

Backend Take-Home Project

Movie Discovery Service — Take-Home

You are working at a new movie streaming startup. You are tasked to build the **movie discovery service**.

Database models (reference; you may improve)

- **users**
 - `id`
 - `name`
 - `date_of_birth`
- **genres**
 - `id`
 - `name` (*unique, case-insensitive*)
- **movies**
 - `id`
 - `title`
 - `year` (*integer*)
 - `genre_id` (*FK → genres.id*)
- **actors**
 - `id`
 - `name`
- **cast**
 - `movie_id` (*FK →  Movies.id - Info Teknologi Terkini*)
 - `actor_id` (*FK → actors.id*)
 - *PRIMARY KEY (movie_id, actor_id)*
- **countries**
 - `code` (*ISO-3166-1 alpha-2, PRIMARY KEY, e.g., US, BR*)

- name
- movie_availability
 - movie_id (FK → [Movies.id - Info Teknologi Terkini](#))
 - country_code (FK → countries.code)
 - PRIMARY KEY (movie_id, country_code)
- saved_movies
 - user_id (FK → [users.id](#))
 - movie_id (FK → [Movies.id - Info Teknologi Terkini](#))
 - date_added (timestamp)
 - PRIMARY KEY (user_id, movie_id)

Indexes recommended: movies(year) ,
 movie_availability(country_code, movie_id) ,
 saved_movies(user_id, date_added) .

APIs to implement

- List genres
 - Input: page (default 1), page_size (default 20, max 100)
 - Output: paged list of genres
- List movies
 - Input: country (ISO-3166-1 alpha-2), genre (genre id), page , page_size , sort (one of year , -year ; default -year)
 - Output: paged list of movies { id, title, year }
 - Details: return only movies available in country
- List movies saved by a user
 - Input: country , user_id , page , page_size , sort (one of date_added , -date_added ; default -date_added)
 - Output: paged list of movies { id, title, year }
 - Details: return only movies available in country and saved by the user
- Save a movie

- **Input:** `country`, `user_id`, `movie_id`
- **Output:** movie detail (define fields; at least `{ id, title, year }`)
- **Rules:**
 - Cannot save the same `(user_id, movie_id)` twice → **409 Conflict**
DUPLICATE_SAVE
 - Cannot save if movie is not available in `country` → **422 Unprocessable Entity**
UNAVAILABLE_IN_COUNTRY
- **Remove a saved movie**
 - **Input:** `user_id`, `movie_id`
 - **Output:** none
 - **Rules:**
 - If the movie does not exist → **404 Not Found** **NOT_SAVED**
(*You may choose idempotent 204 instead, but document it consistently.*)

Common API requirements

- **RESTful** resource design and **proper HTTP status codes**.
- **Authentication required** for non-public endpoints. You may choose the method (e.g., API key, JWT).
Provide a **seed credential** and document how to obtain/use it.
- **Pagination contract:** `page` is **1-based**, default 1; `page_size` default **20**, max **100**.
- **Paged response envelope** (all list endpoints):

```

1 | { "data": [...], "page": 1, "page_size": 20, "total": 123 }
2 |

```

- **Error response shape (all errors):**

```

1 | { "error": { "code": "STRING_CODE", "message": "Human readable", "details": {} } }
2 |

```

- **Country format:** `country` must be ISO-3166-1 alpha-2 (e.g., `US`, `BR`).

Project requirements (must-have)

- Use any backend framework in any language.
- Provide **database migrations** and **seed/dummy data** (suggestion: ~150 movies, 10–15 genres, 3–5 countries, 20–40 actors).
- **Tests:** at least **1 integration test per endpoint**. Include unit tests for:

- Availability filter
- Duplicate save prevention
- Saved list respects country filter
- **Docs:** OpenAPI 3 (YAML/JSON) + short README explaining setup, auth, and examples.
- **Runability:** one-command run (Docker Compose or Makefile), `.env.example`.

Things to consider (you will be evaluated on judgment)

- Improving the schema (e.g., multi-genre via `movie_genres`).
- Performance (indexes, avoiding N+1), security basics (input validation, injection prevention), and collaboration (linting, formatter, readable structure).
- Architectural patterns appropriate to scope (e.g., handlers/services/repos).