Nedenfor finner du et steg-for-steg opplegg for hvordan elevene kan lage et enkelt 4x4 memory-spill (med 8 par) der bildene hentes fra MySQL-databasen. Opplegget tar utgangspunkt i at de allerede kan grunnleggende Node.js, Express, EJS og MySQL (som i ansattprosjektet), og vil nå bygge videre for å lage et lite spill.

1. Planlegging av prosjektet

Hovedkomponenter:

1. **Database** (MySQL) med minst to tabeller:
   - Én tabell for bilder (minst 20 rader).
   - Én tabell for highscores.
2. **EJS-filer** i `views/`-mappen:
   - `index.ejs` (startside)
   - `game.ejs` (spillside)
   - `gameover.ejs` (game over-side)
   - `highscores.ejs` (visning av highscore-liste)
   - (samt `partials/header.ejs` og `partials/footer.ejs` for gjenbruk)
3. `style.css` i `public/`-mappen for et brukervennlig design.
4. **Node.js / Express**-app (`index.js`) som kobler seg til MySQL, henter bilder, genererer 8 par tilfeldig, håndterer post-/get-forespørsler, lagrer highscore, etc.
5. **Front-end JavaScript** (kan ligge i en egen `.js`-fil i `public/`, eller inline i EJS) for selve memory-logikken (snurring, animasjon).

---

2. Oppsett av databasen

2.1. Opprett en database

Eksempel: `CREATE DATABASE memory_db;`

2.2. Tabell for bilder

Opprett en tabell som for eksempel heter `bilder`. Den kan ha følgende felter:

- `id` (PRIMARY KEY, AUTO_INCREMENT)
- `bildenavn` (f.eks. "hund", "bil", "fjell" osv. – bare en beskrivende tekst)
- `bildedata` (BLOB eller MEDIUMBLOB – selve binærdata for bildet)

SQL-eksempel:

```sql
CREATE TABLE bilder (
  id INT AUTO_INCREMENT PRIMARY KEY,
  bildenavn VARCHAR(100) NOT NULL,
  bildedata LONGBLOB NOT NULL
);
```

> *Merk:* Du kan bruke `LONGBLOB` eller `MEDIUMBLOB` avhengig av forventet filstørrelse.

Last deretter inn minst 20 bilder. Du må lagre selve binærdataen. Dette kan gjøres enten med manuelt SQL-INSERT, eller ved et skript i Node.js som leser filer og lagrer dem i databasen.

2.3. Tabell for highscores

Denne skal lagre informasjon om spillerens resultat:

- `id` (PRIMARY KEY, AUTO_INCREMENT)
- `spiller_navn` (VARCHAR) – bare tall og bokstaver
- `tid` (INT) – antall sekunder brukt (eller lignende)
- `forsok` (INT) – antall ganger spilleren har snudd to kort
- `score` (INT) – poengsum basert på tid og forsøk

SQL-eksempel:

```
CREATE TABLE highscores (
  id INT AUTO_INCREMENT PRIMARY KEY,
  spiller_navn VARCHAR(50) NOT NULL,
  tid INT NOT NULL,
  forsok INT NOT NULL,
  score INT NOT NULL
);
```

---

3. Mappestruktur

En typisk mappestruktur kan se slik ut:

```
memory-spill/
├─ public/
│  ├─ style.css
│  └─ main.js        // Kan ha front-end JS (f.eks. memory-funksjonalitet)
├─ views/
│  ├─ partials/
│  │  ├─ header.ejs
│  │  └─ footer.ejs
│  ├─ index.ejs
│  ├─ game.ejs
│  ├─ gameover.ejs
│  └─ highscores.ejs
└─ index.js
```

- **public/**: inneholder statiske filer (CSS, front-end JS, evt. bilder om man skulle trenge – men her har vi bilder i DB).
- **views/**: inneholder EJS-malene.
- **partials/**: gjenbrukbare deler som `header.ejs` og `footer.ejs`.
- **index.js**: selve Node/Express-serveren.

---

4. Bygg EJS-filene steg for steg

4.1. `partials/header.ejs`

- Inneholder HTML-header, `<link>` til CSS, en `<nav>` med lenker til «Startside», «Highscores», etc.
- Eksempel:

```
<!DOCTYPE html>
<html lang="no">
<head>
  <meta charset="UTF-8">
  <title>Memory-spill</title>
  <link rel="stylesheet" href="/style.css">
</head>
<body>
<header>
  <h1>Memory-spill</h1>
  <nav>
    <a href="/">Startside</a>
    <a href="/highscores">Highscores</a>
  </nav>
</header>
```

4.2. `partials/footer.ejs`

- Inneholder footer-elementet og avsluttende `</body>` og `</html>`.

```
<footer>
  <p>Memory-spill © 2025</p>
</footer>
</body>
</html>
```

4.3. `index.ejs` (Startside)

- Skal vise en enkel «Velkommen til memory-spill».
- Knapp «Start spill», som fører spilleren til en side hvor de kan skrive inn spillernavnet.
- For enkelhets skyld kan man enten:
    1. Ha et `<form>` som tar inn spillernavn direkte her, og ved POST går videre til `/game`-ruten.
    2. Eller lenke til en egen side – men ofte er det ok å gjøre alt fra startsiden.

```
<%- include('partials/header') %>

<h2>Velkommen til memory-spill!</h2>
<p>Trykk på "Start spill" for å begynne.</p>

<form action="/game" method="POST">
```

```
  <label for="spillerNavn">Spillernavn:</label><br>
  <input type="text" id="spillerNavn" name="spillerNavn" required pattern="
[A-Za-z0-9]+">
  <!-- pattern="[A-Za-z0-9]+" gjør at vi kun tillater bokstaver og tall,
ingen mellomrom -->

  <button type="submit">Start spill</button>
</form>

<%- include('partials/footer') %>
```

4.4. `game.ejs` (Spillsiden)

- Viser selve 4x4 brettet.
- Her må vi ha en eller annen måte å vise 16 «kort» på, og bildene hentes dynamisk.
- I praksis vil du servere 16 «kort» (8 par) tilfeldig trukket fra de 20 bildene i databasen.
- Under hver kort-lenke/knapp kan du vise en *thumbnail* av bildet (eller bare en bakside før klikk).

```
<%- include('partials/header') %>

<h2>Memory-spill</h2>

<div id="game-board">
  <!-- Her genererer du 16 "kort" -->
  <!-- Eksempel på struktur: -->
  <% for(let i = 0; i < cards.length; i++) { %>
    <div class="card" data-id="<%= cards[i].id %>">
      <div class="card-back"></div>
      <div class="card-front">
        <!-- For å vise bildet, kan du generere en "data-url" fra BLOB i
Node,
             eller servere det via en egen /image/:id-løsning.
             Eksempel: <img src="/image/<%= cards[i].id %>" alt="Memory-
bilde"> -->
        <img src="/image/<%= cards[i].id %>" alt="Memory-bilde">
      </div>
    </div>
  <% } %>
</div>

<!-- En knapp for å gi opp, bare som eksempel -->
<form action="/gameover" method="POST">
  <button type="submit">Avslutt spill</button>
</form>

<script src="/main.js"></script>

<%- include('partials/footer') %>
```

> *Merk:* Her er det mye rom for variasjon, men hovedideen er at hver «card» har et front-bilde og en backside. Man kan styre logikken og snuing med CSS og JavaScript (`main.js`).

4.5. `gameover.ejs`

- Viser en oppsummering av spillets resultater (tid, antall forsøk, score).
- Mulighet for å gå til Highscore-liste eller spille på nytt.

```
<%- include('partials/header') %>

<h2>Game Over, <%= spillerNavn %>!</h2>
<p>Du brukte <%= tid %> sekunder og <%= forsok %> forsøk.</p>
<p>Din score: <%= score %></p>

<a href="/highscores">Se Highscores</a> | <a href="/">Spill på nytt</a>

<%- include('partials/footer') %>
```

4.6. `highscores.ejs`

- Viser en tabell med toppresultater (navn, tid, forsøk, score) sortert på best score (eller hva man vil).
- Eksempel:

```
<%- include('partials/header') %>

<h2>Highscores</h2>

<table>
  <tr>
    <th>Plass</th>
    <th>Navn</th>
    <th>Tid (sek)</th>
    <th>Forsøk</th>
    <th>Score</th>
  </tr>
  <% for(let i = 0; i < scores.length; i++) { %>
    <tr>
      <td><%= i+1 %></td>
      <td><%= scores[i].spiller_navn %></td>
      <td><%= scores[i].tid %></td>
      <td><%= scores[i].forsok %></td>
      <td><%= scores[i].score %></td>
    </tr>
  <% } %>
</table>

<%- include('partials/footer') %>
```

5. CSS (`public/style.css`)

I tillegg til den generelle CSS-en du allerede har (typ. body, header, footer), bør du inkludere styling for selve memory-spillet:

```css
/* Generell styling */
body {
  font-family: Arial, sans-serif;
  margin: 20px;
  background-color: #f9f9f9;
}

header {
  background-color: #333;
  color: #fff;
  padding: 10px;
}

header h1 {
  margin: 0;
  font-size: 1.5rem;
}

nav a {
  color: #fff;
  margin-right: 10px;
  text-decoration: none;
}

nav a:hover {
  text-decoration: underline;
}

footer {
  background-color: #eee;
  padding: 10px;
  margin-top: 20px;
  text-align: center;
}

h2 {
  margin-top: 0;
}

/* Memory-spill board */
#game-board {
  display: grid;
  grid-template-columns: repeat(4, 120px);
  grid-gap: 10px;
  margin: 20px 0;
}
```

```css
.card {
  width: 120px;
  height: 120px;
  position: relative;
  cursor: pointer;
  perspective: 1000px; /* For 3D flip */
}

.card-back,
.card-front {
  width: 100%;
  height: 100%;
  position: absolute;
  backface-visibility: hidden;
  border: 1px solid #ccc;
  display: flex;
  align-items: center;
  justify-content: center;
}

.card-back {
  background-color: #007BFF;
  color: #fff;
  font-size: 1.2rem;
}

.card-front {
  background-color: #fff;
  transform: rotateY(180deg);
}

.card.flipped .card-back {
  transform: rotateY(180deg);
}

.card.flipped .card-front {
  transform: rotateY(0deg);
}

/* Knapper */
button {
  padding: 5px 10px;
  cursor: pointer;
  background-color: #007BFF;
  border: none;
  color: #fff;
}

button:hover {
  background-color: #0056b3;
}

table {
  width: 100%;
```

```
  border-collapse: collapse;
  margin-top: 10px;
  background-color: #fff;
}

table th,
table td {
  border: 1px solid #ccc;
  padding: 8px;
}

table th {
  background-color: #f1f1f1;
}
```

> *Merk:* Her brukes `transform: rotateY()`-teknikk for å flippe kortene. Selve «flippe-late- som-animasjonen» styres med `card.flipped` klassen i JavaScript.

---

6. Node.js-backend (`index.js`)

Nedenfor et omriss av hvordan serveren kan settes opp:

```
const express = require('express');
const mysql = require('mysql2');
const path = require('path');
const bodyParser = require('body-parser');

const app = express();
app.set('view engine', 'ejs');
app.set('views', path.join(__dirname, 'views'));

// Serve statiske filer
app.use(express.static(path.join(__dirname, 'public')));

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

// Opprett database-tilkobling
const db = mysql.createConnection({
  host: 'localhost',
  user: 'root',
  password: '',    // endre om nødvendig
  database: 'memory_db'
});

db.connect((err) => {
  if (err) throw err;
  console.log('MySQL connected...');
});

// Hjemside (index)
```

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

// Rute for å starte spill (POST med spillerNavn)
app.post('/game', (req, res) => {
  const spillerNavn = req.body.spillerNavn;

  // 1) Validér spillerNavn (har vi pattern i HTML? Ok!)
  // 2) Hent 20 bilder fra DB, trekk ut 8 tilfeldig
  const sql = 'SELECT id, bildenavn FROM bilder ORDER BY RAND() LIMIT 8';
  db.query(sql, (err, results) => {
    if (err) throw err;

    // 3) Vi må duplisere disse 8 for å få 8 par (totalt 16).
    //    Deretter stokke dem om (shuffle).
    let cards = [];
    results.forEach((img) => {
      cards.push(img); // par 1
      cards.push(img); // par 2
    });

    // Shuffle function
    function shuffle(array) {
      for (let i = array.length - 1; i > 0; i--) {
        const j = Math.floor(Math.random() * (i + 1));
        [array[i], array[j]] = [array[j], array[i]];
      }
      return array;
    }

    cards = shuffle(cards);

    // 4) Render game.ejs og send med cards og spillerNavn
    res.render('game', {
      spillerNavn: spillerNavn,
      cards: cards
    });
  });
});

// Rute for å servere bildedata via /image/:id
app.get('/image/:id', (req, res) => {
  const id = req.params.id;
  db.query('SELECT bildedata FROM bilder WHERE id=?', [id], (err, results)
=> {
    if (err) throw err;
    if (!results[0]) {
      return res.status(404).send('Bilde ikke funnet');
    }
    const bildedata = results[0].bildedata;
    // Sett korrekt content type (for eksempel "image/jpeg" hvis det er
jpg)
    // eller lagre mime-typen i DB også hvis du vil ha mer presis info.
```

```
    res.setHeader('Content-Type', 'image/jpeg');
    res.send(bildedata);
  });
});

// Rute for "Avslutt spill" (gameover) – her vil du få tid, forsok fra
front-end
app.post('/gameover', (req, res) => {
  const { spillerNavn, tid, forsok } = req.body;

  // Beregn score basert på tid og forsøk:
  // For eksempel:
  //    score = 10000 / (tid * forsok)
  // eller en litt enklere variant:
  //    score = Math.max(0, 1000 - (tid * 2 + forsok * 10));
  // Her kan man eksperimentere.
  const score = Math.max(0, 1000 - (tid * 2 + forsok * 10));

  // Sett inn i DB
  const sql = 'INSERT INTO highscores (spiller_navn, tid, forsok, score)
VALUES (?, ?, ?, ?)';
  db.query(sql, [spillerNavn, tid, forsok, score], (err, result) => {
    if (err) throw err;
    // Send spiller videre til gameover-siden
    res.render('gameover', {
      spillerNavn: spillerNavn,
      tid: tid,
      forsok: forsok,
      score: score
    });
  });
});

// Rute for å vise highscores
app.get('/highscores', (req, res) => {
  const sql = 'SELECT * FROM highscores ORDER BY score DESC LIMIT 10';
  db.query(sql, (err, results) => {
    if (err) throw err;
    res.render('highscores', { scores: results });
  });
});

app.listen(3000, () => {
  console.log('Server kjører på http://localhost:3000');
});
```

Kommentarer

1. **Hente bilder**: Ved `/game` henter vi 8 bilder tilfeldig fra `bilder`-tabellen, dupliserer dem for å få par, og stokker om. Dette sendes videre til `game.ejs` for visning.

2. **Visning av hvert bilde**: Vi bruker en egen GET-rute `/image/:id` for å servere selve binærdataen fra DB. I `<img src="/image/ID">` i EJS hentes bildet.

3. **Lagre resultater**: Når spillet er slutt, send tid og forsøk (og spillerNavn) i en POST til `/gameover`. Her lagres alt i `highscores`-tabellen.

4. **Score-formel**: Her kan dere leke dere litt. F.eks:

   [\text{score} = 1000 - (tid \times 2 + forsok \times 10)]

   Da betyr mindre tid og færre forsøk at man får høyere score. Dere kan også bruke mer fancy formler.

5. **Validering av spillerNavn** gjøres med `pattern="[A-Za-z0-9]+"` i HTML, men man bør også validere på server for å være sikker.

---

7. JavaScript-frontend (f.eks. `public/main.js`)

For å håndtere flipping av kort kan man bruke en enkel eventListener:

```
document.addEventListener('DOMContentLoaded', () => {
  const cards = document.querySelectorAll('.card');

  let flippedCards = [];
  let lockBoard = false;
  let matchedPairs = 0;
  let attempts = 0;
  let startTime = Date.now();

  cards.forEach(card => {
    card.addEventListener('click', () => {
      if (lockBoard) return;            // Hindre klikk under "venting"
      if (flippedCards.length === 2) return; // Har allerede to klikk

      card.classList.add('flipped');
      flippedCards.push(card);

      if (flippedCards.length === 2) {
        attempts++;
        // Sjekk match
        const id1 = flippedCards[0].dataset.id;
        const id2 = flippedCards[1].dataset.id;

        if (id1 === id2) {
          // De forblir åpne
          matchedPairs++;
          flippedCards = [];
          // Sjekk om spillet er ferdig
          if (matchedPairs === 8) {
            // Spill slutt
            const endTime = Date.now();
            const tidISekunder = Math.floor((endTime - startTime) / 1000);
```

```
            // Send data til server for å lagre gameover
            const spillerNavn = getSpillerNavnFromSomewhere();
            // Alternativt kan du legge spillerNavn i en hidden input
            sendGameover(spillerNavn, tidISekunder, attempts);
          }
        } else {
          // Ikke match, vent litt og snu tilbake
          lockBoard = true;
          setTimeout(() => {
            flippedCards[0].classList.remove('flipped');
            flippedCards[1].classList.remove('flipped');
            flippedCards = [];
            lockBoard = false;
          }, 1000);
        }
      }
    });
  });
});

function getSpillerNavnFromSomewhere() {
  // Du kan legge spillerNavn i et hidden element i game.ejs
  // og bare hente .value her, for eksempel
  const hiddenInput = document.getElementById('spillerNavnHidden');
  return hiddenInput ? hiddenInput.value : 'Ukjent';
}

function sendGameover(spillerNavn, tid, forsok) {
  // Lag en form-data og POST til /gameover
  // Du kan bruke fetch:
  fetch('/gameover', {
    method: 'POST',
    headers: { 'Content-Type': 'application/x-www-form-urlencoded' },
    body: `spillerNavn=${spillerNavn}&tid=${tid}&forsok=${forsok}`
  })
  .then(response => response.text())
  .then(html => {
    // Erstatt hele body med gameover-svar:
    document.documentElement.innerHTML = html;
  });
}
```

> *Merk:* Det finnes mange måter å gjøre dette på. Du kan også bruke websideredirekt med `window.location.href = ...`. Over er bare et eksempel på hvordan man kan POSTe data.

---

8. Forslag til score-formel

En enkel formel kan være: [\text{score} = \max\Bigl(0,; 1000 - (\text{tid i sek} \times 2 + \text{forsøk} \times 10)\Bigr)]

- Hvis du bruker **mindre tid** og **færre forsøk**, får du **høyere** poeng.

- Etter et visst punkt kan scoren bli 0 (da du har brukt mange forsøk / lang tid).

Andre alternativer:

- (\text{score} = \lfloor \frac{10000}{(\text{tid i sek} + \text{forsøk})} \rfloor)
- (\text{score} = \text{noe kreativt})

Poenget er at elevene gjerne eksperimenterer med formlene.

---

9. Videre tips og utvidelser

1. **Bruk Sessions**: Du kan lagre `spillerNavn`, start-tid og antall forsøk i en session for å unngå å sende alt i POST hver gang.
2. **Mer animasjon**: Tilleggseffekter med CSS-animasjoner, lyd ved match, etc.
3. **Flere vanskelighetsgrader**: 4x4, 6x6, 8x8.
4. **Flere spilltyper**: Ha en dropdown for valg av 4x4, 6x6. Tilpass antall bilder.
5. **Autentisering**: La brukere opprette konto, logge inn, og ha personlige highscores.

Oppsummering

Stegene for å lage prosjektet:

1. **Database**
   - Opprett `bilder`-tabell (lagre bilder i BLOB-felt).
   - Opprett `highscores`-tabell (lagre navn, tid, forsøk, score).
2. **Mappestruktur og tilhørende filer**
   - `public/` for CSS og ev. front-end JS (`main.js`).
   - `views/` med EJS-filer for startside (`index.ejs`), spillside (`game.ejs`), game over (`gameover.ejs`), highscore (`highscores.ejs`), og partials for header/footer.
3. **Node.js/Express**-app
   - Koble til MySQL.
   - Ruter:
     - `GET /` -> `index.ejs` (startside).
     - `POST /game` -> velger 8 av 20 bilder (tilfeldig), lager par, shuffle, rendrer `game.ejs`.
     - `GET /image/:id` -> serverer bilde-BLOB fra databasen.
     - `POST /gameover` -> mottar tid, forsøk, kalkulerer score, lagrer i DB, rendrer `gameover.ejs`.
     - `GET /highscores` -> henter toppscore fra DB, rendrer `highscores.ejs`.
4. **Front-end logikk**
   - JavaScript for memory-effekten (flippe kort).
   - Hold styr på forsøk, tid, matched par.
   - Når spillet er ferdig, POST data til `/gameover`.

5. **Score**
    - Finn en formel for å beregne en rettferdig poengsum.
    - Vis scoren og la brukerne konkurrere via `highscores.ejs`.
6. **Design**
    - Bruk/tilpass en `style.css`.
    - Vektlegg brukervennlighet og tydelig tilbakemelding når to kort matcher eller ikke.