Need to design an API for workflow and task management features. Assuming that there will be a predefined workflow already placed in DB for below mentioned workflow JSON definition using Liquibase yml script to ensure automatic provisioning during deployment stage.   
Workflow definition JSON:  
{

"name": "General Task Workflow",

"description": "Workflow for task creation, review, and completion.",

"category": "Predefined",

"trigger": {

"events": ["task\_assignment"]

},

"type": "Sequential",

"statuses": [

{"id": "1", "name": "Open"},

{"id": "2", "name": "Under Review"},

{"id": "3", "name": "Rework Required"},

{"id": "4", "name": "Completed"}

],

"transitions": [

{

"id": "1",

"name": "Submit for Review",

"from": "1",

"to": "2",

"action": "Submit for Review",

"actor": "assignee",

"validators": [

{"type": "required\_field", "fields": ["comment"]}

],

"postFunctions": [

{"type": "notify\_users", "users": ["assigner"], "message": "Task Submitted for Review: {task\_type}."},

{"type": "update\_task\_metadata", "action": "update\_task\_status", "status": "Under Review"}

]

},

{

"id": "2",

"name": "Mark as Completed",

"from": "2",

"to": "4",

"action": "Mark as Completed",

"actor": "assigner",

"validators": [

{"type": "required\_field", "fields": ["closure\_comment"], "optional": true}

],

"postFunctions": [

{"type": "notify\_users", "users": ["assignee"], "message": "Task {task\_id} has been marked as completed."},

{"type": "update\_task\_metadata", "action": "update\_task\_status", "status": "Completed"},

{"type": "commit\_updates", "action": "update\_system\_data"}

]

},

{

"id": "3",

"name": "Rework",

"from": "2",

"to": "3",

"action": "Rework",

"actor": "assigner",

"validators": [

{"type": "required\_field", "fields": ["rework\_comment"]}

],

"postFunctions": [

{"type": "notify\_users", "users": ["assignee"], "message": "Task {task\_id} requires rework: {rework\_comment}"},

{"type": "update\_task\_metadata", "action": "update\_task\_status", "status": "Rework Required"}

]

},

{

"id": "4",

"name": "Resubmit for Review",

"from": "3",

"to": "2",

"action": "Resubmit for Review",

"actor": "assignee",

"validators": [

{"type": "required\_field", "fields": ["comment"]}

],

"postFunctions": [

{"type": "notify\_users", "users": ["assigner"], "message": "Task Submitted for Review: {task\_type}\nA task has been resubmitted for your review."},

{"type": "update\_task\_metadata", "action": "update\_task\_status", "status": "Under Review"}

]

}

],

"dynamicAssignment": {

"rules": [

{

"condition": "past\_due\_date",

"parameters": {

"statuses": ["Open", "Under Review", "Rework Required"],

"time\_reference": "due\_date",

"grace\_period\_hours": 24

},

"action": "escalate",

"escalateTo": "assigner",

"postFunctions": [

{"type": "notify\_users", "users": ["assigner", "assignee"], "message": "Task {task\_id} has been escalated to {assigner} due to passing the due date."}

]

}

]

}

}  
  
This would be having statuses, transitions along with validators and post functions.  
Workflow triggering condition would be based on task\_assignment event.  
  
DB schema:  
-- To hold workflow definition

CREATE TABLE workflows (

id BIGINT GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY, -- Unique identifier for the workflow

ontology VARCHAR(50) NOT NULL DEFAULT 'GENERAL', -- Ontology type (e.g., "FRAUD\_INVESTIGATION"), defaults to 'GENERAL'

is\_system\_defined BOOLEAN NOT NULL DEFAULT FALSE, -- Indicates if the workflow is a core system workflow

definition JSONB NOT NULL, -- Storing workflow details (name, description, category, type, triggers, states, transitions, etc.)

created\_on BIGINT NOT NULL, -- Timestamp when the workflow was created

tenant\_id BIGINT NOT NULL, -- Tenant identifier for multi-tenancy

CONSTRAINT uk\_workflows\_name\_tenant UNIQUE ((definition->>'name'), tenant\_id), -- Ensures workflow names are unique per tenant

CONSTRAINT check\_category CHECK (definition->>'category' IN ('PREDEFINED', 'ADHOC')), -- Validates category in JSONB

CONSTRAINT check\_type CHECK (definition->>'type' IN ('SEQUENTIAL', 'PARALLEL', 'AUTO\_APPROVAL')), -- Validates type in JSONB

);

CREATE INDEX idx\_workflows\_name ON workflows USING GIN ((definition->>'name'));

CREATE INDEX idx\_workflows\_category ON workflows USING GIN ((definition->>'category'));

CREATE INDEX idx\_workflows\_type ON workflows USING GIN ((definition->>'type'));

### CREATE INDEX idx\_workflows\_trigger ON workflows USING GIN ((definition->'trigger'->'events')); Now lets say workflow definition is been placed successfully in workflows table. Task Management: Prerequisite

**Task workflow Definition**

* Task workflow definition should be in place that allows task transition as described in this document.
* Actors:
  + Task Creators - Users with appropriate roles - Supervisor, Analyst
  + Task Assignees -
    - Supervisors to other supervisors, Analyst & Jr Analyst
    - Analyst to another analysts and Jr Analyst
    - This framework should be configurable and applicable for any new role or custom role.
* Task Transitions States: Create -> Open -> Submit for Review -> Review Required/Completed
* Notifications for state transitions (Assigned, Submitted for Review, Requires Rework, Marked as Completed).

**Task Creation:**A screenshot of a computer

AI-generated content may be incorrect. **Lets say on UI I have similar such screen to create task.**

Create task action from above would open a create task panel. The panel will show and need information to create the task:

* + Task Title (Mandatory) - 100 chars
  + Task Type (Mandatory): It can hold multiple task types but assume as of now it will hold single task type i.e. General Task
    - If ‘General Task’ is selected – this means that an adhoc task is being assigned.
    - Description (Optional) - 500 chars
    - Assignee (Mandatory) – Single Select Dropdown - list of specified users that can work on the task
    - Due Date (Mandatory) – should be current or future date not past date.
    - Priority – Mandatory (High, Medium, Low)
    - Additional attachments (Optional)
      * User can attach any supporting document in the task if required (upto 100 MB per file)
      * System should check the uploaded format – Documents (pdf, doc, docx, xls, xlsx, csv, txt, odt), Images (png, jpg, jpeg), Videos (mp4, avi, mov, wmv), Audios (mp3, wav, acc). Any other format cannot be uploaded.
    - After filling in all the mandatory information, ‘Create’ option creates a new task and assigns to the assignee user.
    - A toast message is displayed on successful task creation – “The task has been created successfully!”
    - This task will also list under Task Management -> Task List screen
  + **Post Task Creation**
  + A unique task ID is assigned to the task
  + **Task ID format** – TK00000X (e.g. TK000001, TK000002 – this will allow creating upto 999999 tasks for a deployed tenant.
  + Task status is set as ‘Open’
  + Task will be listed under Task Management screen with basic details
  + Task assignee will receive a notification and can access the task from there or under task management list

### Task Execution

* The task assignee receives notifications on every task assignment.
* The notification will show:

***Task Assigned: <Task Type>***

*You have been assigned a task. Please proceed as necessary.*

* There is a dedicated Tasks module under Junior Analyst user login, where all the assigned tasks to him will be listed in a grid.
* Assignee can access a new task from the notification center or from the task management list
* The task management listing page displays all the tasks assigned with basic details – Task ID, Title, Task Type, Assigned On, Assigned By, Due On & Status.
* Clicking on a task opens the task homepage screen. A back navigation should be available to come back to the task list screen.
* Here the assignee can see all the details related to the task – Task ID, Title, Type, Description, Assigned On, Assigned By, Due on along with attachments (if any).
* Assignee can also view any comments added or attachments in the task.
* There is also a history timeline – that shows the timeline of the task status transitions over a period.
* Here, the assignee would also see an option to Proceed on the Task.
* On proceeding, the assignee would be able to see either the list of resources or the target, on which the task was created.
* If the task type was General, then the underlying resources and target profile would be in view only mode and no operations / data change can be performed (any CTA, action options should be hidden) from the screens.

**Submit for Review**

* + Back on the task homepage, user will see a CTA ‘Submit for Review’.
  + For Verify Target Profile task – user can simply complete the edits and come back to task home page and Submit the task.
  + User can also attach any supporting document in the task if required (upto 100 MB per file) - system should check the uploaded format – Documents (pdf, doc, docx, xls, xlsx, csv, txt, odt), Images (png, jpg, jpeg), Videos (mp4, avi, mov, wmv), Audios (mp3, wav, acc). Any other format cannot be uploaded.
  + Submitting for review will ask the user to mandatorily provide a comment and a confirmation message.
  + Once submitted for review, the system should send a notification to the Task Assigner and would be available on his task management list.
  + This action sets the Task Status to ‘Under Review’.
  + The task details screen also shows the progress made on the task visually (Task Assigned -> Submitted for review -> Next Actions) with the timeline.

### Task Review & Closure

* After task submission by the assignee, the assigner receives a notification in his login.

***Task Submitted for Review: Verify/Edit Analysis Summary***

A task has been submitted for your review. Please take necessary action.

* Assigner can also access the task from the task management list, where it is listed along with all other tasks initiated by him. The task list has basic details of the task – Task ID, Task Title, Type, Assigned On, Assigned To, Due Date, Status.
* Upon accessing a task, the assigner navigates to the Task homepage - a similar interface as for the assignee, with all the task details and comments made for review.
* Here, the assigner can take multiple actions on the task:
  + **Review:**
    - In this action, the assigner would be navigated to the screen; whichever were the part of the task to review the changes made by assignee.
    - A back navigation would be available to come back to the task home page.
  + **Mark as Completed:**
    - If the task execution results were satisfactory, the assigner can Mark the Task as Completed with an optional closure comment.
    - This action will change the task status to ‘Completed’.
    - The assignee will receive a notification of this status change.
* **Rework Required**
  + - If the task execution results were unsatisfactory, the assigner can Mark it with – Rework Required along with a reason and a comment (Mandatory).
    - This changes the task status to ‘Rework Required’
    - What would be the reason and should we include this in workflow json definition?

### Rework & Closure

* The assignee will receive a notification of this status change and can access the task and carry out required rework in the similar way as described before.
* Assignee submits the task for review again and the cycle continues until assigner marks it as completed at some point.

### Comments

* The assignee or assigner can make comments on the task at any point.
* When user starts adding a comment, the system should show options to Save or Cancel it, and user can take required action.
* All the comments made during the review process (reword required, marking as completed) also be added in comments thread.

### Attachments

* Any user (assignee or assigner) can add allowed files to the attachment section and it will be visible to assignee and assigner.
* The system should allow a viewer to be able to view the files content.

### General

* Completed tasks will still be available in view only mode to the users.
* All comments and attachments involved resources/target profiles etc can be accessed from a completed task.

DB Schema:  
-- To hold workflow definition

CREATE TABLE workflows (

id BIGINT GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY, -- Unique identifier for the workflow

ontology VARCHAR(50) NOT NULL DEFAULT 'GENERAL', -- Ontology type (e.g., "FRAUD\_INVESTIGATION"), defaults to 'GENERAL'

is\_system\_defined BOOLEAN NOT NULL DEFAULT FALSE, -- Indicates if the workflow is a core system workflow

definition JSONB NOT NULL, -- Storing workflow details (name, description, category, type, triggers, states, transitions, etc.)

created\_on BIGINT NOT NULL, -- Timestamp when the workflow was created

tenant\_id BIGINT NOT NULL, -- Tenant identifier for multi-tenancy

CONSTRAINT uk\_workflows\_name\_tenant UNIQUE ((definition->>'name'), tenant\_id), -- Ensures workflow names are unique per tenant

CONSTRAINT check\_category CHECK (definition->>'category' IN ('PREDEFINED', 'ADHOC')), -- Validates category in JSONB

CONSTRAINT check\_type CHECK (definition->>'type' IN ('SEQUENTIAL', 'PARALLEL', 'AUTO\_APPROVAL')), -- Validates type in JSONB

);

CREATE INDEX idx\_workflows\_name ON workflows USING GIN ((definition->>'name'));

CREATE INDEX idx\_workflows\_category ON workflows USING GIN ((definition->>'category'));

CREATE INDEX idx\_workflows\_type ON workflows USING GIN ((definition->>'type'));

CREATE INDEX idx\_workflows\_trigger ON workflows USING GIN ((definition->'trigger'->'events'));

-- To hold tracking of workflow instances

CREATE TABLE workflow\_instances (

id BIGINT GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY, -- Unique identifier for the workflow instance

workflow\_id BIGINT NOT NULL REFERENCES workflows(id), -- Foreign key to the associated workflow

task\_id BIGINT NOT NULL REFERENCES tasks(id), -- Foreign key to the associated task

current\_status VARCHAR(50) NOT NULL, -- Current status of the workflow instance (e.g., "OPEN")

current\_assignee BIGINT, -- Current assignee responsible for the next action

started\_on BIGINT NOT NULL, -- Timestamp when the workflow instance started

updated\_on BIGINT, -- Timestamp when the workflow instance updated

completed\_on BIGINT, -- Timestamp when the workflow instance completed (nullable)

tenant\_id BIGINT NOT NULL, -- Tenant identifier for multi-tenancy

CONSTRAINT fk\_workflow\_instances\_workflow FOREIGN KEY (workflow\_id) REFERENCES workflows(id),

CONSTRAINT fk\_workflow\_instances\_task FOREIGN KEY (task\_id) REFERENCES tasks(id)

);

-- To hold task details

CREATE TABLE tasks (

id BIGINT GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY, -- Unique identifier for the task

title VARCHAR(100) NOT NULL, -- Task title (max 100 chars)

description TEXT, -- Optional description (max 500 chars)

task\_type JSONB NOT NULL, -- e.g., ["Verify/Edit Resource Summary"]

assignee BIGINT NOT NULL, -- Assignee user id

assigner BIGINT NOT NULL, -- Assigner user id

due\_date TIMESTAMP NOT NULL, -- Due date for the task

priority VARCHAR(10) NOT NULL CHECK (priority IN ('HIGH', 'MEDIUM', 'LOW')), -- Priority: HIGH, MEDIUM, LOW

status VARCHAR(50) NOT NULL, -- e.g., Open, Under Review

created\_on BIGINT NOT NULL, -- Creation timestamp

updated\_on BIGINT NOT NULL, -- Last updated timestamp

tenant\_id BIGINT NOT NULL, -- Tenant identifier for multi-tenancy

CONSTRAINT uk\_task\_id\_tenant UNIQUE (id, tenant\_id) -- Ensures task IDs are unique per tenant

);

CREATE INDEX idx\_tasks\_task\_type ON tasks USING GIN (task\_type);

-- To hold task comments

CREATE TABLE task\_comments (

id BIGINT GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY, -- Unique identifier for the comment

task\_id BIGINT NOT NULL REFERENCES tasks(id), -- Foreign key to the associated task

content TEXT NOT NULL, -- Comment content

created\_by BIGINT NOT NULL, -- User who created the comment

created\_on BIGINT NOT NULL, -- Timestamp when the comment was created

CONSTRAINT fk\_task\_comments\_task FOREIGN KEY (task\_id) REFERENCES tasks(id) -- Ensures task\_id exists

CONSTRAINT fk\_task\_comments\_transition FOREIGN KEY (transition\_id) REFERENCES task\_history(id) -- To refer comment for specific transition

);

-- To hold attachments

CREATE TABLE task\_attachments (

id BIGINT GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY, -- Unique identifier for the attachment

task\_id BIGINT NOT NULL REFERENCES tasks(id), -- Foreign key to the associated task

file\_name VARCHAR(255) NOT NULL, -- Name of the file (e.g., "supporting\_doc.pdf")

object\_key VARCHAR(255) NOT NULL, -- Object key in the bucket (e.g., "tasks/TK000001/supporting\_doc.pdf")

file\_type VARCHAR(50) NOT NULL, -- File type (e.g., "pdf")

file\_size BIGINT NOT NULL, -- File size in bytes

uploaded\_by BIGINT NOT NULL, -- User who uploaded the file

uploaded\_on BIGINT NOT NULL, -- Timestamp when the file was uploaded

CONSTRAINT fk\_task\_attachments\_task FOREIGN KEY (task\_id) REFERENCES tasks(id), -- Ensures task\_id exists

CONSTRAINT check\_file\_size CHECK (file\_size <= 104857600) -- 100 MB limit

);

-- To hold task history timeline

CREATE TABLE task\_history (

id BIGINT GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY, -- Unique identifier for the history entry

task\_id BIGINT NOT NULL REFERENCES tasks(id), -- Foreign key to the associated task

previous\_status VARCHAR(50) NOT NULL, -- Previous status (e.g., "Open")

new\_status VARCHAR(50) NOT NULL, -- New status (e.g., "Under Review")

created\_by BIGINT NOT NULL, -- User who performed the transition

created\_on BIGINT NOT NULL, -- Timestamp of the transition

CONSTRAINT fk\_task\_history\_task FOREIGN KEY (task\_id) REFERENCES tasks(id) -- Ensures task\_id exists

);