Contents

Introduction

Welcome to the Dev Test Activity! Prepare yourself for a variety of tasks. You can opt to work on the backend, frontend, or even both simultaneously (full stack).

The goal is to develop a Web Application for managing a movie catalog. The Dev Test is divided into two main Challenges:

- **Challenge 1:** This focuses specifically on developing a RESTful API. You will create this API from scratch using the technology stack you are most proficient in. Additionally, you will need to store data, with the option to use either in-memory structures or a relational/NoSQL database.
- **Challenge 2:** This revolves around UI development, where you will replicate provided designs in terms of styles. The UI you create will interact with an existing API sandbox. Like the previous challenge, you will have freedom to choose the stack you prefer.

Each challenge includes tasks of varying difficulty levels (Easy, Intermediate, and Hard**), as well as Bonus Tasks that encourage you to implement tests and full-stack development, integrating both frontend and backend**. There is no strict order to implement the challenges; it is up to you to tackle any task in any order. We recommend familiarizing yourself with all the tasks inside the challenges, so you can wisely prioritize your implementation. Select your preferred path and let us see what you are capable of!

Remarks

- You can consult external online sources like articles and documentation, but the use of AI code-generating tools is not allowed. If you reference external documentation, share the topics on your screen.
- During the dev test, will be Trainers will be on the call to interact, understand your solutions, and provide support. At the end of the test, there will be brief demos to show your progress and rationale.
- After the test, upload your code to a repository on GitLab, where it will be evaluated for cleanliness, organization, folder structure, and best practices. We will provide the repository link and credentials on the test day.

I. Backend Challenge

Task 1 (Easy – 10 pts): Implement an API for reading and creating movies

Acceptance Criteria:

- Develop a RESTful API service from scratch to manage movies, enabling users to perform the following actions:
  - Read the list of existing movies.
  - Create a new movie.
- Develop the service structure with a focus on best development practices like clean code, clear documentation, and coding standards.
- Choose any backend framework that fits your preferred technology stack for this service.
- Use “In Memory” persistence at this stage, selecting the best data structure like arrays, lists, dictionaries, or maps. No need to save movie data in a database yet.

API Specification:

API Endpoint to Get a list of existent movies	
Endpoint	GET /api/v1/movies
Response Status	“Ok” response
Response Body Example	`[ { "id": 1, "title": "Die Hard", "category": "Action", "releaseYear": 1988 }, { "id": 2, "title": "The Matrix", "category": "Science Fiction", "releaseYear": 1999 } ]`
API Endpoint to Create a movie	
Endpoint	POST /api/v1/movies
Request Body Example	`{ "title": "Shawshank Redemption", "category": "Drama", "releaseYear": 1994 }`
Response Status	“Created” response

Bonus Points (10 pts): Full stack integration

- Use this API to complete Frontend Challenge – Task 1.

Task 2 (Intermediate – 10 pts): Implement validations on adding a new movie entry

Acceptance Criteria:

- Implement validations for the title, category, and releaseYear fields before saving the movie when using the POST /api/v1/movies endpoint.
- The error response body will have the following Json schema:

```
{
    "message": "Unable to create the movie",
    "detail": "{{Provide a detailed explanation of the error}}"
}
```

Field Validation Specifications:

Title field must not be empty.	
Response Status	“Bad Request” response
Detail	'Title' must not be empty.
Title field should only allow letters, numbers, and spaces.	
Response Status	“Bad Request” response
Detail	'Title' contains invalid characters. Only letters, numbers and spaces are allowed.
Title field must be 50 characters or fewer	
Response Status	“Bad Request” response
Detail	The length of 'Title' must be 50 characters or fewer.
Category field must not be empty.	
Response Status	“Bad Request” response
Detail	'Category' must not be empty.
Category field only allows: Action, Science Fiction, Drama, Thriller, Horror, and Comedy.	
Response Status	“Bad Request” response
Detail	'Category' is not in the correct format. Only the following categories are allowed: Action, Science Fiction, Drama, Thriller, Horror, and Comedy.
Release Year field must have values between 1888 and the current year (2024)	
Response Status	“Bad Request” response
Detail	'Release Year' must be greater than or equal to '1888' and less than or equal to '2024'

Bonus Points (10 pts): Unit Testing

- To earn bonus points, ensure you include comprehensive test coverage, including unit tests specifically targeting the implemented validation logic.

Task 3 (Intermediate – 10 pts): Refactor the API to include movie ratings functionality

Acceptance Criteria:

- Implement an API endpoint to allow the addition of ratings to a specific movie.
- Furthermore, extend the GET /api/v1/movies endpoint to incorporate two additional fields in the movie schema:
- rateAverage: This field represents the average rating received by the movie.
- voteCount: This field indicates the total number of times the movie has been rated.
- Check that the rating is between 1 and 5 and make sure the movie ID for voting exists. The error response body will have the following Json schema:

```
{
    "message": "Unable to create the movie",
    "detail": "{{Provide a detailed explanation of the error}}"
}
```

API Specification:

API Endpoint to Create Ratings for an existing movie	
Endpoint	POST /api/v1/movies/{:id}/ratings
Request Body Example	`{ "value": 5 }`
Response Status	“Created” response
[EXTEND] API Endpoint to Get a list of existent movies with updated fields	
Endpoint	GET /api/v1/movies
Response Status	“Ok” response
Response Body Example	`[ { "id": 1, "title": "Die Hard", "category": "Action", "releaseYear": 1988, "rateAverage": 3.4, "voteCount": 11 }, { "id": 2, "title": "The Matrix", "category": "Science Fiction", "releaseYear": 1999, "rateAverage": 5, "voteCount": 1 } ]`

Validation Specifications:

Rating value should be between 1 and 5	
Response Status	“Bad Request” response
Detail	'Value' must be greater than or equal '1' and less than or equal to '5'
Movie id does not exist	
Response Status	“Not Found” response
Detail	The requested movie ID does not exist.

Bonus Points (10 pts): Full stack integration & Unit Testing

- Use this API to complete Frontend Challenge – Task 3.
- Implement unit tests to cover the validation logic.

Task 4 (Hard – 20 pts): Store API data in a database

For this task make sure your implementation includes the ability to integrate and utilize a database.

Acceptance Criteria:

- Store the API data in a database.
- Select the most appropriate database technology for your project's specific needs and requirements. Consider factors such as data structure, scalability, performance, etc.
- Select any database technology (SQL or NoSQL) for data storage.
- Employ the most effective database design principles to model your database.

II. Frontend Challenge

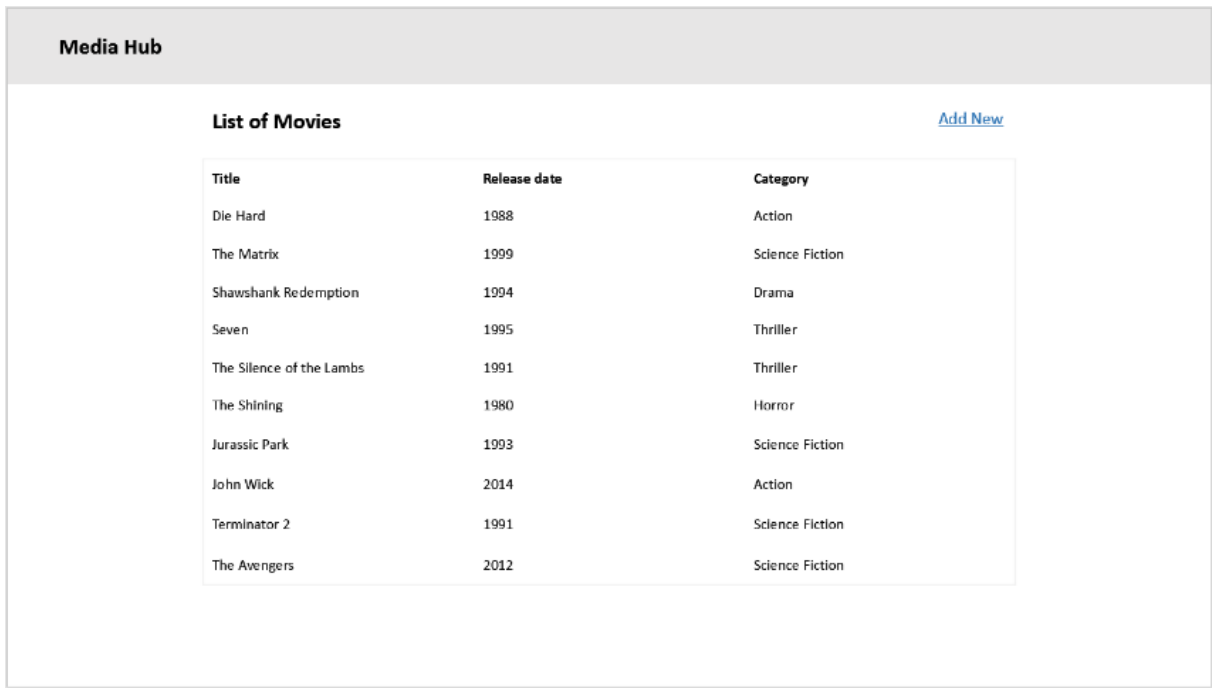
Task 1 (Easy - 10 pts): Create UI components for listing and creating movies

Acceptance Criteria:

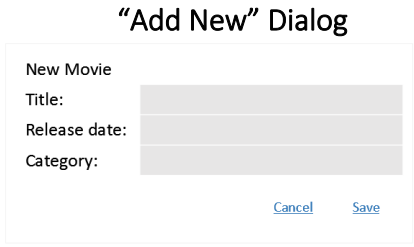
- Create a web-based user interface (UI) for managing movies. The UI will interact with a RESTful API, enabling users to perform the following actions:
  - Show the list of existing movies.
  - Create a new movie.
- Develop the UI structure to manage movies with a focus on best development practices like clean code, clear documentation, and coding standards.
- Choose any UI framework or libraries that fits your preferred technology stack.
- Follow the design specifications section to implement this task.

Design Specifications:

Main View



“Add New” Dialog



Sandbox Specifications:

You can use a deployed sandbox that provides all the necessary endpoints for implementing this task as UI only. You can find the API docs in here: https://sandbox-devtest.azurewebsites.net/swagger

Below are the endpoints available in the sandbox, which you are free to utilize to implement this task:

Sandbox API Endpoint to Get a list of existent movies	
Endpoint	GET https://sandbox-devtest.azurewebsites.net/api/v1/movies
Response Body Example	[{ "id": 1, "title": "Die Hard", "category": "Action", "releaseYear": 1988, "rateAverage": 4.5, "voteCount": 10 }, { "id": 2, "title": "The Matrix", "category": "Science Fiction", "releaseYear": 1999, "rateAverage": 5, "voteCount": 1 }]

Sandbox API Endpoint to Create a movie	
Endpoint	POST https://sandbox-devtest.azurewebsites.net/api/v1/movies
Request Body Example	`{ "title": "Shawshank Redemption", "category": "Drama", "releaseYear": 1994 }`

Bonus Points (10 pts): Full stack integration

- Consume the endpoints implemented in Backend Challenge – Task 1.

Task 2 (Intermediate - 10 pts): Implement validation mechanisms for creating a new movie entry

Before sending the request to the API we need to validate the inputs after pressing the 'save' button in the "new movie" dialog.

Acceptance Criteria:

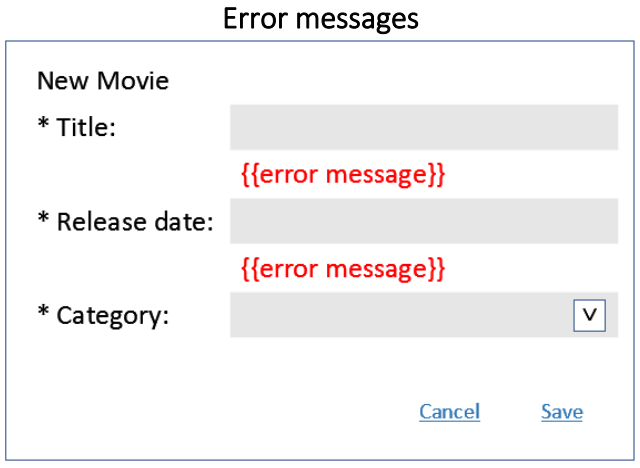
- Restrict title input to a maximum of 50 characters.
- Implement a dropdown menu to allow users to select from permitted categories.
- Display error messages when validation fails before executing the **POST** request to **/api/v1/movies** in the creation form.
- Handle unexpected API errors, such as HTTP status code 500, by implementing proper error-handling mechanisms.
- Follow the design specifications section to implement this task.

Field Validation Specifications:

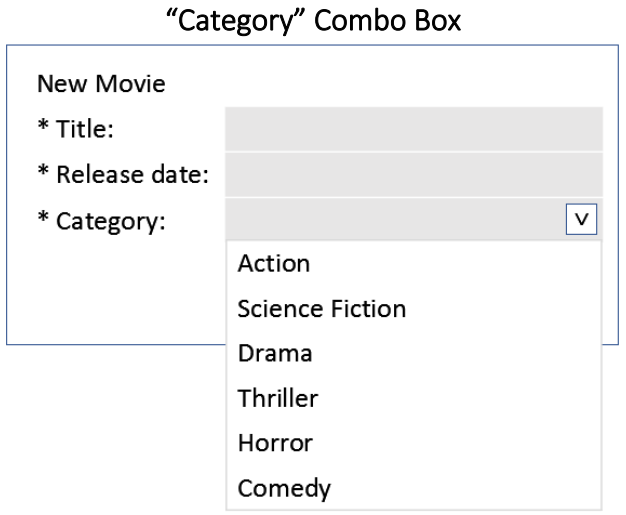
Title input must not be empty.	
Error message	Must not be empty.
Title field should only allow letters, numbers, and spaces.	
Error message	Only letters, numbers and spaces are allowed.
Category field must not be empty.	
Error message	Must not be empty.
Release Year field must have values between 1888 and the current year (2024)	
Error message	Must be greater than or equal to '1888' and less than or equal to '2024'
Handle unexpected API errors	
Toast Error Message	An error occurred while processing your request.

Design Specifications:

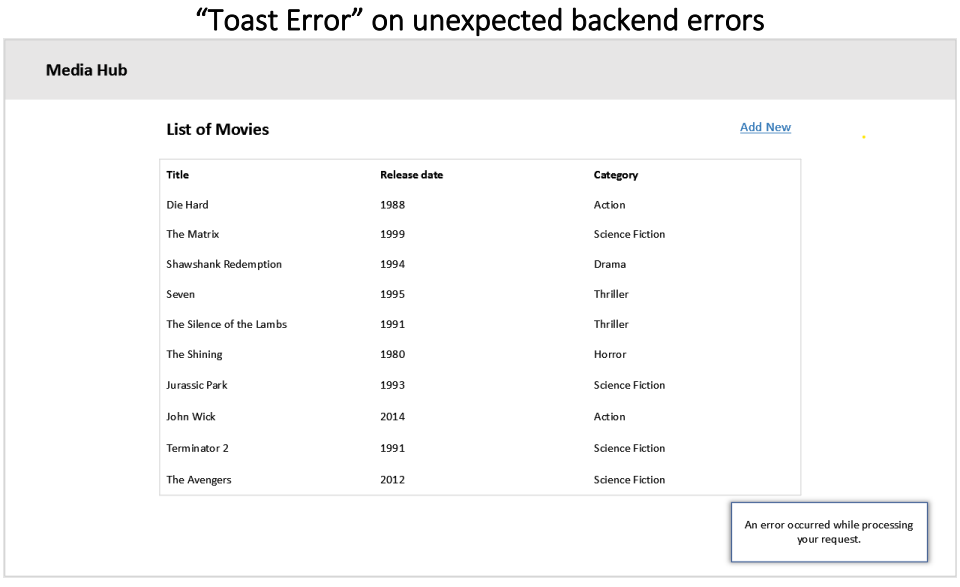
Error messages



"Category" Combo Box



"Toast Error" on unexpected backend errors



Sandbox Specifications:

To simulate unexpected backend errors, you can use the API 'v2' to create movies. By directing your frontend to the 'v2' API, you can simulate 500 errors on the create endpoint. Below is the endpoint available in the sandbox:

Sandbox API Endpoint that response 500 error while creating a movie	
Endpoint	POST https://sandbox-devtest.azurewebsites.net/api/v2/movies

Bonus Points (10 pts): Unit Testing

- To earn bonus points, ensure you include comprehensive test coverage, including unit tests specifically targeting the implemented validation logic.

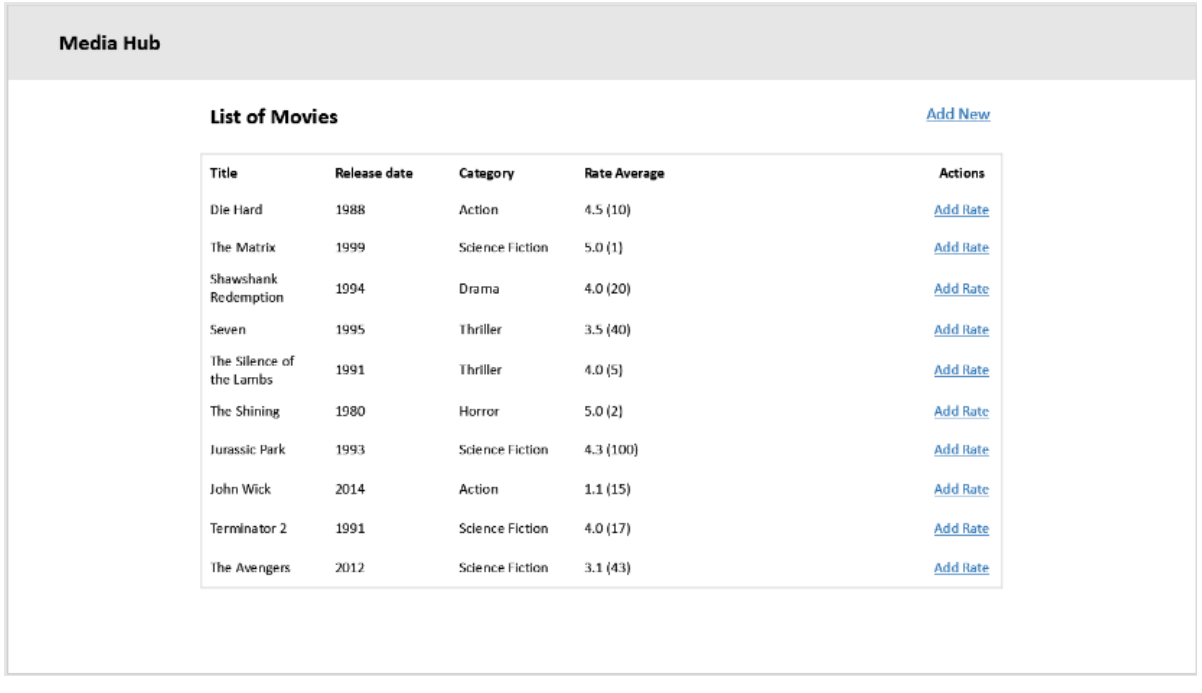
Task 3 (Intermediate – 10 pts): Implement movie ratings functionality

Acceptance Criteria

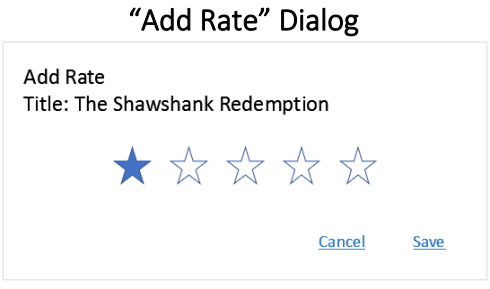
- Introduce a new column titled "Rate Average" in the movie database, displaying the average rating computed based on the received votes. This column will have two components:
    - The "rateAverage" value represents the average rating received by each movie, calculated from the accumulated votes.
    - The "voteCount" value (in parentheses) indicates the total number of times each movie has been rated by users.
- Add an "Actions" column that will provide a rating option for each movie. Clicking on this option will open a dialog where users can rate the movie.
- The minimum rating value is 1, and the maximum rating value is 5.
- Follow the design specifications section to implement this task.

Design Specifications:

Main View



"Add Rate" Dialog



Sandbox Specifications:

You can use a deployed sandbox that provides all the necessary endpoints for implementing this task as UI only. You can find the API docs in here: https://sandbox-devtest.azurewebsites.net/swagger

Below is the endpoint available in the sandbox, which you are free to utilize to implement this task:

Sandbox API Endpoint to Create movie ratings	
Endpoint	POST https://sandbox-devtest.azurewebsites.net/api/v1/movies/{id}/ratings • Replace {id} with the id of their movie to rate
Request Body Example	`{ "value": 5 }`

Bonus Points (10 pts): Full stack integration & Unit Testing

- Consume the endpoints implemented in Backend Challenge – Task 3.
- Implement unit tests to cover the validation logic.

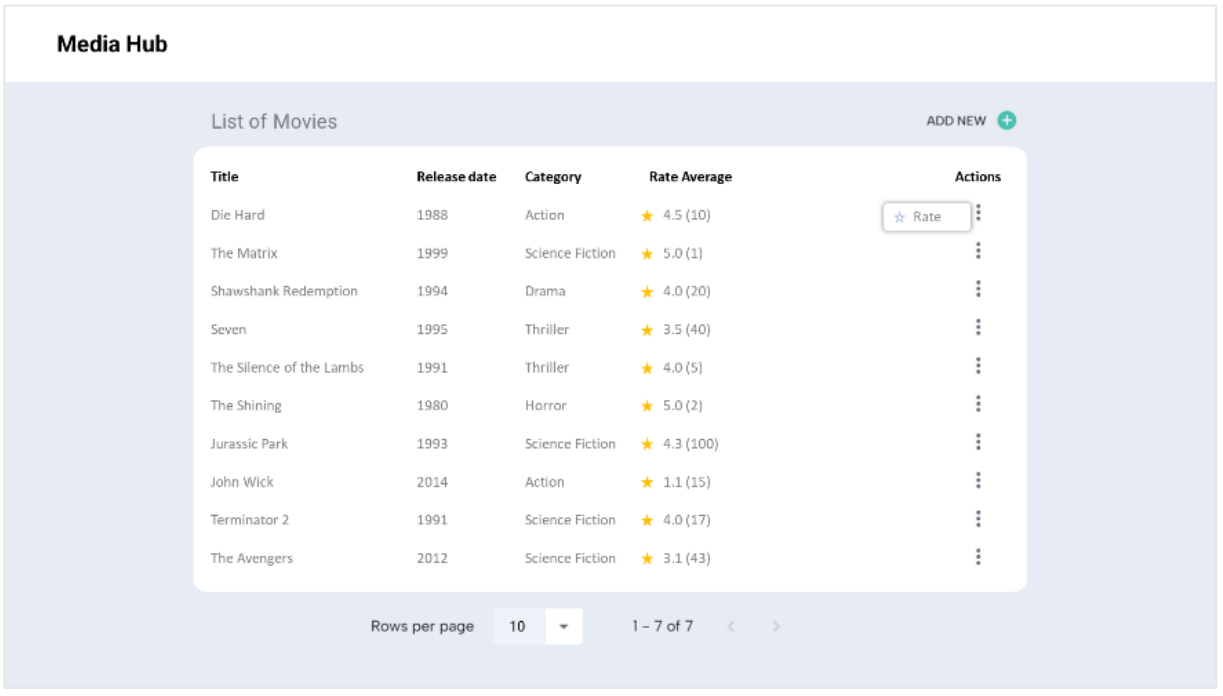
Task 4 (Hard – 20 pts): Enhance UI/UX

Acceptance Criteria

- Enhance the UI/UX for desktop mode according to the design specifications.
- Implement responsiveness for mobile devices.
- Validate desktop UI/UX enhancements align with design specifications.
- Ensure desktop mode improvements enhance user experience.
- Implement mobile responsiveness for seamless adaptation.
- Confirm consistent presentation and usability across Desktop and mobile sizes.
- Verify content accessibility and user-friendliness on mobile devices.
- Follow the design specifications section to implement this task

Design Specifications:

Desktop



Mobile mode