

PLASM PROJECT

Workflow (functions and scripts)

main.py

```
get_content_type()
get_title(content_type)
search_content(content_type, title)
select_from_results(results)
fetch_complete_data(content_type, api_id)
choose_review_type()
get_analytic_ratings(content_type)
get_subjective_rating()
get_common_review_data()
merge_all_data(api_data, review_type, ratings, common_data)
```

content_handler.py

api_manager.py

data_fusion.py

sql_handler.py

Workflow (structure)

1. Display menu:

Initialize configuration(Reviews, API, database)

Ask user, options:

1. Add new review
2. View all reviews
3. Search reviews
4. Exit

2. Add new review

Display menu: "content did you consume?" (PLASM)

1. Film (película)
2. Series (serie)

...

↓

User input

↓

Control error.

↓

Display prompt: "Enter the title of the film:"

↓

User input

↓

Control error.

3.Api search

```
function recieve: content_type, title  
    ↓  
Route to appropriate API handler:  
    - "film" → TMDb search_movie()  
    - "series" → TMDb search_tv()  
    - "anime" → Jikan search_anime()  
    - "manga" → Jikan search_manga()  
    - "book" → GoogleBooks search_books()  
    ↓
```

Make API call with retry logic:

↓

Parse API response:

Extract: id, title, year, poster_path, overview, rating

↓

Normalize to standard format JSON:

↓

Return: list of 3-5 normalized results

4.Select the result from the API and building the json

Display search results (prettified):

"Which one did you watch? (1-5) :"

↓

User input:

↓

Control error

↓

Return: selected_result (full object from search results)

Into api_manager.py

Input: content, api_result

↓

Make detailed API call:

GET /movie/551?api_key=TMDB_KEY (for films)

GET /anime/1?api_key=JIKAN_KEY (for anime)

...

↓

Also fetch additional data and receive complete TMDb JSON response

↓

Preprocess/normalize based on content type:

↓

Output normalized structure:

↓

Download poster locally:

↓

Return: complete normalized API data

5. Making the review

Receive: normalized API data

↓

Display nicely formatted:

↓

"Ready to review this? (y/n)"

↓

User confirms: "y"

↓

Continue to review phase

6. Adding the user review

Display menu:

1. ANALYTIC (Structured ratings)
2. SUBJECTIVE (Simple overall score)

↓

User input: (chooses ANALYTIC)

Return: review_type = "analytic"

Receive: content_type = "film", api_data

↓

Dimension weights for this content type:

- Direction (25% weight)
- Writing (25% weight)
- Acting (25% weight)
- Technical (25% weight)

↓

Control error:

↓

Calculate final_score AUTOMATICALLY:

↓

In the other case: Subjective review or fast review

- "What's your overall rating?"
- "How much did you enjoy it? (0-10) :"

↓

User input

↓

Control error

↓

Now, the review part: (This part is same for both ANALYTIC and SUBJECTIVE)

Input Review text (optional): "Write your review (press Enter to skip) :"

↓

Would rewatch, Watch context, status

7. Finishing the review and export to the database.

Receive 4 components:

1. api_data (from API preprocessing)
2. review_type ("analytic" or "subjective")
3. analytic_ratings (if review_type == "analytic")
or subjective_rating (if review_type == "subjective")
4. user_data (review text, would_rewatch, etc.)

↓

Merge into final structure:

↓

Validate merged structure:

↓

Check for duplicates in database:

↓

Return: validated final_data dict

8. Adding to the SQL database

Receive: final_data (complete review dict)

↓

Convert complex fields to JSON strings:

- genres: ["Adventure", "Drama", "Thriller"] → JSON string
- analytic_ratings: {...} → JSON string
- production_companies: [...] → JSON string

↓

Insert into SQLite database:

↓

Control error:

↓

Commit transaction

↓

Get inserted review ID from database

↓

Return: review_id (for confirmation)

Data flow:

User Input → API Search → User Selects → API Fetch → Data Preprocess → Display → Review Type → Ratings → Common Data → Merge → SQL Save → JSON Export → Complete

Architecture:

main.py (orchestration)

```
├── content_handler.py (user questions)
├── api_manager.py (API integration)
└── data_fusion.py (data merging)
└── storage/ (SQL & JSON saving)
```