

INF142 Obligatorisk Oppgave 1

Nikolai Gangstø

Februar 2025

Innhold

a) Applikasjonsnivåprotokoll	2
1.1 Format	2
1.2 Meldinger	2
1.2.1 Post problem (valg 1. i client.py)	2
1.2.2 Spør etter problem (valg 2. i client.py)	3
1.2.3 Vis en problemformulering (valg 3. i client.py)	3
1.2.4 Vis alternativer (valg 4. i client.py)	4
1.2.5 Stem på alternativ (valg 5. i client.py)	4
1.2.6 Vis stemmer (valg 6. i client.py)	5
b) Tilstandsløs vs tilstandsfull og TCP vs UDP	5
2.1 Tilstandsløs vs tilstandsfull	5
2.2 TCP vs UDP	5
c) Implementering av klient-server-applikasjonen i Python	5
3.1 Se server.py og client.py.	5

a) Applikasjonsnivåprotokoll

1.1 Format

Å bruke JSON-format virker som et bra utgangspunkt, siden man da kan sende meldinger i JSON og lagre dem i en JSON-liste. Dette gjør det også mulig å bruke en Python-dictionary som "database".

```
database = {
    100: {
        "tittel": "Hva skal jeg ha til middag?",
        "alternativ": {
            "Pølse": 0, "Hamburger": 0, "Pizza": 0
        }
    }
}
```

1.2 Meldinger

1.2.1 Post problem (valg 1. i client.py)

Klienten oppgir tittel og alternativer, og får tilbake en problemID. Mens problemet har blitt lagret i databasen hos tjeneren.

- Klient → Server:

```
{
    "kommando": 1,
    "tittel": "tittel tekst",
    "alternativ": ["Alternativ 1", "Alternativ 2"]
}
```

- Server → Klient:

```
{
    "melding": "Problem lagt til i database",
    "problemID": 100
}
```

1.2.2 Spør etter problem (valg 2. i client.py)

Klienten velger valg 2 i prompt, og så sendes kommandoen "2" som til tjeneren, og tjeneren svarer med alle problemene som ligger på serveren.

- Klient → Server:

```
{
  "kommando": 2
}
```

- Server → Klient:

```
{
  100: {
    "tittel": "Hva skal jeg ha til middag?",
    "alternativ": {
      "Pølse": 0, "Hamburger": 0, "Pizza": 0
    }
  },
  (...)
}
```

1.2.3 Vis en problemformulering (valg 3. i client.py)

Klienten sender først kommando "3", så problemID i samme tekststreng, og for tilbake info for dette problemet.

- Klient → Server:

```
{
  "kommando": 3,
  "problemID": 100
}
```

- Server → Klient (Hvis funnet):

```
{
  "tittel": "Hva skal jeg ha til middag?",
  "alternativ": ["Pølse", "Hamburger", "Pizza"]
}
```

- Server → Klient (Hvis ikke funnet):

```
{
  "melding": "Ønsket problem finnes ikke"
}
```

1.2.4 Vis alternativer (valg 4. i client.py)

Klient sender "4" og problemID til ønsket problem og får tilbake alternativene som tilhører dette problemet.

- Klient → Server:

```
{  
  "kommando": 4,  
  "problemID": 100  
}
```

- Server → Klient:

```
{  
  "tittel": "Hva skal jeg ha til middag?",  
  "alternativ": ["Pølse", "Hamburger", "Pizza"]  
}
```

1.2.5 Stem på alternativ (valg 5. i client.py)

Klienten sender "5" + "problemID" + sin ønsket stemme til tjener og får tilbake at stemmen er mottatt, og oppdaterte stemmer på problemet.

- Klient → Server:

```
{  
  "kommando": 5,  
  "problemID": 100,  
  "stemme": "Pølse"  
}
```

- Server → Klient:

```
{  
  "melding": "Stemme mottatt",  
  "tittel": "Hva skal jeg ha til middag?",  
  "alternativ": {  
    "Pølse": 1, "Hamburger": 0, "Pizza": 0  
  }  
}
```

1.2.6 Vis stemmer (valg 6. i client.py)

Klient oppgir kommando "6" og ID, og får tilbake antall stemmer på følgende problem.

- Klient → Server:

```
{  
    "kommando": 6,  
    "problemID": 100  
}
```

- Server → Klient:

```
{  
    "tittel": "Hva skal jeg ha til middag?",  
    "alternativ": {  
        "Pølse": 5, "Hamburger": 2, "Pizza": 3  
    }  
}
```

b) Tilstandsløs vs tilstandsfull og TCP vs UDP

2.1 Tilstandsløs vs tilstandsfull

Applikasjonsprotokollen er tilstandsløs, siden hver melding inneholder all nødvendig informasjon.

2.2 TCP vs UDP

TCP er best egnet da vi trenger pålitelig overføring av data for å sikre at stemmer registreres riktig.

c) Implementering av klient-server-applikasjonen i Python

3.1 Se server.py og client.py.