# React & Next Js

## Take-Home Assignment: TMDB Movie Explorer (Next.js SSR)

### 1. Overview

**Experience level:** ~2 years

**Time limit:** 3 to 4 hours

**Objective:** Demonstrate practical capability with Next.js App Router, server-side rendering, API integration, caching, error handling, testing, and deployment.

This is not a UI design exercise. You can build the UI as per your choice. Functional correctness, architecture, and production readiness matter more than visual polish.

---

### 2. What You Are Expected to Build

Build a small Movie Explorer application using data from **The Movie Database (TMDB)**.

The application must include:

1. A movie search and listing page
2. A movie detail page
3. Server-side rendering for all data-driven pages
4. A backend-for-frontend layer using Next.js Route Handlers
5. A deployed, publicly accessible instance

You must implement the frontend yourself. No UI will be provided.

---

### 3. Technical Constraints (Strict)

The following are mandatory:

1. **TypeScript is required.** The codebase must compile with TypeScript without errors.
2. Next.js **App Router** using the `/app` directory
3. **Server Components by default**
4. **Client Components only where required** (for example, controlled inputs)
5. **Route Handlers** under `/app/api/*` for all TMDB calls

6. **No TMDB API calls from the browser**
7. **No secrets exposed to the client**
8. Deployment to **Netlify, Vercel, or equivalent**

Violation of these constraints will result in rejection.

---

## 4. TMDB API Setup

You must use a free TMDB account.

Steps:

1. Create or log in at https://www.themoviedb.org
2. Go to Account Settings → API https://www.themoviedb.org/settings/api
3. Request API access (free).
4. Copy your **API Read Access Token (v4)**.

Authentication header:

```
Authorization: Bearer <TMDB_READ_ACCESS_TOKEN>
```

Store the token in `.env.local`. It must never appear in client-side code or browser network requests.

TMDB API documentation: https://developer.themoviedb.org/docs

---

## 5. External API Endpoints You Must Use

All calls must be made from **Route Handlers only**.

### 5.1 Search Movies

```
GET https://api.themoviedb.org/3/search/movie
```

Required query parameters:

- `query`
- `page`
- `include_adult=false`

Optional:

- `language=en-US`

---

### 5.2 Movie Details

```
GET https://api.themoviedb.org/3/movie/{movie_id}
```

You must use:

```
append_to_response=videos,credits
```

---

### 5.3 Configuration (Images)

```
GET https://api.themoviedb.org/3/configuration
```

You must construct image URLs using the configuration response.

---

## 6. Internal API Contract (Your App)

You must expose the following Route Handlers.

### 6.1 Search

```
GET /api/movies/search?q=<string>&page=<number>
```

Validation rules:

- `q` is required, trimmed, minimum length 2
- `page` must be >= 1

Normalized response shape:

```
{
  "page": 1,
  "total_pages": 10,
  "total_results": 200,
  "results": [
    {
      "id": 123,
      "title": "Movie title",
      "release_date": "YYYY-MM-DD",
      "overview": "Overview text",
      "poster_url": "https://...",
      "vote_average": 7.5
    }
  ]
}
```

---

### 6.2 Movie Details

```
GET /api/movies/{id}
```

Response must include:

- Core movie details
- Genres
- Runtime
- Rating
- Top 5 cast members
- Trailers (YouTube keys where available)
- Poster and backdrop URLs

---

### 6.3 Configuration

```
GET /api/config
```

You may either:

- Expose only what the UI needs, or
- Keep configuration entirely server-side and return full URLs

Both approaches are acceptable if implemented correctly.

---

## 7. Pages and SSR Requirements

### 7.1 Search and List Page

Route: `/`

Must:

1. Be server-rendered
2. Read data from `searchParams` Example: `/?q=batman&page=1`
3. Support pagination via query parameters
4. Show:
    - Loading state
    - Empty state
    - Error state

Client-side fetching for primary data is not allowed.

---

### 7.2 Movie Detail Page

Route: `/movie/[id]`

Must:

1. Be server-rendered
2. Handle invalid IDs gracefully (404 or not-found UI)
3. Set page metadata (title and description) based on movie data

---

# 8. Caching and Rate Limiting

## 8.1 Caching (Required)

You must implement caching and explain your choice in the README:

- `revalidate: 60` for search and details (preferred), or
- `cache: 'no-store'` if intentionally choosing fresh data

The `/configuration` endpoint should be cached aggressively (for example, 24 hours).

---

## 8.2 Rate Limits

TMDB may return HTTP 429.

You must:

- Detect 429 responses
- Return a structured error from your API
- Show a meaningful user-facing message

---

# 9. Testing Requirements

Minimum **two automated tests** are required.

## Mandatory Test 1

Route Handler test:

- One success case
- One failure case (for example, upstream error or 429)

---

## Mandatory Test 2

Choose one:

- UI or component test for empty or error state
- Non-trivial utility test (for example, TMDB response mapping)

Tests must provide real signal. Trivial or placeholder tests are not acceptable.

---

## 10. Deployment Requirement

You must deploy the application to **Netlify, Vercel, or equivalent**.

Submission must include:

- Public URL of the deployed app
- Environment variables configured securely
- Production build running correctly

Local-only submissions are not acceptable.

---

## 11. Build and Quality Gate

The following must pass:

- `npm run build`
- `npm run start` (or equivalent production run)
- `npm run lint` (if configured)
- `npm run typecheck` (or equivalent TypeScript check)

---

## 12. Security Rules (Auto-Fail Conditions)

Any of the following result in rejection:

1. TypeScript compilation errors (including failing `npm run typecheck`)
2. TMDB token visible in browser network requests
3. TMDB token committed to the repository
4. Client-side calls to `api.themoviedb.org`
5. Client-only data fetching replacing SSR
6. Non-working deployed application

---

## 13. Submission Checklist

Provide:

1. Git repository

2. Deployed URL
3. `.env.example`
4. README including:
    - Setup instructions
    - Required environment variables
    - Caching strategy explanation
    - How to run tests and build

---

## 14. Evaluation Criteria

| Area | What Is Assessed |
|---|---|
| SSR | Correct server-side rendering using App Router |
| API Design | Clean Route Handlers, validation, normalized responses |
| Caching | Correct and intentional caching strategy |
| Error Handling | Graceful handling of failures and rate limits |
| Code Quality | Structure, separation of concerns, readability |
| Tests | Meaningful coverage, not superficial |
| Deployment | Working production deployment |

---

## 15. Reflection (Optional but Encouraged)

Briefly describe:

- One trade-off you made
- One improvement you would implement with more time