Movie Listing React Application — Documentation

Created: 01 September 2025

---

1. Project Overview

Name: Movie Listing React Application

Problem statement: Create a React application that fetches real movie data from The Movie Database (TMDb) API. The app should provide filtering and sorting features, and — as a bonus — support light and dark theme modes.

Purpose of this document (SRD): This Software Requirements Document (SRD) captures functional and non-functional requirements, high-level architecture, use cases, data model, component breakdown, sequence and deployment diagrams, test strategies, and acceptance criteria to guide implementation and QA.

Audience: Product owner, developers, QA, DevOps, designers.

---

2. Goals & Success Criteria

Primary goals:

- Fetch and display movie data from TMDb.
- Provide filtering (by genre, release year, rating) and sorting (by popularity, release date, rating).
- Support responsive UI and good UX for discoverability.

Bonus goal: Dark/light theme toggle persisted across sessions.

Success criteria / Acceptance tests:

- App fetches and displays lists of movies with poster, title, release date and rating.
- Filtering and sorting update results without a full page reload.

- Theme toggle updates UI instantly and persists across browser refreshes.
- Proper error handling for API failures and clear user-facing messages.

---

3. Stakeholders

- Product owner / Client
- Frontend developer(s)
- QA engineer
- Designer
- DevOps engineer

---

4. Terminology & Constraints

- **TMDb**: The Movie Database (https://www.themoviedb.org/) — requires API key.
- **Client-only app**: App will be implemented as a single-page React app. For security of the TMDb API key, a small proxy or environment variable approach is recommended (see Security section).
- **Rate limits**: TMDb enforces request limits; implement caching/pagination to reduce calls.

---

5. Functional Requirements

1. **FR1 — Browse Movies**: App shall display a paginated list of movies from TMDb (default: popular movies).
2. **FR2 — Movie Details**: App shall show a details view (modal or dedicated route) containing overview, cast (optional), genres, and links to trailer (if available).
3. **FR3 — Filtering**: App shall allow filtering by genre, release year, and minimum rating.
4. **FR4 — Sorting**: App shall allow sorting by popularity, release date, and rating — each ascending/descending.
5. **FR5 — Search**: App shall allow free-text search on movie titles (debounced input).

6. **FR6 — Theme**: App shall support toggleable dark and light themes; user preference shall persist in localStorage.
7. **FR7 — Responsive**: App shall render correctly on desktop, tablet, and mobile.
8. **FR8 — Error Handling**: App shall show friendly messages for network/API errors and when no results are found.

---

6. Non-functional Requirements

- **NFR1 — Performance:** First meaningful paint ≤ 1.5s on decent connections; list updates (filter/sort) under 300ms for cached requests.
- **NFR2 — Accessibility:** Keyboard navigable, semantic HTML, color contrast for both themes (WCAG AA preferred).
- **NFR3 — Security:** API key must not be exposed publicly in VCS; use environment variables or server-side proxy. Avoid storing secrets in client-side code.
- **NFR4 — Maintainability:** Code must follow component-based architecture, documented, and covered by unit tests for core logic.
- **NFR5 — Scalability:** Support large result sets via pagination / infinite scroll.

---

7. Data Model (frontend view)

This app primarily consumes TMDb JSON. Example simplified model used in the UI:

```
Movie {
  id: number,
  title: string,
  poster_path: string | null,
  release_date: string | null,
  vote_average: number,
  overview: string,
  genre_ids: number[],
  popularity: number
}
```

Also store Genre { id: number, name: string } from TMDb Genres endpoint.

---

8. API Integration & Backend considerations

Primary endpoints (TMDb public API): - /movie/popular — popular movies - /search/movie — search movies by query - /movie/{movie_id} — movie details - /genre/movie/list — genre list

Key notes: - Use query params for page, sort_by (when using discover endpoint), with_genres, primary_release_year, and vote_average.gte (or client-side filter for rating). - Implement client-side caching (in-memory or local caching layer) to avoid repeating identical queries.

Security option A (recommended for production): small server-side proxy endpoint that injects the TMDb API key and forwards requests.

Security option B (for quick demos only): use environment variables and embed a read-only key during build, with caution.

Rate limit handling: detect 429 responses and surface a user-friendly retry message; exponential backoff when retrying programmatically.

---

9. UI / UX Design

Primary screens: - Home / Browse: grid of movie cards with poster, title, rating and release year. - Details: modal or route /movies/:id showing extended information. - Filters/Controls: sidebar or top bar containing genre multi-select, year dropdown or range, rating slider, sort dropdown, and search box. - Settings: theme toggle.

Interaction patterns: - Filtering and sorting should be instant and show a loading indicator while fetching. - Search input must be debounced (e.g., 400ms). - Pagination: either infinite scroll or classic paged controls (choose one; default: paged controls for simplicity).

---

10. Use Cases

Use Case 1 — Browse and Discover Movies

Primary actor: End user

Preconditions: User opens the app

Main success scenario: 1. App loads and fetches first page of popular movies. 2. User scrolls or navigates to next pages. 3. App displays movie cards in grid.

Alternate flows: API error → show retry button.

Use Case 2 — Filter by Genre / Year / Rating

Primary actor: End user

Main success scenario: 1. User selects one or more genres. 2. User optionally sets a minimum rating or specific year. 3. App fetches and displays filtered results.

Use Case 3 — Sort Results

Primary actor: End user

Main success scenario: 1. User selects sort option (e.g., Release Date desc). 2. App updates results accordingly.

Use Case 4 — View Movie Details

Primary actor: End user

Main success scenario: 1. User clicks a movie card. 2. App opens detail view (modal or route) and fetches movie details. 3. User reads details and optionally opens trailer link.

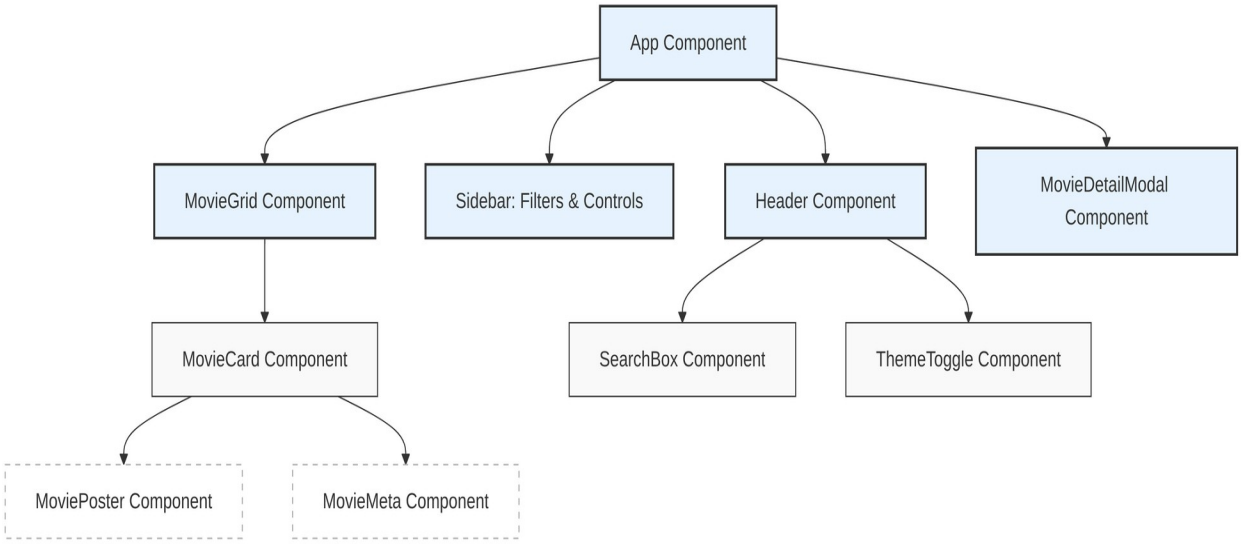
Use Case 5 — Toggle Theme

Primary actor: End user

Main success scenario: 1. User toggles theme switch. 2. UI updates to dark/light theme instantly. 3. Preference is stored in localStorage.

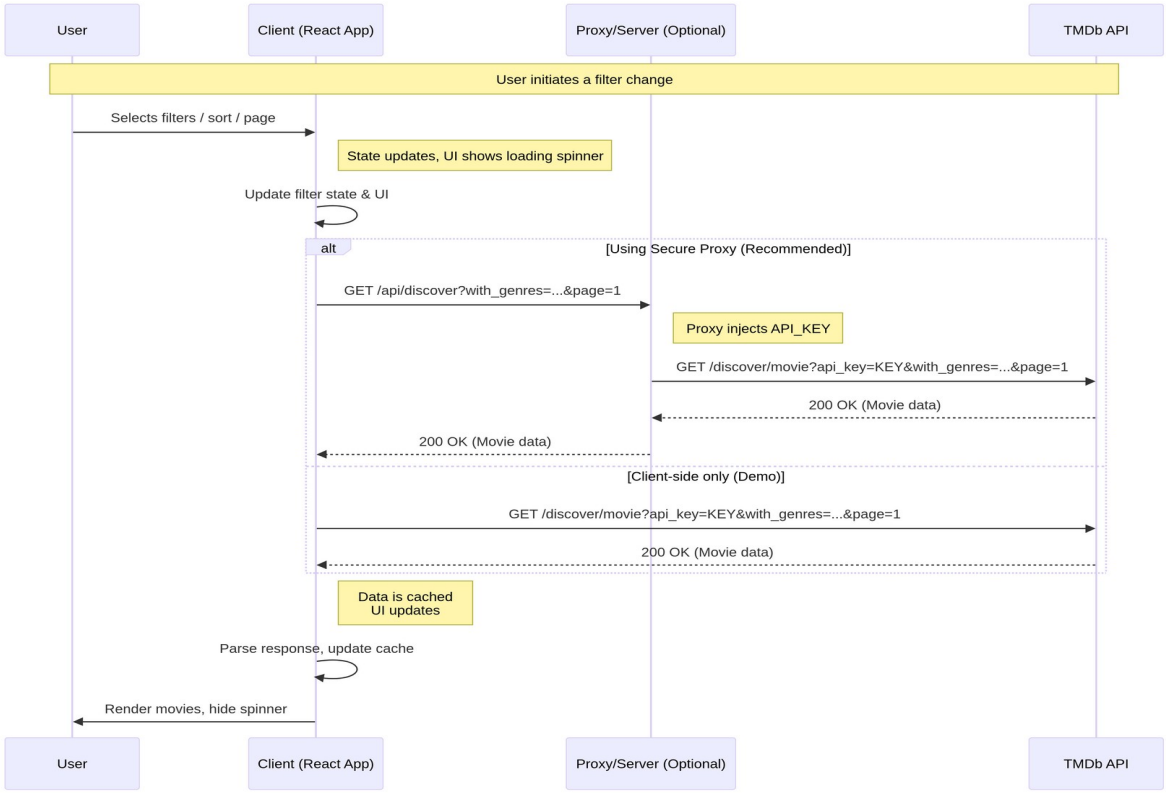
---

11. UI Component Diagram (React)

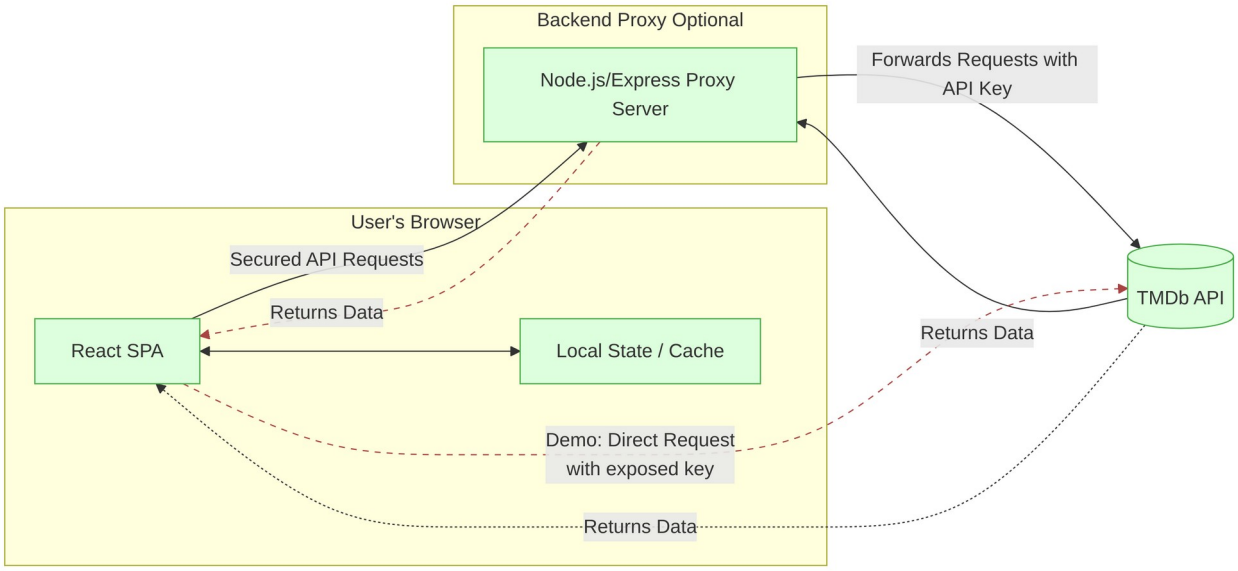


---

12. Sequence Diagram — Fetch & Filter Flow



13. High-level Architecture Diagram



Deployment: Static hosting (Netlify, Vercel, GitHub Pages) for client; optional Node/Express small API on Vercel/Heroku for proxy.

---

14. Implementation Notes & Choices

- **React version:** v18+ (functional components + hooks)
- **Styling:** Tailwind CSS or CSS Modules. For theme, use CSS variables (custom properties) toggled at `:root` or via `data-theme` attribute.
- **State management:** React Context for theme; local component state + query caching for movie queries. Consider React Query (TanStack Query) for data fetching, caching, stale-while-revalidate behavior and retry/backoff.
- **Routing:** React Router for navigation to `/movies/:id` (optional). Use modal overlay for details on top of list.
- **Testing:** Jest + React Testing Library for unit and integration tests. E2E tests using Playwright or Cypress (basic flows).

- **Icons & images:** Use TMDb image base URL with appropriate size (w342, w500). Provide placeholder when `poster_path` is null.

---

15. Error Handling & Edge Cases

- Handle missing poster images gracefully.
- If TMDb returns partial data (missing release date), display — or `TBD`.
- When no results found after filtering, show a friendly explanation and a clear `Reset filters` action.
- For network failures show `Unable to reach movies service. Retry` and an exponential-backoff retry for background refresh.

---

16. Security & Secrets

- **Never commit API keys** to public VCS.
- Use `.env` files with `REACT_APP_TMDB_KEY` for local development and CI/CD environment variables for deployment.
- For production, prefer a minimal backend proxy which stores the key securely and forwards requests.
- When using proxy, implement simple rate-limiting & caching to reduce calls to TMDb.

---

17. Testing Strategy

- **Unit tests:** components (MovieCard, Filtering logic), utilities (format dates), and reducers/hooks.
- **Integration tests:** pages (browse -> filter -> open detail), search debounce behavior.
- **E2E tests:** critical flows: initial load, filter+sort, view details, theme toggle persistence.
- **Accessibility tests:** axe-core integration for automated checks.

---

18. Acceptance Criteria (detailed)

- ☐ App displays popular movies on first load.
- ☐ User can search with debounced input; results update accordingly.
- ☐ User can filter by genre, release year and minimum rating and results reflect selection.
- ☐ User can sort results by popularity, date and rating.
- ☐ Movie details view loads full details and gracefully handles missing fields.
- ☐ Theme toggle updates UI and persists between sessions.
- ☐ App handles API errors and 429 rate limits with user-facing messages.

---

19. Developer Checklist — Minimum Viable Implementation

1. Create React app skeleton and layout (Header, Sidebar/Controls, MovieGrid).
2. Implement TMDb integration for /movie/popular and /genre/movie/list.
3. Render MovieCard with poster, title, release year, rating.
4. Implement filtering controls wired to API (or client-side filter for certain fields).
5. Implement sorting control and pagination.
6. Add details view (modal or route) fetching /movie/{id}.
7. Add theme toggle with persistence.
8. Add basic tests and CI linting.

---

20. Future Enhancements (post-MVP)

- Add user authentication and favorites / watchlist using a backend and database.
- Add cast details and external trailers embedded from YouTube.
- Add offline-first experience / PWA capabilities.
- Add localisation and multiple languages.

---

21. Appendix

Helpful TMDb docs: *(developer to look up latest URLs & rate limits on TMDb site when implementing)*

Example image URL: https://image.tmdb.org/t/p/w342/{poster_path}

Example discover query (discover endpoint):

https://api.themoviedb.org/3/discover/movie?
api_key=<<KEY>>&page=1&with_genres=18&sort_by=popularity.desc

---

End of document.