

1. Introduction

This project implements a web-based “Memory Game” (also known as Concentration or Matching Pairs) using HTML5, CSS3, and JavaScript (with jQuery). The goal is to flip over pairs of matching cards until all matches are found. Key enhancements include:

- **SVG Icon Cards** instead of simple numbers
- **Difficulty Levels** (grid sizes: 4×4, 4×6, 6×6)
- **Flip Animation** via CSS 3D transforms
- **Start Game Flow**: A preview period where all cards are shown face-up for 5 seconds before play begins
- **Background Music** with a toggle button (mute/unmute)
- **Push Notifications** (Notification API) on game completion
- **Offline Support** through a Service Worker (caching assets)
- **LocalStorage Persistence** for high scores/results
- **Responsive Design** for mobile devices and desktops

All comments in the JavaScript code are written in English, and UI labels are in English to match the documentation requirements.

2. Project Features

1. **User Authentication**
 - Simple “Login/Register” form stored in LocalStorage.
 - Upon successful login (or registration), the user is redirected to the game page.
2. **Difficulty Selection**
 - Options:
 - 4×4 (8 pairs)
 - 4×6 (12 pairs)
 - 6×6 (18 pairs)
 - The dropdown is enabled only before the game starts. Once the player clicks “Start Game,” the dropdown becomes disabled until the current game ends.
3. **Start Game Sequence**
 - A “Start Game” button appears alongside the grid.
 - Upon click:
 1. Difficulty dropdown is locked.
 2. Background music begins (if not already playing).
 3. All cards flip face-up for 5 seconds so the player can memorize positions.
 4. After 5 seconds, cards flip back face-down and the timer starts.
 5. The player can then click cards to flip and attempt matches.
4. **Card Flip Animation**
 - Each card is an HTML `<div class="card">` with an inner wrapper `<div class="card__inner">`.
 - CSS 3D transforms (`transform: rotateY(180deg)`) create a smooth flip effect when `.flipped` is toggled.
5. **SVG Icon Usage**
 - Cards use local SVG files named `icon1.svg` through `icon18.svg` in `assets/svg/`.

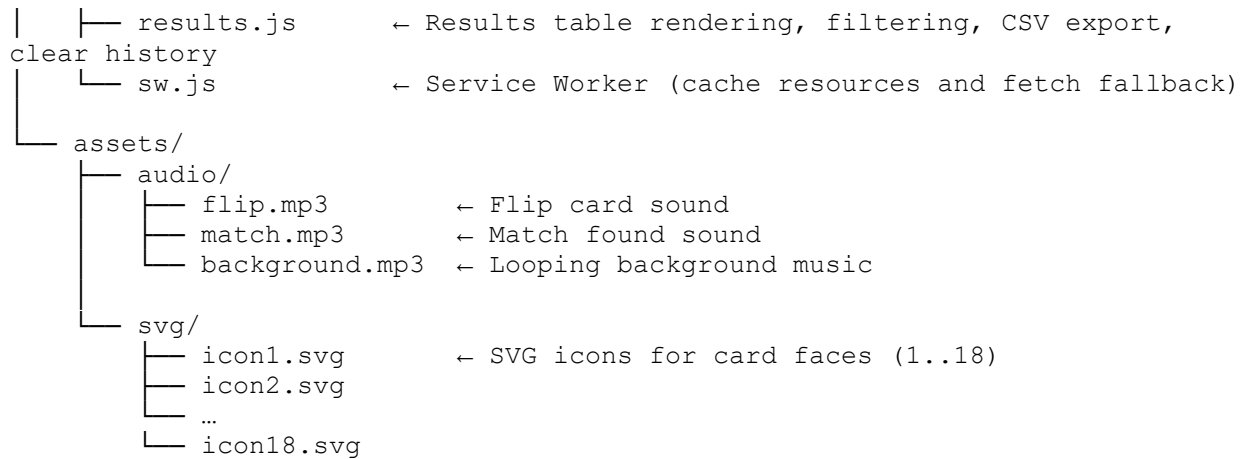
- The face value (numeric) corresponds to an SVG file (e.g., `faceValue = 3` → `icon3.svg`).
 - 6. **Background Music & Audio Effects**
 - **Flip Sound** (`flip.mp3`), **Match Sound** (`match.mp3`), and **Background Music** (`background.mp3`) are preloaded via HTML `<audio>` elements.
 - A “**Music On/Off**” toggle button mutes or unmutes background music at any time.
 - 7. **Timer & Move Counter**
 - Timer starts only after the 5-second preview phase ends.
 - Move counter increments each time the second card in a pair is flipped (regardless of match or not).
 - 8. **Game Completion Modal & Notifications**
 - Once all pairs are matched:
 - The timer stops.
 - A modal appears showing final time and moves.
 - A push notification is sent (if permission granted) with the summary.
 - The user can save the result (persists to `LocalStorage`) or choose “Play Again.”
 - 9. **Results Page**
 - Shows a table of all saved game results (date, username, moves, time).
 - Live filtering by username.
 - Export results to CSV.
 - Clear entire history.
 - 10. **Offline Support (PWA)**
 - A Service Worker (`js/sw.js`) caches HTML, CSS, JS, audio, and SVG files.
 - When offline, the game still loads and functions (except saving results to `LocalStorage`, which is blocked and alerts the user).
 - 11. **Responsive Design**
 - CSS media queries ensure the grid scales to fit mobile widths.
 - Cards remain square, and the grid’s `max-width` adapts to viewport size.
-

3. Project Structure

```

memory-game/
├── index.html           ← Login/Register page (HTML5 form, geolocation
display)
├── game.html           ← Main game page (grid, controls, modal)
├── results.html        ← Results page (history table, filters, export)
├── manifest.json       ← PWA manifest
├──
├── css/
│   └── styles.css      ← All styling: layout, grid, animations, responsive
├── js/
│   ├── app.js          ← Initialization, offline/online handling, SW
registration, auth check, geolocation
│   ├── auth.js         ← Login/Register logic (LocalStorage users)
│   └── game.js         ← MemoryGame class, “Start Game” logic, flip
matching, notifications

```



4. Detailed Functionality

5.1. index.html (Login / Registration)

- **HTML5 Form** (`<form id="auth-form">`):
 - Fields: username (text, required), password (password, required).
 - Validation: Built-in HTML5 “required” attributes.
 - On submit:
 1. Retrieves existing users from `localStorage.getItem('memoryGameUsers')` (array of { username, password }).
 2. If username exists, checks password; if correct, stores `memoryGameUser = username` in `LocalStorage` and redirects to `game.html`. Otherwise, alerts “Incorrect password.”
 3. If username does not exist, registers a new user (push to array), saves in `LocalStorage`, sets `memoryGameUser`, and redirects.
 - **Offline Notice** (`<p id="offline-status">...</p>`):
 - Initially hidden by CSS class `.hidden`.
 - `app.js` checks `navigator.onLine` and toggles `.hidden` accordingly.
 - **Geolocation Display**:
 - If user’s permission is granted, `navigator.geolocation.getCurrentPosition()` retrieves coordinates and displays them under the header (`<p id="user-location">`).
-

4.2. game.html (Memory Game)

4.2.1. Layout Overview

```
<header class="site-header">
  <h1>Memory Game</h1>
  <button id="logout-btn">Logout</button>
</header>

<main>
  <section class="game-info">
    <!-- Difficulty dropdown -->
```

```

    <!-- Player name, moves, time -->
    <!-- Offline notice -->
    <!-- Start Game button -->
    <!-- Music On/Off button -->
</section>

<section class="game-board">
  <div id="game-grid" class="game-grid"></div>
</section>

<!-- Modal for "Game Over!" -->
<div id="game-over-modal" class="modal hidden"> ... </div>

<!-- Audio elements (flip, match, background) -->
</main>

<footer> ... </footer>

```

- **Controls:**
 - **Difficulty:** <select id="difficulty-select">
 - **Player:**
 - **Moves:**
 - **Time:**
 - **Offline Notice:** <p id="offline-status-game">
 - **Start Game:** <button id="start-game-btn">Start Game</button>
 - **Music Toggle:** <button id="toggle-music-btn">🔊 Music On</button>
- **Game Grid:**
 - <div id="game-grid" class="game-grid"></div>
 - Initially rendered with face-down cards.
- **Modal:**
 - <div id="game-over-modal" class="modal hidden"> ... </div>
 - Shown only when all pairs are matched.

4.2.2. User Flow

1. **Page Load**
 - JavaScript checks LocalStorage for `memoryGameUser`. If absent, redirects to `index.html`.
 - Difficulty dropdown is enabled, “Start Game” is enabled, music toggle is enabled (though music is initially paused).
 - Card grid is rendered but all cards remain face-down (no timer or flip logic active).
2. **User clicks “Start Game”**
 - `game.performStartSequence()` is called:
 - Sets `gameStarted = true`.
 - Disables difficulty dropdown.
 - Starts background music (if not playing).
 - Immediately flips all cards face-up (adding `.flipped` CSS class).
 - After 5 seconds, flips back any unmatched cards (removes `.flipped`) and calls `startTimer()` to begin counting.
 - “Start Game” button becomes disabled.
 - Moves and timer remain at 0 until the 5-second preview ends.
3. **During Play**
 - The user may click any face-down card (since `gameStarted === true`).
 - On the first card of a pair, it flips face-up and plays flip sound.

- On the second card of a pair:
 - Flips face-up and plays flip sound.
 - `moves` increments by 1, displayed in ``.
 - If the two `faceValues` match:
 - Both cards remain face-up (`.matched` CSS class).
 - Match sound plays.
 - If all cards are matched: `endGame()` is triggered.
 - If they do not match: after 1 second, both flip back face-down.
- 4. **Toggling Music**
 - Clicking `<button id="toggle-music-btn">` toggles `bgMusic.pause()` vs. `.play()`.
 - The button text updates between “🔊 Music On” and “🔇 Music Off.”
- 5. **Game Over**
 - When every card’s `isMatched === true`:
 - `endGame()` stops the timer, sets `gameOver = true`, and displays the modal.
 - Modal shows final time and final moves.
 - If Notification permission is granted, a browser notification pops up summarizing the result.
 - The user can click “**Save Result**” to write the result into `LocalStorage` (array of objects), then redirect to `results.html`.
 - Or click “**Play Again**” (reload the page).
- 6. **Post-Game**
 - Difficulty dropdown remains disabled until page reload.
 - Background music can still be toggled on/off.
 - The game grid remains visible in matched state until reload.

4.3. `results.html` (Game Results)

- **Controls:**
 - **Filter by Username:** `<input id="filter-username">`
 - **Export to CSV:** `<button id="export-csv-btn">`
 - **Clear History:** `<button id="clear-history-btn">`
 - **Play Again:** `<button id="back-to-game-btn">`
 - **Logout:** `<button id="logout-btn-results">`
- **Results Table:**
 - `<table><thead>...</thead><tbody id="results-body"></tbody></table>`
 - Renders rows: Date (localized string), Username, Moves, Time.
- **Functionality:**
 1. On page load, check `memoryGameUser`; if absent, redirect to `index.html`.
 2. Retrieve `memoryGameResults` from `LocalStorage` (array).
 3. Render all results sorted by date descending.
 4. Live filter on `keyup` or input in the username filter (`.includes()` case-insensitive).
 5. **Export to CSV:** Creates a CSV string with header row, then loops through all results, producing `<a download>` link and clicking it programmatically.
 6. **Clear History:** `localStorage.removeItem('memoryGameResults')`, re-render empty state (“No saved results.”) if none remain.
 7. **Play Again:** Redirect to `game.html`.

5. Technologies and APIs Used

1. HTML5

- o Semantic elements: `<main>`, `<header>`, `<footer>`, `<section>`
- o Form validation: `required`, `type="email"`, `autofocus`
- o `<audio>` elements for sound playback

2. CSS3

- o Grid Layout (`display: grid; grid-template-columns: repeat(n, 1fr)`)
- o 3D transforms & transitions (`transform: rotateY(180deg)`, `transition: transform 0.6s`)
- o Media queries for responsiveness (`@media (max-width: 600px) { ... }`)
- o Custom properties (if needed) and classes for blocking/changing visibility (`.hidden`)



3. JavaScript (ES6+)

- o **jQuery** for DOM ready (`$(function() {})`), event handling (`.on('click', ...)`), and simple selectors (`$('#id')`)
- o Prototypal “class” syntax (`class MemoryGame { ... }`)
- o Use of `const/let` block-scoped variables
- o Template literals (``...${var}...``) for string concatenation
- o Strong use of array functions (`.map()`, `.forEach()`, `.every()`)
- o `LocalStorage` for persisting user accounts (`memoryGameUsers`), current player (`memoryGameUser`), and game results (`memoryGameResults`)
- o `Service Worker API` (`navigator.serviceWorker.register('js/sw.js')`) for offline caching
- o `Notification API` (`new Notification(...)`) for push notifications
- o `Geolocation API` (`navigator.geolocation.getCurrentPosition()`)

4. Progressive Web App (PWA)

- o `manifest.json` includes icons, theme and background colors, and `start_url`
- o `sw.js` caches HTML, CSS, JS, audio, and SVG files so that the game still loads offline

6. How to Play

1. **Log In / Register** via `index.html`.
2. **Choose Difficulty** (4×4, 4×6, 6×6).
3. **Click “Start Game”:**
 - o All cards flip face-up for 5 seconds to help memorize.
 - o After 5 seconds, cards flip back face-down, timer starts, and you can click cards.
4. **Flip Cards:**
 - o Click any two cards. If they match, they stay face-up; if not, they flip back after 1 second.
 - o Each pair attempt increments the move counter.
5. **Mute/Unmute Music** using the  /  button at any time.
6. **Game Over:**
 - o Once all pairs are matched, the timer stops.
 - o A modal shows your final time and moves.

- A push notification also appears (if allowed).
 - Click “**Save Result**” to store in high-score history, or “**Play Again**” to restart.
-

8. Offline Support

- **Service Worker (`js/sw.js`) caches:**
 - All HTML pages (`index.html`, `game.html`, `results.html`)
 - CSS (`css/styles.css`)
 - JS files (`js/app.js`, `js/auth.js`, `js/game.js`, `js/results.js`, `js/sw.js`)
 - Audio files (`assets/audio/flip.mp3`, `match.mp3`, `background.mp3`)
 - SVG icons (`assets/svg/icon1.svg ... icon18.svg`)
 - `manifest.json`
 - jQuery CDN fallback (cached once online)
 - **Behavior:**
 1. On first visit, Service Worker installs and caches all assets.
 2. Subsequent visits (online or offline) serve cached assets, ensuring the game loads even without network.
 3. If offline, saving results is disabled: an alert notifies the user that “You are offline. Cannot save the result.”
 4. All other game functions (flipping, matching, timer) work offline because they rely only on cached JS/CSS and LocalStorage.
-

9. Code Comments

All JavaScript files include inline comments (in English) explaining:

- Purpose of each method
- Parameter descriptions
- Event-handler logic
- Edge cases (e.g., ignoring clicks when `gameStarted === false`)
- Notification permission checks
- Service Worker cache steps

Refer to each file for detailed code-line comments:

- **`app.js`:**
 - Lines 1–40: Set current year in footer; handle online/offline messages; register Service Worker; geolocation; auth check and redirection.
 - Logout button logic.
- **`auth.js`:**
 - Lines 1–30: Intercept form submission; validate credentials; LocalStorage user store; redirect to game page.
- **`game.js`:**
 - Class `MemoryGame`—documented methods: `constructor()`, `setupNotifications()`, `startTimer()`, `loadGridSize()`, `resetGameData()`, `generateCards()`, `renderGrid()`, `handleClick()`, `flipCard()`, `endGame()`, `performStartSequence()`.

- **results.js:**
 - Lines 1–60: Load and render results table; live filter; clear history with confirmation; export to CSV; back to game logic.
 - **sw.js:**
 - Lines 1–50: Installation (`cache.addAll(urlsToCache)`), fetch interception (serve from cache first), activate event (remove old caches).
-