbeide med kode.

Google Docs:
Benyttes til dokumenter som skal lages. Vi valgte denne plattformen ettersom den har god integrasjon for samarbeid.

Deltagere:
Sara Waaler Eriksen
Ingrid Sørdal Volden
Haakon Kristian Hagen
Jens-Andreas Hanssen Rensaa
Finn Julius Stephansen-Smith