Memory Game Project

Project Overview

Memory Game is a single-page web application that allows users to register/log in, play a classic "concentration" memory-matching card game with customizable grid sizes, and view/save their game results. Core features include:

- **User Authentication** (client-side login/registration via `localStorage`)
- **Difficulty Levels** (4×4, 4×6, 6×6 grid sizes)
- **Flip/Match Card Animations** (using CSS 3D transforms)
- **Audio Feedback** (flip sound, match sound, background music)
- **Push Notifications** (via the Notification API on game completion)
- **Drag & Drop** for Matched Pairs (users drag matched cards into a "Matched Pairs" zone)
- **History API** integration (push/pop state after each match/mismatch)
- **WebSocket Chat** stub for potential real-time multiplayer or chat
- **Offline Support** (Service Worker for asset caching, PWA-compatible manifest)
- **Results Storage & Display** (saves games to `localStorage`, exports CSV, filters by username)

Features

1. **User Authentication**
    - Single-page form to register or log in.
    - Credentials stored in `localStorage` (no actual backend).
    - Auto-redirect to login if unauthenticated.
2. **Difficulty Levels**
    - Three grid sizes: 4×4 (8 pairs), 4×6 (12 pairs), 6×6 (18 pairs).
    - Difficulty must be selected *before* pressing "Start Game."
    - Once the game starts, the dropdown becomes disabled.
3. **Card Flip & Match Logic**
    - On "Start Game," all cards flip face-up for 5 seconds, then flip back → timer begins.
    - Players click/tap cards to reveal.
    - If two revealed cards match, they remain visible and become draggable.
    - If they do not match, they flip back after a one-second delay.
    - Background music (looping) starts on the first click.
4. **Audio Feedback**
    - Flip sound (short "click") each time a card flips.
    - Match sound ("ding") each time a pair is found.
    - Background music that can be toggled on/off.
5. **Push Notifications**
    - On game completion (all pairs found), the app triggers a Notification API alert (if permission granted).
    - Notification includes user's moves and time, and auto-closes after 5 seconds.

6. **Drag & Drop "Matched Pairs"**
   - Successfully matched cards become draggable.
   - A "Matched Pairs (Drag Here)" drop zone appears below the card grid.
   - When a user drags a matched card into this zone:
     - A cloned card appears inside the matched zone (static, non-draggable).
     - The original and its twin sibling are both visually hidden (but their grid slots remain occupied).
   - When multiple pairs are dropped, they wrap onto a new line once the row is full.
7. **History API & State Restoration**
   - After each match/mismatch, the current game state (including which cards are flipped/matched, current moves, time, difficulty, etc.) is pushed into `history.state` as a Base64-encoded JSON.
   - When the user navigates "Back"/"Forward" in the browser, the app intercepts the `popstate` event to restore the exact game state: card positions, flips, matches, moves, and elapsed time.
8. **WebSocket Chat Stub**
   - A brand-new "Game Chat" widget on the right side of the game page.
   - Connects to `wss://echo.websocket.org` (public echo server) as soon as the game page loads.
   - Input field + "Send" button allow sending a text message; the server echoes it back, and both system and peer messages appear in a scrollable "chat-log."
   - Intended as a foundation for future real-time multiplayer or chat integration.
9. **Offline Support & PWA**
   - A `sw.js` service worker caches all essential assets on first load (`html`, `css`, `js`, `audio`, `svg`, `manifest.json`).
   - Subsequent loads while offline will serve cached files.
   - A basic `manifest.json` (512×512 SVG icons) allows "Install to Home Screen" on mobile/desktop.
10. **Results Storage & Display**
    - After game completion, user clicks "Save Result" → an object is appended to `memoryGameResults` in `localStorage`:

      ```
      {
        date:       "2025-06-02T18:25:10.456Z",
        username:   "alice",
        difficulty: "4 × 6",
        moves:      15,
        time:       "00:42"
      }
      ```

    - **Results Page** shows a styled table with columns: Date | Username | Difficulty | Moves | Time.
    - Live filtering by username, Export to CSV, Clear History.
    - Empty-state message ("No saved results.") appears if there are no records.

Tech Stack

- **HTML5**
  - Semantic tags (`<header>`, `<main>`, `<section>`, `<footer>`)
  - Form elements with `required`, `placeholder`, `autofocus` attributes
  - Notifications API, History API, Service Worker API

- **CSS3**
    - Flexbox & CSS Grid for responsive layout
    - CSS 3D transforms for flip animations (`transform: rotateY(180deg)`)
    - Transitions and keyframes for smooth popups and modals
    - Media queries for mobile/responsive adjustments
- **JavaScript (ES6+)**
    - **OOP & Prototype Inheritance**
        - `function MemoryGame(…) { … }` as base constructor
        - `EnhancedMemoryGame` extends `MemoryGame` via `Object.create(...)`
    - **jQuery 3.6.0** for DOM manipulation and event handling
    - **LocalStorage** for storing users, current user, and saved results
    - **Fetch API** to load SVG files and inline them dynamically
    - **Notification API** for push notifications on game completion
    - **Drag & Drop API** (native HTML5) for matched cards
    - **History API** (`history.pushState`, `window.onpopstate`) for state snapshots
    - **WebSocket API** for a simple chat-echo stub
- **Assets**
    - **SVG Icons** (`assets/svg/icon1.svg` through `icon18.svg`) used on card backs
    - **Audio Files** (`assets/audio/flip.mp3`, `match.mp3`, `background.mp3`)
    - **PWA Manifest** (`manifest.json`) with 192×192 & 512×512 SVG icons

---

File Structure

```
/ (project root)
├── index.html              # Login / Registration page
├── game.html               # Main game interface
├── results.html            # Saved results display
├── manifest.json           # PWA manifest file
├── sw.js                   # Service Worker for caching
├── css/
│   └── styles.css          # Main stylesheet
├── js/
│   ├── app.js              # Initialization, SW registration, auth-check
│   ├── auth.js             # Login/Registration logic
│   ├── game.js             # MemoryGame + EnhancedMemoryGame logic
│   └── results.js          # Results-page rendering & controls
├── assets/
│   ├── audio/
│   │   ├── flip.mp3
│   │   ├── match.mp3
│   │   └── background.mp3
│   └── svg/
│       ├── icon1.svg
│       ├── icon2.svg
│       └── … up to icon18.svg
```

Installation & Setup

1. **Clone or Download Repository and run Index.html**
2. **Serve via Static HTTP Server (For Service Worker to work)**
   - **Using Python:**

     ```
     python3 -m http.server 8000
     ```

     Then navigate to `http://localhost:8000/index.html`.

3. **Open in Browser**
   Navigate to `http://localhost:8000/index.html` (or the URL your server printed). The **Login / Register** page should appear.
4. **Allow Notifications & Location (Optional)**
   - On the login page, if you permit geolocation, you'll see your coordinates displayed.
   - When you first load the game page, the app will ask for Notification permission. Grant it so that the "Game Completed" notification can appear at the end.
5. **Install as PWA (Optional)**
   On supported browsers (Chrome/Edge/Firefox mobile), you may see an "Install" prompt. Accepting installs the app to your home screen. The Service Worker ensures offline functionality.

Usage

1. **Register or Log In**
   - On **index.html**, enter a **Username** and **Password**.
   - If the username is new, the app registers it in `localStorage`. If it already exists, the app checks the password.
   - On success, you are redirected to `game.html`. On failure, you see an "Incorrect password" or "Please enter valid credentials" alert.
2. **Select Difficulty & Start Game**
   - On **game.html**, choose one of the three difficulty options from the dropdown (`4 × 4`, `4 × 6`, `6 × 6`).
   - Click **Start Game** → all cards flip face-up for 5 seconds; background music begins when you click the first card.
   - After 5 seconds, cards flip back to their hidden side, the timer begins, and you may start revealing cards.
3. **Play the Memory Game**
   - Click on any card to flip it. The flip animation and corresponding audio play.
   - Click a second card:
     - If it matches the first, both cards stay face-up, play a "match" sound, gain the `.matched` CSS class, and become draggable.
     - If not a match, both cards flip back face-down after a one-second delay.
   - The **Moves** counter increments each time you select a second card (regardless of match).
   - The **Timer** shows elapsed time in `MM:SS` format.
4. **Drag Matched Pairs into the Drop Zone**

   - Once two cards are matched, drag either of them into the “Matched Pairs (Drag Here)” dashed box below the grid.
   - On drop:
     1. A clone of the dragged card appears in the matched zone (with a slight “archived” opacity).
     2. The original card and its twin sibling (the other card with the same faceValue) receive `hidden-card` (visibility: hidden), preserving blank space in the grid.
     3. If more than four cards occupy the matched zone’s width, they wrap onto the next line, just as in the grid itself.

5. **Game Completion**
   - Once **all** pairs are found (every card has `.matched`), the game automatically calls `endGame()`:
     - Timer stops.
     - A modal popup appears (“Game Over! Your Time: MM:SS Your Moves: N”).
     - If Notification permission was granted, a push notification appears (“Memory Game Completed! You finished in N moves and MM:SS.”).
   - In the modal, click **Save Result** or **Play Again**:
     - **Save Result**:
       1. If online, the app records an object `{ date, username, difficulty, moves, time }` into `memoryGameResults` in `localStorage` and navigates to `results.html`.
       2. If offline, you see an alert “You are offline. Cannot save the result.”
     - **Play Again** simply reloads `game.html`, resetting to the “Start Game” state.
6. **Results Page**
   - On `results.html`, you see:
     - A “Back to Game” button (← Play Again) and a “Logout” button in the header.
     - A **Results Controls** panel with:
       - A “Filter by Username” text input (live-filters results as you type).
       - “Export to CSV” (downloads `memory_game_results.csv` including Date, Username, Difficulty, Moves, Time).
       - “Clear History” (prompts “Are you sure…?”; if confirmed, removes `memoryGameResults` entirely).
     - A **Results Table** card:
       - Column headers: Date | Username | Difficulty | Moves | Time
       - Each row is one saved result (newest first).
       - If there are zero results, a “No saved results.” italic message appears instead of the table.
     - Footer shows the current year and “Memory Game.”
7. **Logout**
   - Clicking any **Logout** button (on game or results page) calls `localStorage.removeItem('memoryGameUser')` and redirects to `index.html`. The user must log in again to play.

Detailed Functionality

Authentication (Login/Registration)

- **Location:** `index.html` + `js/auth.js`
- **Flow:**
    1. User types username/password in the form.
    2. On submit, `auth.js` reads `localStorage.getItem('memoryGameUsers')` (JSON array).
    3. If no existing user with that username, registers a new entry `{ username, password }` and sets `localStorage.setItem('memoryGameUser', username)`.
    4. If user exists, verifies password; if correct, sets `memoryGameUser` and redirects; if incorrect, alerts “Incorrect password.”
    5. On any page except `index.html`, `app.js` checks `localStorage.getItem('memoryGameUser')`; if missing, redirects to `index.html`.
- **Offline Behavior:**
    - If user is offline on the login page, the form still submits (no server).
    - On game/results pages, if offline, a red “You are offline.” message appears under the game info or login section.

Game Page

- **Location:** `game.html` + `js/app.js` + `js/game.js` + `css/styles.css`
- **Structure (`game.html`):**

1. **Initialization (`$(document).ready` in `app.js`)** - Set current year in footers. - Detect online/offline; show/hide `#offline-status-game`. - Register Service Worker (`sw.js`). - Check `localStorage.getItem('memoryGameUser')`; if missing, redirect to `index.html`. - Logout button removes `memoryGameUser` & redirects to login. - If on `index.html`, request geolocation and display coordinates under `<p id="user-location">`.

2. **EnhancedMemoryGame Constructor (in `game.js`)**
    - Inherits from `MemoryGame` which implements core logic.
    - Sets up drag & drop, inlines SVG icons, initializes WebSocket echo stub, listens to `popstate`.
3. **Start Game Sequence (`EnhancedMemoryGame.prototype.performStartSequence`)**
    - Sets `this.gameStarted = true` and disables the difficulty dropdown.
    - Flips all cards face-up and plays background music.
    - After a 5 second timeout, flips unmatched cards back and calls `this.startTimer()`.
    - Pushes the initial state into `history.state`.
4. **Game Loop (`MemoryGame.prototype.handleClick`, `.flipCard`, `.endGame`)**
    - On each card click, flips card, plays `flip-sound`, and compares two selected cards.
    - If matched, plays `match-sound`, adds `.matched` class, calls `_pushHistoryState()`, and if all matched, calls `endGame()`.
    - If not matched, flips both back after 1 second and calls `_pushHistoryState()`.
    - Updating moves count, updating DOM elements for `#move-count` and `#timer`.

5. **Drag & Drop**
   - In `EnhancedMemoryGame.renderGrid()`, after rendering all cards, attach `dragstart` listeners to `.card`.
   - In the matched-zone drop handler:
     1. Clone the dragged card element (`.card.archived`) and append it to `#matched-zone`.
     2. Add `.hidden-card` to both the original and its twin sibling, preserving empty grid slots.
     3. As more cards fill the zone, CSS flex-wrap ensures extra cards flow to a new row.
6. **History API**
   - `_pushHistoryState()` collects an array of `{id, isFlipped, isMatched}` for each card, plus moves/time/difficulty/rows/cols, encodes it, and pushes a new URL (`?state=…`).
   - `_listenHistoryPopState()` binds to `window.onpopstate` and calls `_restoreState(state)` to re-apply card flips, matches, move counts, and timer.
7. **WebSocket Stub**
   - `this.socket = new WebSocket('wss://echo.websocket.org');`
   - On `open`, `message`, `close`, and `error`, log to the "chat-log" (`#chat-log`).
   - `#chat-send-btn` reads `#chat-input`, sends the text via `socket.send(msg)`, and appends "You: …" to chat-log.
8. **End Game Modal**
   - Once all pairs are matched, `endGame()` stops timer, sets `this.gameOver = true`, updates `#final-time`/`#final-moves`, reveals `#game-over-modal`.
   - If Notification permission is "granted," calls `new Notification(...)` with icon `assets/svg/icon1.svg`.
9. **Save & Play Again Buttons**
   - **Save Result (`#save-result-btn`)**: See earlier section in this doc. Stores `{ date, username, difficulty, moves, time }` and navigates to `results.html`.
   - **Play Again (`#play-again-btn`)**: `window.location.reload()` resets the page to pre-start state.

Results Page

- **Location:** `results.html` + `js/results.js` + `css/styles.css`
- **Rendering Logic (in `results.js`):**
  1. On document ready, get `memoryGameUser`; if missing, redirect to `index.html`.
  2. Define `loadResults()` → returns parsed array from `localStorage.getItem('memoryGameResults')` or `[ ]`.
  3. Define `renderResults(filter)` → filters by username substring, empties `#results-body`, and:
     - If no results remain, show `#no-results-msg`.
     - Otherwise sort descending by date and append one `<tr>` per result
  4. Bind `#filter-username on('input')` to re-call `renderResults(...)`.
  5. Bind `#clear-history-btn` to confirm & `localStorage.removeItem('memoryGameResults')`.
  6. Bind `#export-csv-btn` to build a CSV string with header `Date,Username,Difficulty,Moves,Time\n` and each row's values; generate a Blob, create a temporary `<a>`, and call `click()` to download.

7. Bind `#back-to-game-btn` to `window.location.href = 'game.html'`.
8. Bind `#logout-btn-results` to `localStorage.removeItem('memoryGameUser')` + `window.location.href = 'index.html'`.

Offline / Service Worker

- **Location:** `sw.js`
- **Install Handler:**

```
const CACHE_NAME = 'memory-game-cache-v2';
const urlsToCache = [
  'index.html',
  'game.html',
  'results.html',
  'css/styles.css',
  'js/app.js',
  'js/auth.js',
  'js/game.js',
  'js/results.js',
  'assets/audio/flip.mp3',
  'assets/audio/match.mp3',
  'assets/audio/background.mp3',
  'manifest.json',
  'https://code.jquery.com/jquery-3.6.0.min.js',
  'assets/svg/icon1.svg',
  'assets/svg/icon2.svg',
  /* …through icon18.svg… */
  'assets/svg/icon18.svg'
];

self.addEventListener('install', event => {
  event.waitUntil(
    caches.open(CACHE_NAME)
      .then(cache => {
        return cache.addAll(urlsToCache);
      })
      .catch(err => {
        console.error('Cache.addAll failed:', err);
        // Optionally log which URL(s) failed by individually calling
fetch(url)
      });
  );
});
```

- **Fetch Handler:**

```
self.addEventListener('fetch', event => {
  event.respondWith(
    caches.match(event.request)
      .then(resp => resp || fetch(event.request))
  );
});
```

- **Activate Handler (cache cleanup):**

```
self.addEventListener('activate', event => {
  const whitelist = [CACHE_NAME];
  event.waitUntil(
    caches.keys().then(cacheNames =>
      Promise.all(
        cacheNames.map(name => {
          if (!whitelist.includes(name)) {
            return caches.delete(name);
          }
        })
      )
    )
  );
});
```

Styles & Layout

- **CSS variables:** None—straight classic style rules.
- **Main layout** uses a three-column flex container on the game page:
  1. Left column: **Game Info & Controls** (fixed width ~250px)
  2. Center column: **Card Grid** + **Matched Pairs Zone** + **Game Rules**
  3. Right column: **Game Chat** (fixed width ~250px)
- **Responsive adjustments** via a single media query (`@media (max-width: 600px)`):
  - `.game-info`, `.results-controls`, and `.results-table-card` adjust to vertical stacking and narrower widths.
  - `.game-grid` max-width changes to `90vw`.
- **Card Grid**
  - `.game-grid.size-4x4`: `grid-template-columns: repeat(4, 100px)`
  - `.game-grid.size-4x6`: `grid-template-columns: repeat(4, 80px)`
  - `.game-grid.size-6x6`: `grid-template-columns: repeat(6, 60px)`
  - Each `.card`:

    ```
    position: relative;
    width: 100%;           /* fills the grid cell */
    padding-top: 100%;     /* 1:1 aspect ratio → square */
    perspective: 1000px;   /* 3D flip container */
    cursor: pointer;
    ```

  - `.card__inner` toggles `transform: rotateY(180deg)` when the parent `.card` gets `.flipped`.
  - `.card__face--back svg { width: 60%; height: 60%; }` constrains icons the same as before.
- **Matched Pairs Zone**
  - `#matched-zone` is a flex container, `flex-wrap: wrap; gap: 10px;`
  - Each cloned `.card` inside the matched zone uses fixed `width: 100px; padding-top: 100px;` so they never grow too large.
  - Once four 100×100 cards plus 3 gaps × 10px fill the row (430px total), the next card wraps to a new line.
- **Modals & Overlays**
  - `.modal { position: fixed; top: 0; left: 0; width: 100%; height: 100%; background: rgba(0,0,0,0.5); display: flex; justify-content: center; align-items: center; }`
  - `.modal-content { background: #fff; padding: 2rem; border-radius: 8px; animation: fadeIn 0.5s; }`
  - Keyframe `@keyframes fadeIn { from { opacity: 0; transform: scale(0.9); } to { opacity: 1; transform: scale(1); } }`

- **Results Table Card**
    - `.results-table-card { background: #fff; border-radius: 8px; box-shadow: 0 2px 8px rgba(0,0,0,0.1); padding: 1rem; margin-bottom: 2rem; overflow-x: auto; }`
    - `.results-table thead th { background-color: #2196F3; color: #fff; padding: 0.75rem 1rem; }`
    - `.results-table tbody td { padding: 0.6rem 1rem; }`
    - Alternating row background stripe: `tbody tr:nth-child(even) { background-color: #f9f9f9; }`
    - `.no-results { text-align: center; font-style: italic; color: #777; padding: 1rem 0; }`
- **Buttons**
    - `.btn { padding: 0.6rem 1.2rem; border-radius: 4px; cursor: pointer; transition: background-color 0.25s, transform 0.15s; }`
    - `.btn-primary { background-color: #2196F3; color: #fff; }`
    - `.btn-primary:hover { background-color: #1976D2; transform: translateY(-1px); }`
    - `.btn-secondary { background-color: #E0E0E0; color: #333; }`
    - `.btn-danger { background-color: #F44336; color: #fff; }`

Asset Management

- **SVG Icons (assets/svg/icon1.svg … icon18.svg)**
    - Each `iconN.svg` represents a unique card face.
    - In `game.js`, after rendering `.card` elements, `_inlineAllSVGs()` fetches each `iconN.svg` via `fetch`, parses it as XML, and replaces the placeholder `<div class="svg-placeholder">` with the actual inline `<svg>…</svg>`.
    - Each inline `<svg>` has a click listener that briefly highlights it (CSS `filter: brightness(1.2)`), demonstrating event handling on SVG.
- **Audio Files (assets/audio/*)**
    - `flip.mp3` (short card-flip sound).
    - `match.mp3` (success chime for a matched pair).
    - `background.mp3` (looping background music).
    - All three are loaded via `<audio id="…">` elements with `preload="auto"`.
    - The game uses `this.audioFlip.play()`, `this.audioMatch.play()`, and `this.bgMusic.play()` (with appropriate error catching for autoplay restrictions).
- **Manifest (manifest.json)**

```
{
"name": "Memory Game",
"short_name": "Memory",
"start_url": "index.html",
"display": "standalone",
"background_color": "#ffffff",
"theme_color": "#2196f3",
"icons": [
  {
    "src": "assets/svg/icon1.svg",
    "sizes": "192x192",
    "type": "image/svg+xml"
  },
  {
    "src": "assets/svg/icon2.svg",
    "sizes": "512x512",
    "type": "image/svg+xml"
    }
  ]
}
```

Progressive Web App (PWA) Configuration

1. **Register Service Worker**
   In `js/app.js` inside `$(document).ready(...)`:

   ```
   if ('serviceWorker' in navigator) {
     navigator.serviceWorker
       .register('sw.js')
       .then(() => console.log('Service Worker registered'))
       .catch(err => console.error('SW registration error:', err));
   }
   ```

2. **Manifest Link**
   In `index.html`, include:

   ```
   <link rel="manifest" href="manifest.json">
   ```

3. **Icon Caching in `sw.js`**
   All `assets/svg/iconN.svg` files appear in the `urlsToCache` array. This means once installed, the PWA will work offline, serving icons, audio, CSS, and JS from the cache.
4. **Offline Fallback**
   The fetch handler:

   ```
   event.respondWith(
     caches.match(event.request).then(resp => resp ||
   fetch(event.request))
   );
   ```

   ensures that, if a resource is in the cache (e.g. `index.html`, `css/styles.css`), it gets served offline; otherwise the request falls back to the network.