

# PLASM PROJECT

## Workflow (functions and scripts)

	get_content_type()
main.py	get_title(content_type)
	search_content(content_type, title)
content_handler.py	select_from_results(results)
	fetch_complete_data(content_type, api_id)
api_manager.py	choose_review_type()
	get_analytic_ratings(content_type)
data_fusion.py	get_subjective_rating()
	get_common_review_data()
sql_handler.py	merge_all_data(api_data, review_type, ratings, common_data)

## 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)