Stegvis fremgangsmåte for å lage chatteprogrammet ved bruk av `action="post"` i form:

Siden du ønsker at elevene skal bruke `action="post"` i et `<form>`-element slik de har lært, har vi justert fremgangsmåten for å reflektere dette.

---

1. Opprett et nytt Node.js-prosjekt

- Lag en ny mappe for prosjektet ditt.
- Åpne en terminal i denne mappen og kjør `npm init -y` for å initialisere et nytt Node.js-prosjekt.

2. Installer nødvendige avhengigheter

- Installer Express ved å kjøre:

```
npm install express
```

- Valgfritt: Installer `nodemon` for enklere utvikling:

```
npm install --save-dev nodemon
```

3. Sett opp grunnleggende server med Express

- Lag en fil kalt `app.js`.
- Importer Express og opprett en app-instans:

```
const express = require('express');
const app = express();

app.listen(3000, () => {
  console.log('Server kjører på port 3000');
});
```

4. Legg til middleware for å håndtere POST-data

- Bruk `express.urlencoded()` middleware for å kunne lese URL-enkodet data fra forms:

```
app.use(express.urlencoded({ extended: true }));
```

5. Server statiske filer

- Legg til følgende i `app.js` for å servere statiske filer fra en `public`-mappe:

```
app.use(express.static('public'));
```

- Opprett en mappe kalt `public` i prosjektet ditt.

6. Lag HTML-filen for chat-grensesnittet

- I `public`-mappen, lag en fil kalt `index.html`.
- Legg til et `<form>`-element med `method="post"` og `action="/send"`:

```
<!DOCTYPE html>
<html lang="no">
<head>
  <meta charset="UTF-8">
  <title>Enkel Chat</title>
</head>
<body>
  <form method="post" action="/send">
    <input type="text" name="name" placeholder="Ditt navn" required>
    <textarea name="message" placeholder="Skriv meldingen din her..."
required></textarea>
    <input type="submit" value="Send">
  </form>

  <div id="messages"></div>

  <script src="script.js"></script>
</body>
</html>
```

7. Legg til klient-side JavaScript for å håndtere ENTER-tasten

- Lag en fil `script.js` i `public`-mappen.
- I `script.js`, legg til en hendelseslytter på `<textarea>` som lytter etter `keydown`-hendelser:

```
const textarea = document.querySelector('textarea');
textarea.addEventListener('keydown', function(event) {
  if (event.key === 'Enter' && !event.shiftKey) {
    event.preventDefault();
    document.querySelector('form').submit();
  }
});
```

- Dette vil sende formet når brukeren trykker ENTER i tekstområdet uten å holde SHIFT (for linjeskift).

8. Sett opp en datastruktur for å lagre meldinger på serveren

- I `app.js`, lag en array for å lagre meldingene:

```
const messages = [];
```

9. Håndter POST-forespørsler fra formet

- Legg til en POST-rute på `'/send'` i `app.js`:

```
app.post('/send', (req, res) => {
  const name = req.body.name;
  const message = req.body.message;
  const timestamp = new Date().toLocaleTimeString('no-NO', { hour: '2-
digit', minute: '2-digit' });
  messages.unshift({ timestamp, name, message });
  res.redirect('/');
});
```

- Meldingen lagres med tidsstempel, navn og meldingstekst.
- `messages.unshift()` legger meldingen foran i arrayen slik at de nyeste meldingene vises øverst.

10. Lag en GET-rute for å hente meldinger

- I `app.js`, lag en rute på `'/messages'` som sender meldingene som JSON:

```
app.get('/messages', (req, res) => {
  res.json(messages);
});
```

11. Oppdater klienten til å hente meldinger regelmessig

- I `script.js`, lag en funksjon som henter meldingene fra serveren hvert 3. sekund:

```
function fetchMessages() {
  fetch('/messages')
    .then(response => response.json())
    .then(data => {
      const messagesDiv = document.getElementById('messages');
      messagesDiv.innerHTML = '';
      data.forEach(msg => {
        const messageElement = document.createElement('p');
        messageElement.textContent = `${msg.timestamp} - ${msg.name}:
${msg.message}`;
        messagesDiv.appendChild(messageElement);
      });
    });
```

```
}

setInterval(fetchMessages, 3000);
fetchMessages();
```

- Denne funksjonen oppdaterer meldingsvisningen i HTML.

12. Oppdater meldinger umiddelbart etter sending

- Etter at formet er sendt og siden lastes på nytt, vil `fetchMessages()` bli kalt igjen og vise den nye meldingen.
- For å sikre at meldinger oppdateres umiddelbart for avsenderen, kan du kalle `fetchMessages()` etter at siden er lastet:

```
window.onload = fetchMessages;
```

13. Formater meldinger med tidsstempel på serveren

- Tidsstempelet er allerede formatert i steg 9 ved bruk av `toLocaleTimeString()`.

14. Test applikasjonen lokalt

- Start serveren ved å kjøre:

```
node app.js
```

eller med nodemon:

```
npx nodemon app.js
```

- Åpne nettleseren og naviger til `http://localhost:3000`.
- Test funksjonaliteten ved å sende meldinger fra ulike nettleservinduer eller enheter.

15. Gjør serveren tilgjengelig over internett med ngrok

- Installer ngrok fra ngrok.com.
- Start ngrok med kommandoen:

```
ngrok http 3000
```

- Del den genererte ngrok-URL-en med klassen slik at alle kan teste applikasjonen sammen.

Ekstra tips:

- **Sideoppdatering ved sending av melding:** Ved bruk av `action="post"` og `res.redirect('/')`, vil siden oppdatere seg når meldingen sendes. Dette er akseptabelt siden det reflekterer hvordan forms tradisjonelt fungerer.
- **Forbedring uten sideoppdatering:** Hvis du ønsker å unngå sideoppdatering, kan du senere introdusere AJAX for å sende data asynkront, men dette kan være utenfor dagens læringsmål.
- **Feilhåndtering:** Legg til enkle feilmeldinger hvis navn eller melding mangler, selv om `required`-attributtet i HTML-formen skal håndtere dette.
- **Stil og design:** Oppmuntre elevene til å legge til CSS for å gjøre chatten mer attraktiv og brukervennlig.

Med denne fremgangsmåten får elevene praktisk erfaring med å bruke forms og `action="post"` for å sende data til serveren, samtidig som de ser hvordan serveren håndterer data og sender oppdateringer tilbake til klienten. Dette gir en god forståelse av grunnleggende webutvikling med Node.js og Express.