Software Design Specification

for

Research Grant Management System

Part II

Group No.: 3

Suraj A/L Prakash	**243UC247CL**
Law Jun Feng	**243UC247CP**
Lim Jian Feng	**243UC247BZ**

Date: 25/1/2026

Revisions

Version	Primary Author(s)	Description of Version	Date Completed
SRS in Part 1(as Ver 1.0) SDS in Part 2(as Ver 2.0.X) *System Documentation in Part 3 (as Ver 3.0) Draft Type and Number	Full Name	Information about the revision. This table does not need to be filled in whenever a document is touched, only when the version is being upgraded.	00/00/00

1 System Overview

1.1 Description

The **Research Grant Management System (RGMS)** is a centralized platform designed to manage the full lifecycle of university research grants. The system facilitates the transition of research ideas from initial proposal through to funded project execution and departmental oversight. It operates on a **three-tier architecture**, separating user interactions from core business logic and database persistence.

To provide a clear understanding of the system's requirements, its major processes are categorized into the following functional groups:

1. Proposal Management & Versioning

This group handles the entry and iteration of research data.

- **Submission**: Researchers upload proposals and associated metadata through a standardized web interface.
- **Version Control**: The system automatically detects existing titles and increments version numbers (e.g., v1.0 to v1.1) to ensure data integrity and prevent duplicates.
- **Notifications**: Automated alerts inform users of deadlines, submission confirmations, and pending reviews.

2. Peer Review & Assessment

This process governs the merit-based evaluation of submissions.

- **Evaluation**: Reviewers provide quantitative scores and qualitative feedback on assigned proposals.
- **Document Review**: Authorized Reviewers can access and assess full research PDF documents directly within the system.
- **Result Hand-off**: Once completed, assessments are "locked" and dispatched to the HOD for final decision-making.

3. Financial Oversight & Governance

This group constitutes the administrative control center managed by the Head of Department (HOD).

- **Decision Logic**: The HOD reviews proposal merits and reviewer feedback to formally approve or reject grant applications.
- **Grant Allocation**: Approved proposals are initialized as active grants, with funds strictly validated against available department reserves.
- **Threshold Monitoring**: The system continuously tracks budget health (Total Allocated vs. Total Spent).
- **Critical Alerts**: Automated visual alerts (Red) are triggered if a project's budget usage exceeds the **90% safety threshold**.

4. Strategic Monitoring & Analytics

This process focuses on departmental Key Performance Indicators (KPIs) and project progress.

- **Progress Tracking**: The HOD monitors project timelines and milestone completion percentages based on researcher reports.
- **Department Analytics**: An aggregation engine pulls data across all grants to calculate overall performance metrics and "burn rates".
- **Reporting**: Administrative tools allow for the export of high-level analytics into PDF or Excel formats for external stakeholders.

1.2 Actors

Actor	Use Cases
Researcher	● Upload research proposals and reports ● Document Version Control ● Track grant status & budget usage(track own budget) ● Receive notifications (deadlines, approvals)
Reviewer	● Review submitted proposals ● View researcher documents ● Give evaluation / feedback(visible for both HOD & Researcher) ● Receive notifications for pending reviews(from researcher)
Head of Department(HOD)	● Approve / reject grant proposals ● Monitor research project progress ● Grant Allocation & Budget Tracking ● Monitor Research Dashboard (KPIs)

1.3 Assumptions and Dependencies

1.3.1 Assumptions

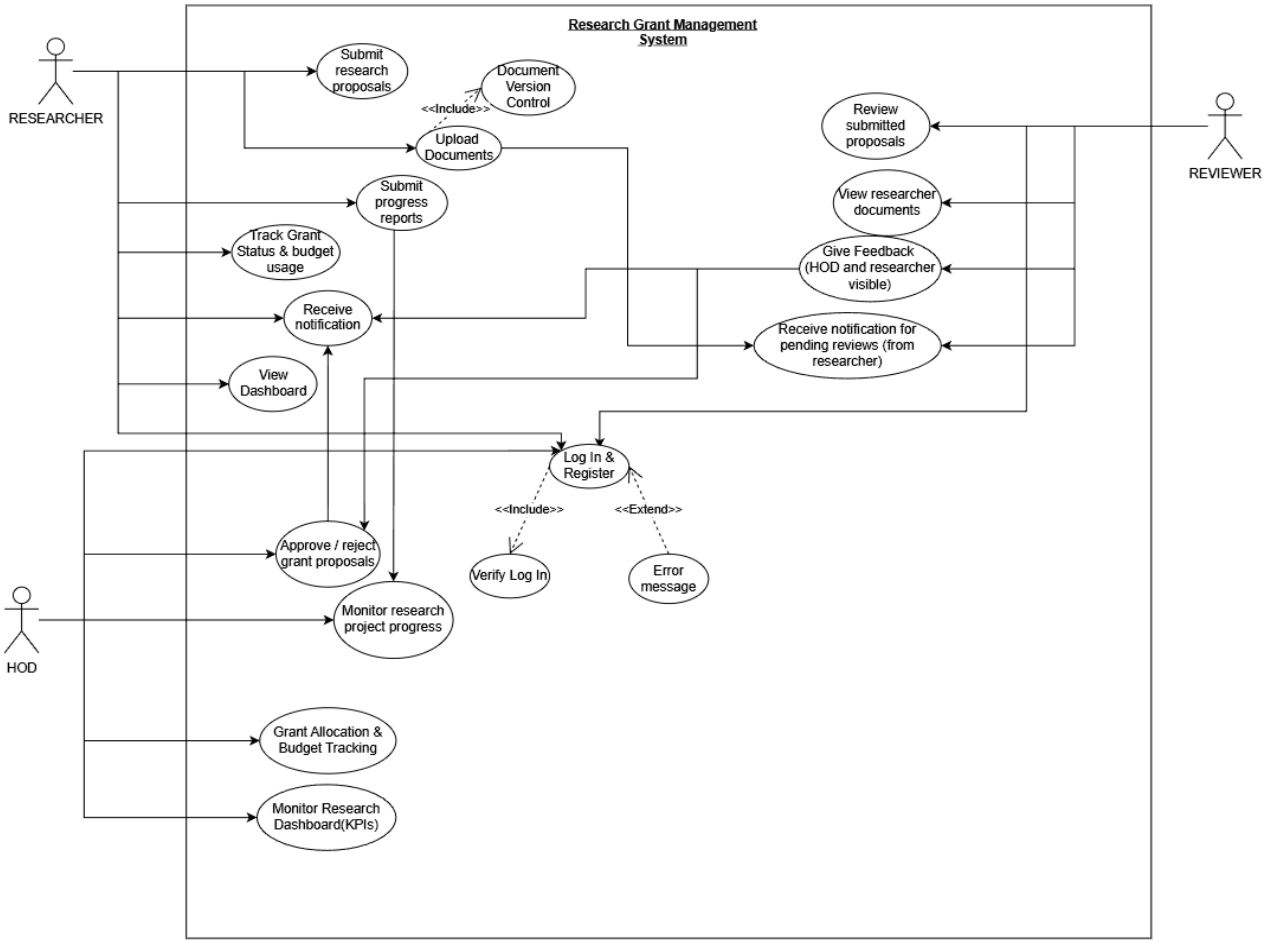
1. **Organizational Scale**: The system is designed for a university department or small faculty. It assumes a scale of approximately 10–50 active researchers and 1 HOD user, ensuring the **SQLite** database remains performant.

2. **Centralized Data Repository**: It is assumed that all historical research data necessary for current metrics has been migrated into the system's database to ensure the **Dashboard Manager** provides accurate analytics.
3. **HOD Financial Authority**: The system design assumes that the Head of Department (HOD) has the sole administrative authority to approve grant allocations and that deductions from the department budget are legally binding within the university's financial framework.
4. **Network Accessibility**: It is assumed that users have stable intranet or internet access to interact with the Django web interface, as the system does not support an offline data-entry mode.
5. **Role-Based Access Control (RBAC)**: The architecture assumes a strict hierarchy where "HOD" roles have full oversight of financial data, while "Researcher" roles are limited to their own proposal and progress report submissions.

1.3.2 Dependencies

1. **Python & Django Ecosystem**: The project is entirely dependent on the **Python 3.x** environment and the **Django 4.x** framework. Any significant version changes during deployment may affect the ORM logic or URL routing.
2. **Data Science Stack (Matplotlib & Seaborn)**: The advanced reporting and PDF export features depend on the installation of the **Matplotlib** and **Seaborn** libraries in the server's virtual environment to render high-resolution charts.
3. **Relational Database Engine**: While the system currently uses **SQLite** for development and small-scale use, the project depends on a SQL-compliant engine. Transitioning to a large-scale production environment would require a dependency shift to **PostgreSQL**.
4. **PDF Rendering Engine**: The "Export to PDF" functionality for administrative audits depends on third-party libraries such as **xhtml2pdf** or **ReportLab** to translate HTML templates into document formats.
5. **Browser Compatibility**: The user interface depends on modern web browsers (e.g., Chrome, Firefox, Safari) that support HTML5 and ES6 JavaScript standards for the d*ashboard visualizations to render correctly.*

1.4 Use Case Diagram

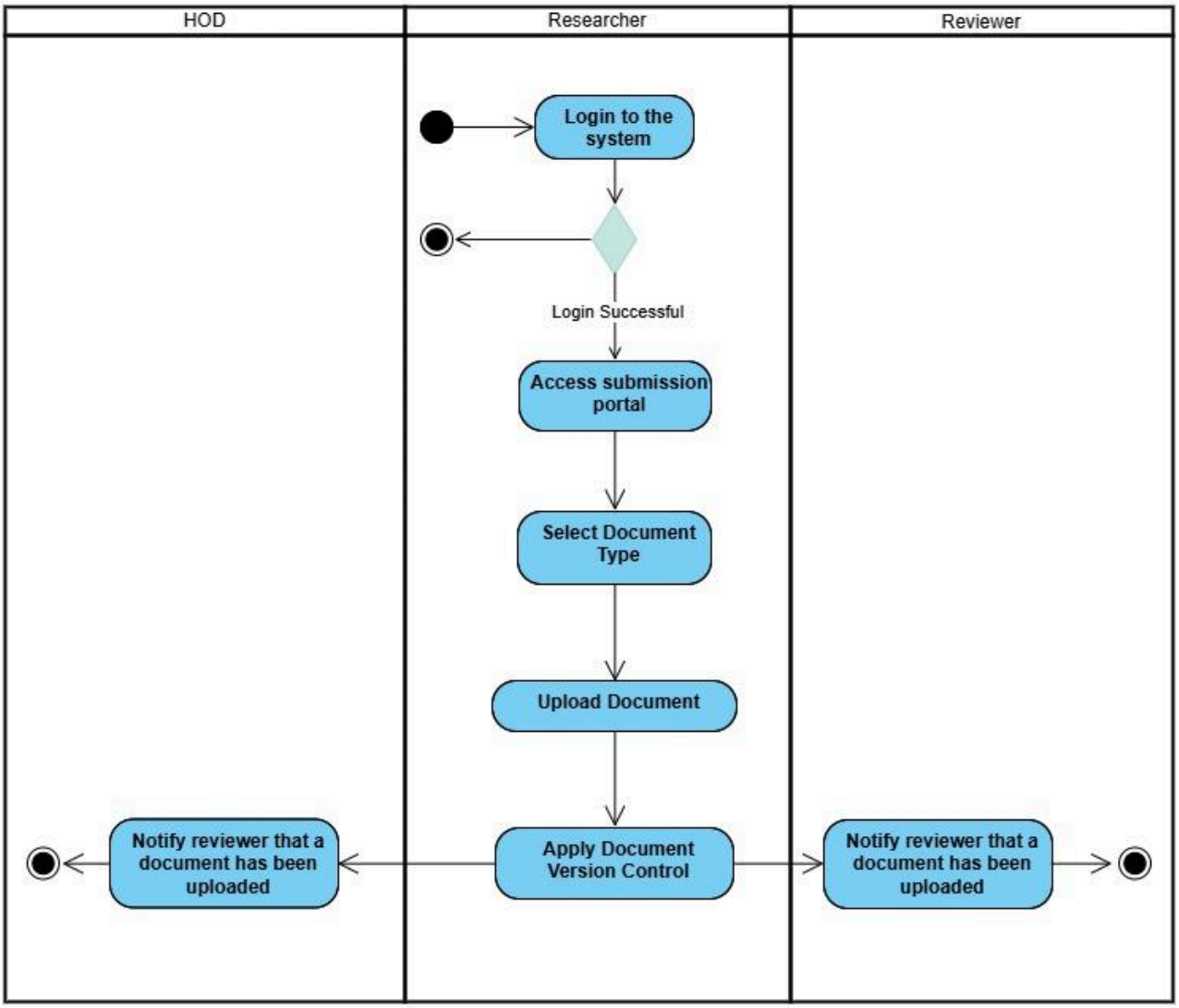


2 Activity Diagrams

2.1 Activity Diagrams

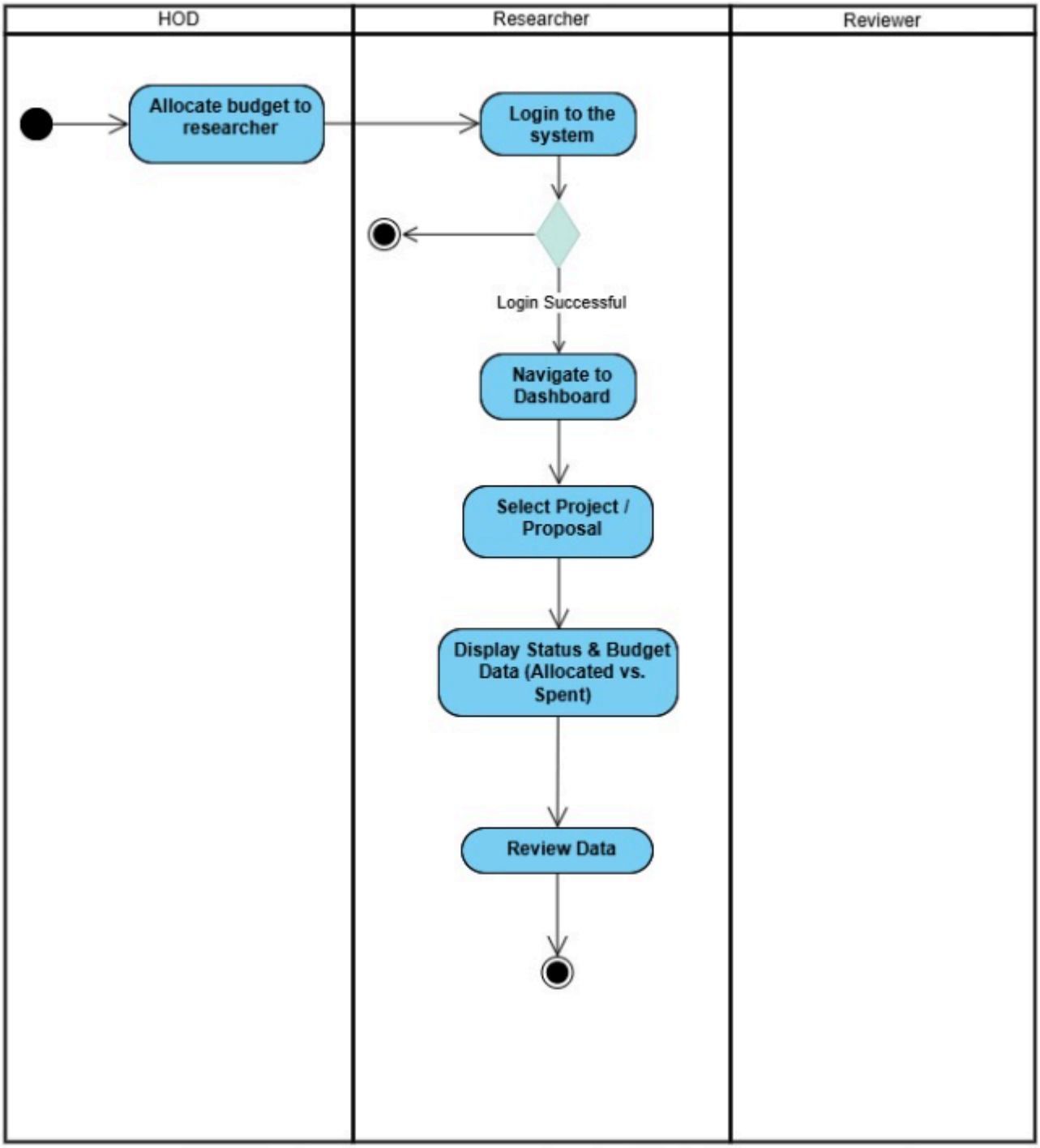
2.1.1 Use Case 1:Upload Research Proposals and Reports

This critical use case covers the submission of all required documents to the RGMS. The Researcher uses an online portal to submit research proposals to initiate a grant request and, subsequently, to upload progress reports for active projects. Successful uploads trigger the system's Document Version Control and may trigger notifications to Reviewers.



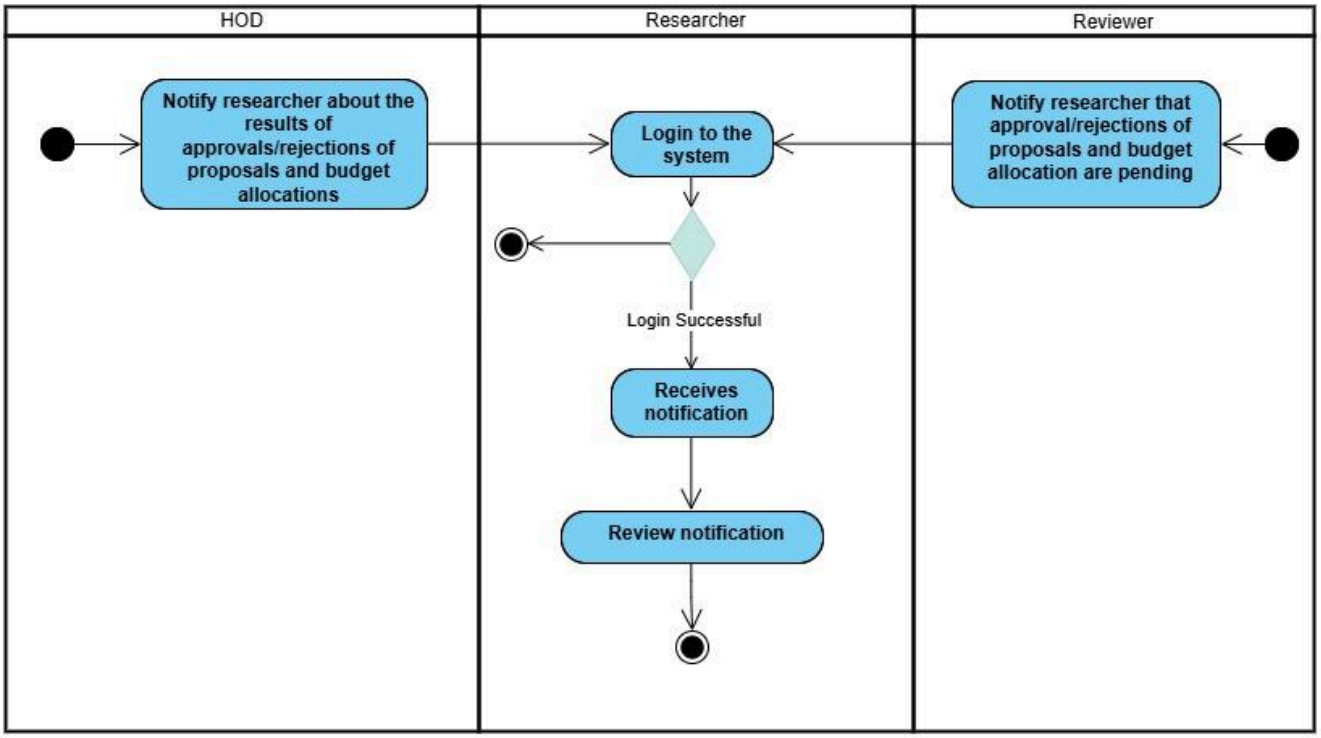
2.1.2 Use Case 2: Track Grant Status & Budget Usage

This use case enables the Researcher to actively monitor the status of their submitted proposals throughout the review cycle and review the financial health of any active project. They access a centralized Dashboard to view funding status and project progress (KPIs), specifically tracking their own allocated budget and expenditure.



2.1.3 Use Case 3: Receive Notifications (Deadlines, Approvals)

This use case details the system's ability to automate alerts to the Researcher. The Researcher receives notifications for critical workflow events, such as impending deadlines (e.g., for progress reports) and the outcome of proposal reviews (approvals or rejections). Notifications ensure the Researcher stays informed and prevents delays caused by manual correspondence.



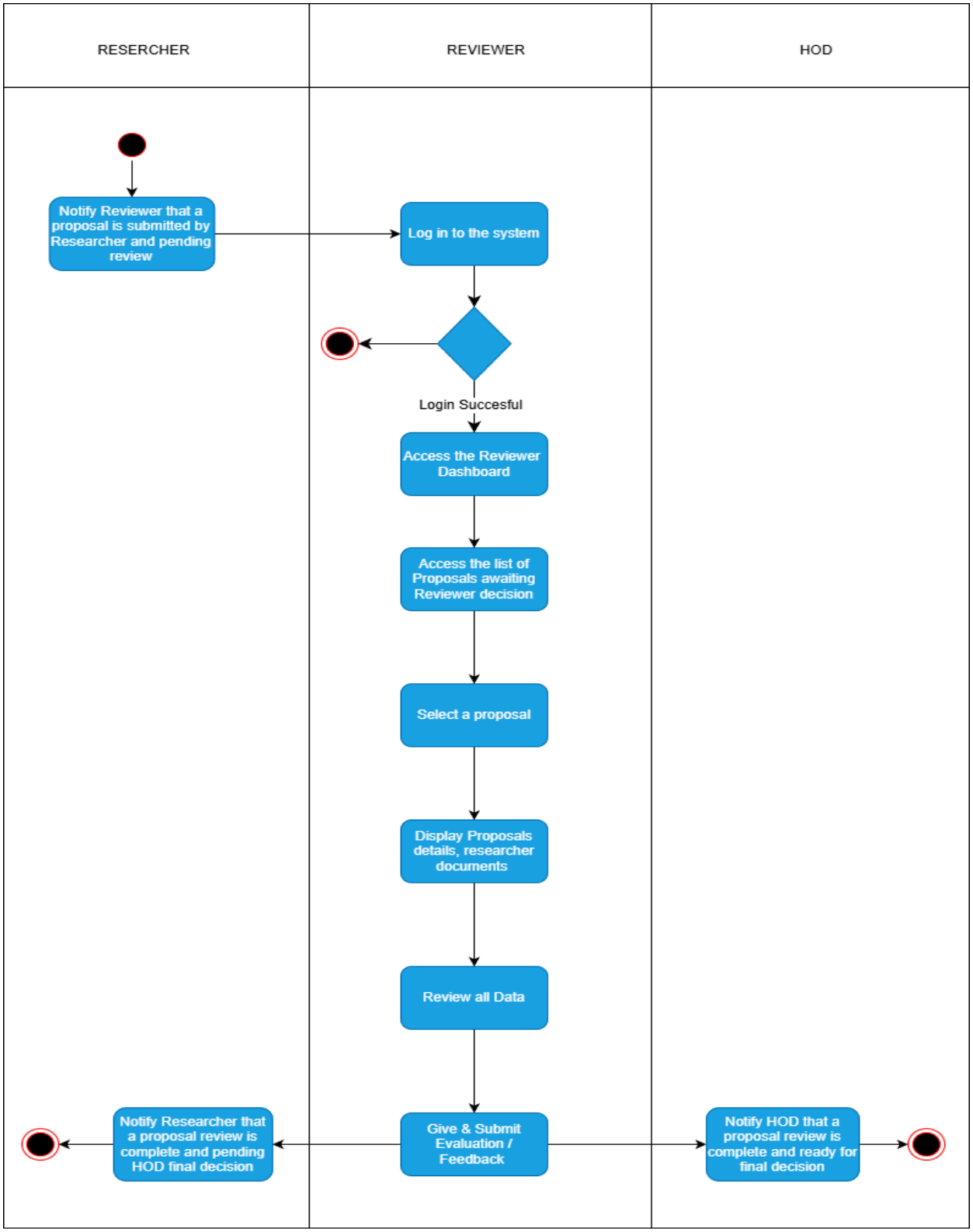
2.1.4 Use Case 4

Review & Feedback Researcher documents / proposals

This use case defines the end-to-end evaluation process. The **Authorized Reviewer** begins by having the system display a list of their assigned proposals, along with key metadata like the **Proposal ID**, **title**, and **submission date**. The system retrieves and displays the **full research proposal document** and any associated **researcher documents** from storage for the Reviewer's assessment. After reviewing the material, the Reviewer enters the necessary **evaluation scores** (e.g., numerical ratings for methodology) and provides **detailed textual feedback/comments**. The Reviewer concludes their input by submitting a **final review.**Upon submission, the system stores the complete **Review Record**, including all scores, feedback in the database, and updates the **Proposal Status** to 'Review Complete'.

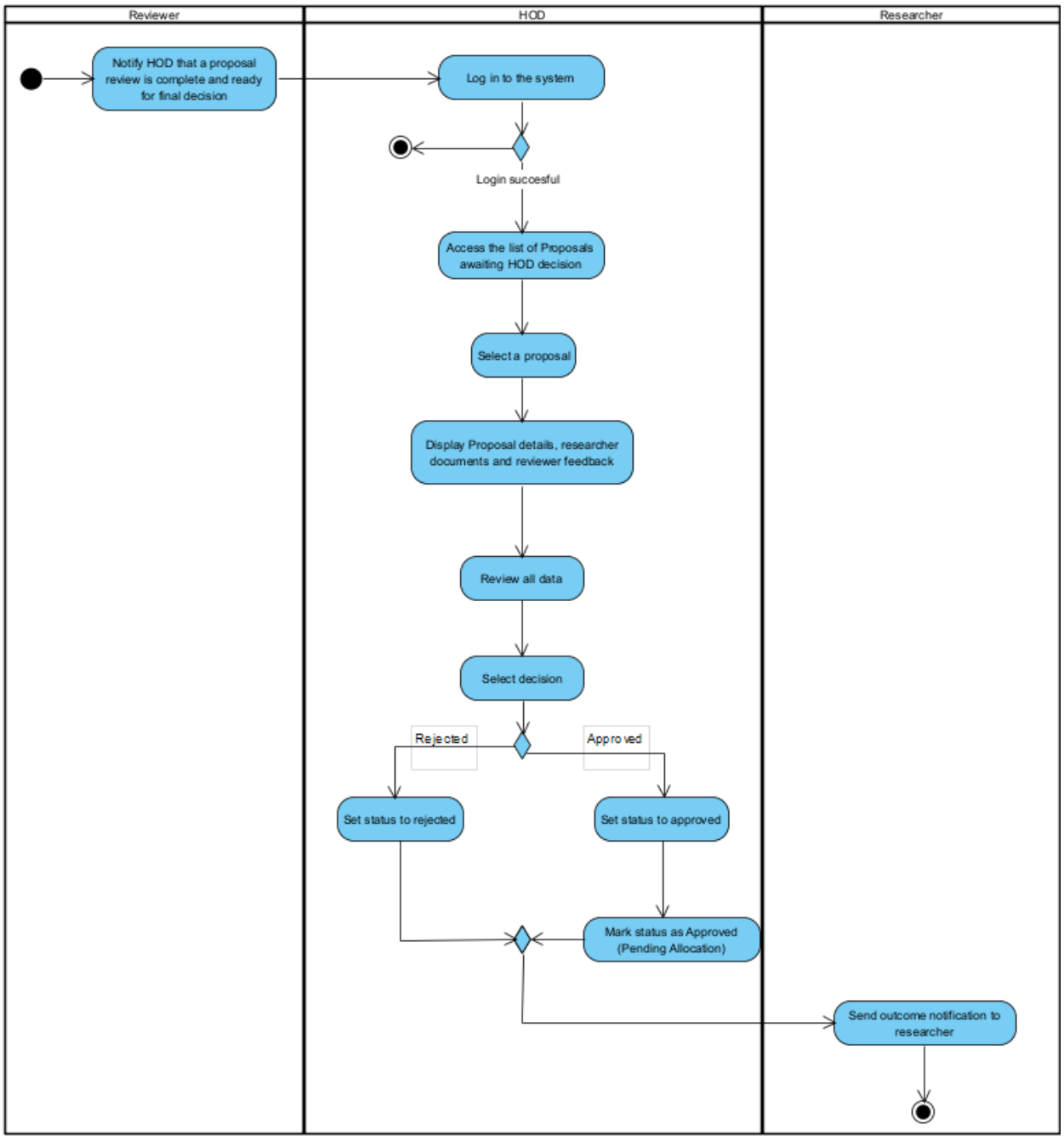
Receive notification for pending reviews(from researcher)

This use case ensures the Reviewer is promptly informed when a new proposal or revised version is submitted and awaiting evaluation. Triggered by the Researcher's submission, the system retrieves the **Reviewer's contact information** (e.g., email address) and the **Proposal's metadata** (ID, title, and calculated review deadline) from the database. The system then sends a **Notification** (via email or in-system alert) to the Reviewer, displaying the **Proposal Title**, **ID**, **review deadline**, and a **call-to-action link** to begin the review. The system logs a **Notification Record** in storage, noting the Reviewer ID, Proposal ID, and the **timestamp of the notification sent**. To access the proposal, the Reviewer must interact with the notification by clicking the link and then successfully **logging into the system**.


RESERCHER
REVIEWER
HOD
Notify Reviewer that a proposal is submitted by Researcher and pending review
Log in to the system
Login Succesful
Access the Reviewer Dashboard
Access the list of Proposals awaiting Reviewer decision
Select a proposal
Display Proposals details, researcher documents
Review all Data
Give & Submit Evaluation / Feedback
Notify Researcher that a proposal review is complete and pending HOD final decision
Notify HOD that a proposal review is complete and ready for final decision

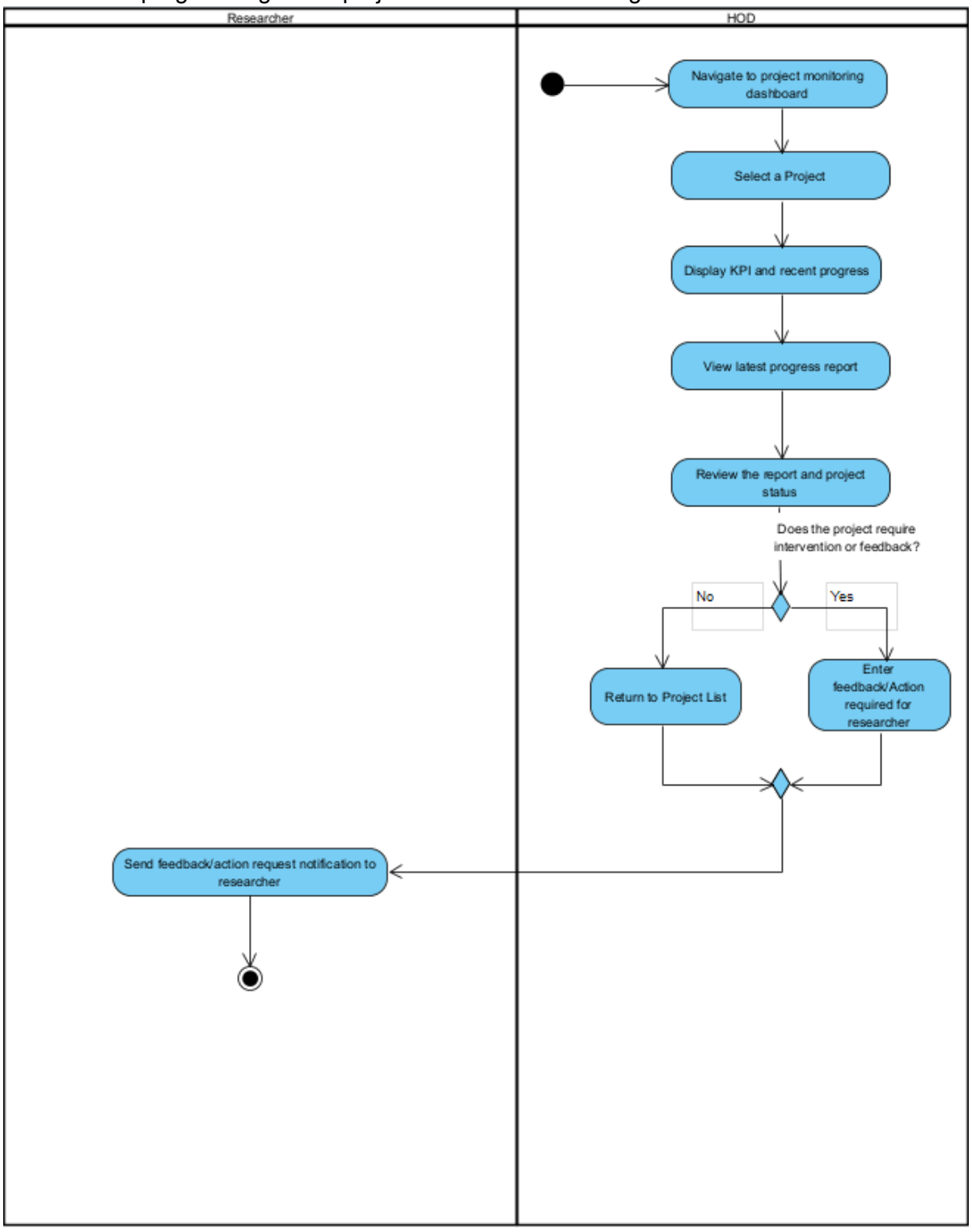
2.1.5 Use Case 5: **Approve/Reject Grant Proposals**

The HOD initiates this process by retrieving a list of pending grant applications from the database. The system displays the applicant's profile, the full research proposal, and the requested budget details for the HOD to review. After evaluating the merit of the application, the HOD enters a final decision (Approve or Reject) along with specific review comments. Finally, the system stores the updated proposal status and the HOD's feedback into the database to conclude the transaction.



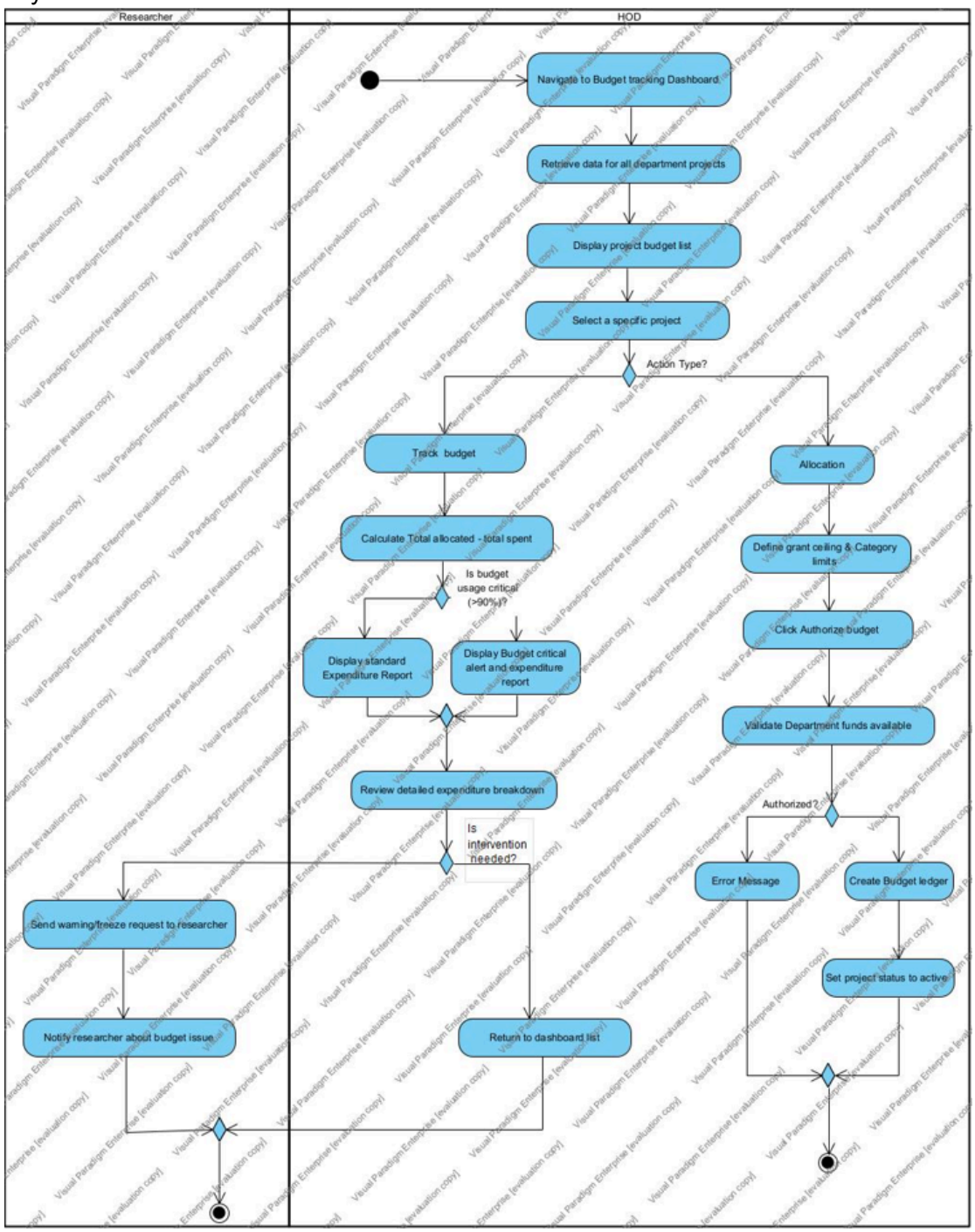
2.1.6 Use Case 6: **Monitor Research Project Progress**

To track performance, the HOD enters a search for a specific project, prompting the system to retrieve the relevant milestones and submitted deliverables. The system displays the project's current progress timeline, percentage of completion, and any adherence issues. Based on this view, the HOD enters a status flag (e.g., "On Track" or "Needs Intervention"), which the system then stores as a progress log in the project file for future auditing.



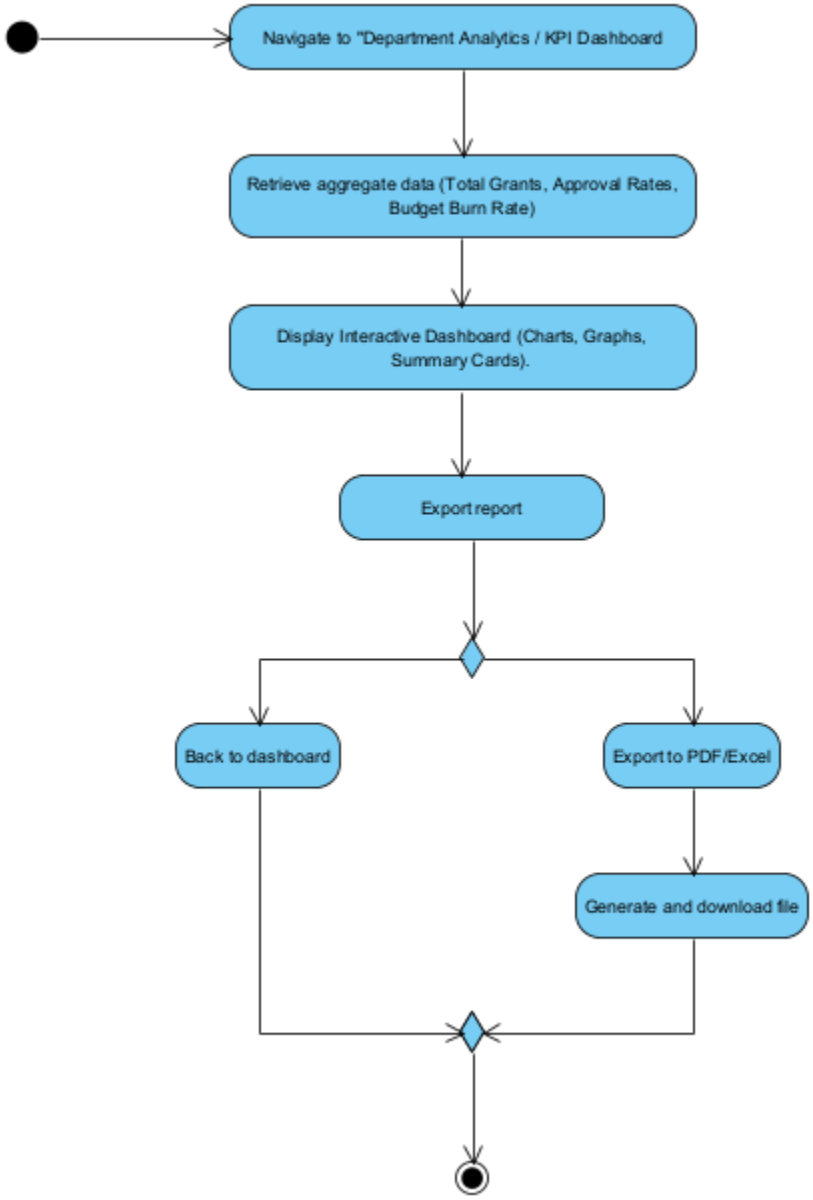
2.1.7 Use Case 7: **Grant Allocation & Budget Tracking**

This use case describes how the **Head of Department (HOD)** monitors the financial health of active research projects. The system automatically calculates budget usage (Total Allocated vs. Total Spent) and provides visual indicators for the HOD. The process focuses on identifying projects with critical budget usage (e.g., >90%) or those exceeding limits, allowing the HOD to review detailed expenditure reports and intervene by sending warnings or freezing spending if necessary.



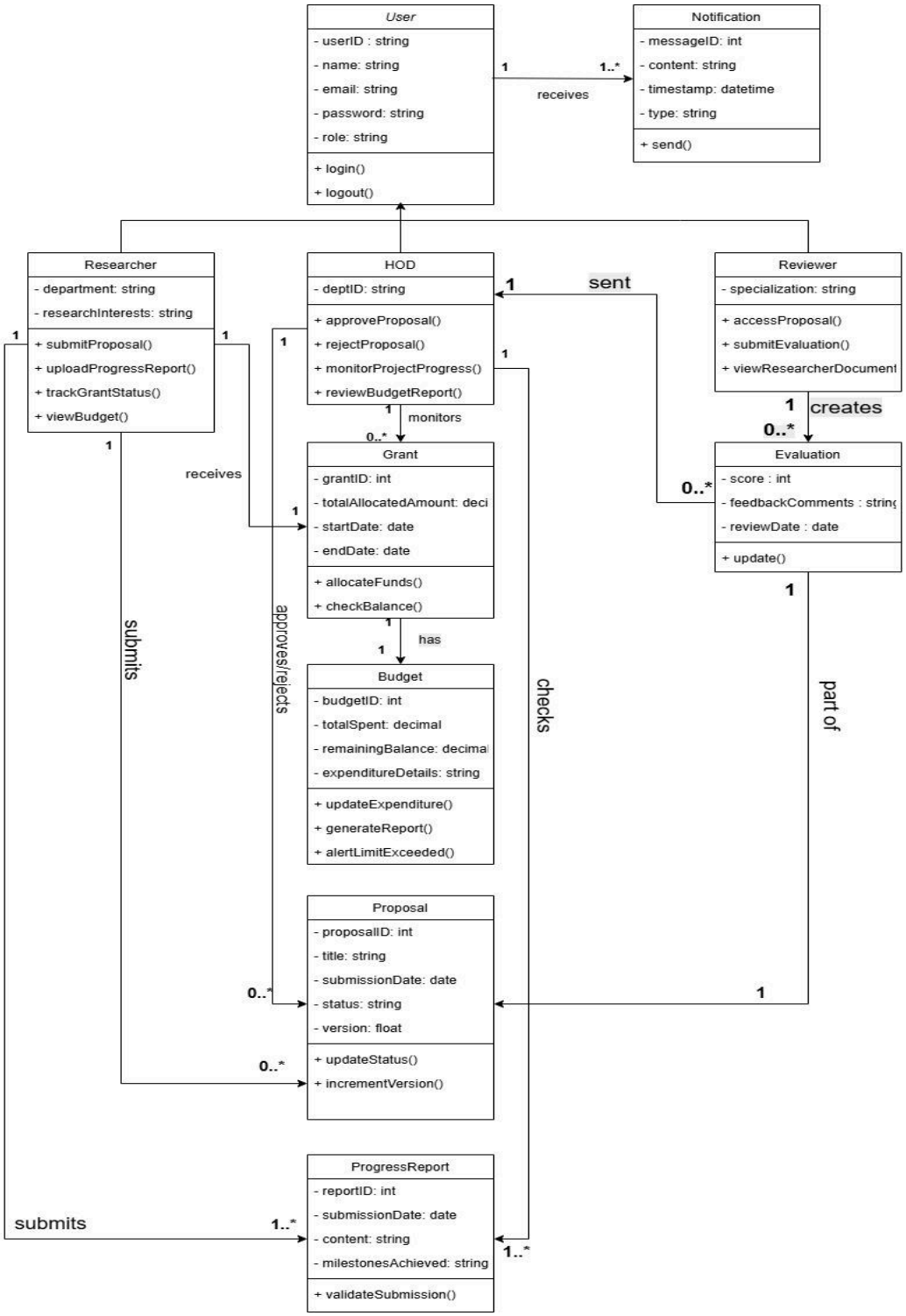
2.1.8 Use Case 8: **Monitor Research Dashboard (KPIs)**

The HOD analyzes departmental trends by entering specific filter criteria, such as a date range (e.g., 2024-2025) or faculty name. The system retrieves aggregated data on all proposals and funds to calculate performance metrics. It then displays visual Key Performance Indicators (KPIs), such as success rate charts and total funding graphs, while storing the HOD's query parameters and session activity for usage tracking.



3 Data Design

3.1 Design Class Diagram



3.2 Data Dictionary

User Entity

Field Name	Data Type	Length	PK/FK	Description
userID	Integer	10	PK	Unique identifier for the user.
name	String	100	-	The title of the research project.
email	String	100	-	Academic or professional email address.
password	String	255	-	Encrypted login credentials
role	String	20	-	Identifies if the user is a Researcher, Reviewer, or HOD.

Proposal Entity

Field Name	Data Type	Length	PK/FK	Description
proposalID	Integer	-	PK	Unique identifier for the proposal.
title	String	255		The title of the research project.
submissionDate	Date			Date the proposal was submitted.
status	String	50		Tracks state (e.g., Pending, Under Review, Approved)
version	Float			Tracks document iterations for version control.

Grant Entity

Field Name	Data Type	Length	PK/FK	Description
grantID	Integer	-	PK	Unique identifier for the grant
totalAllocatedAmount	Decimal	12,2		The total funding amount authorized.
startDate	Date			Start date of the funding period.
endDate	Date	50		End date of the funding period.

proposalID	Integer		FK	Links the grant back to the original proposal.

Budget Entity

Field Name	Data Type	Length	PK/FK	Description
budgetID	Integer	-	PK	Unique identifier for the budget.
totalSpent	Decimal	12,2	-	The cumulative amount spent so far.
remainingBalance	Decimal	12,2	-	Calculated balance (Allocated - Spent).
expenditureDetails	String	1000	-	Text log of project expenses.
grantID	Integer	-	FK	Links the budget to the specific Grant

ProgressReport Entity

Field Name	Data Type	Length	PK/FK	Description
reportID	Integer	-	PK	Unique identifier for the report.
submissionDate	Decimal	-	-	The date the report was filed.
content	Decimal	1000	-	Summary of recent project progress.
milestonesAchieved	String	500	-	Specific goals reached during this period.
proposalID	Integer	-	FK	Links the report to the parent proposal.

Researcher Entity

Field Name	Data Type	Length	PK/FK	Description
userID	Integer	10	PK/FK	Inherited primary key from User.
department	string	100	-	The faculty or department the researcher belongs to
researchinterests	string	255	-	Specific fields of study for the researcher.

Reviewer Entity

Field Name	Data Type	Length	PK/FK	Description
userID	Integer	10	PK/FK	Inherited primary key from User.
specialization	string	100	-	The reviewer's area of expert knowledge.

HOD Entity

Field Name	Data Type	Length	PK/FK	Description
userID	Integer	10	PK/FK	Inherited primary key from User.
deptID	string	100	-	Unique identifier for the department they lead.

Evaluation Entity

Field Name	Data Type	Length	PK/FK	Description
score	Integer	100	PK	Numerical rating assigned by the Reviewer.
feedbackComments	string	1000	-	Qualitative remarks regarding the proposal.
reviewDate	date	1000	-	The date the evaluation was submitted.

Notification Entity

Field Name	Data Type	Length	PK/FK	Description
messageID	Integer	-	PK	Unique identifier for the notification.
content	string	500	-	The text body of the notification message.
timestamp	datetime	-	-	The exact time the notification was sent.
type	string	20	-	Category of alert (e.g., Email, System Alert).

3.3 Data Structures

3.3.1 Data Structure 1: User

Attribute	Data Type	Description
id	BigInt	Unique auto-generated identifier for the user.
username	Varchar	The unique login name chosen by the user.
email	Varchar	The registered email address used for contact.
password	Varchar	The hashed security string used for authentication.
role	Varchar	Defines access level ('Researcher', 'HOD', or 'Reviewer').

3.3.2 Data Structure 2: Researcher

Attribute	Data Type	Description
user_ptr_id	BigInt	Foreign key linking to the base User table.

department	Varchar	The academic faculty or department the researcher belongs to.
research_interest	Varchar	The specific fields of study or topics the researcher focuses on.

3.3.3 Data Structure 3: HOD

Attribute	Data Type	Description
user_ptr_id	BigInt	Foreign key linking to the base User table.
department	Varchar	The specific department managed by the HOD.
total_department_budget	Decimal	The master budget pool available for the department (RM).

3.3.4 Data Structure 4: Reviewer

Attribute	Data Type	Description
user_ptr_id	BigInt	Foreign key linking to the base User table.

expertise_area	Varchar	The specific field of study the reviewer specializes in.

3.3.5 Data Structure 5: Proposal

Attribute	Data Type	Description
proposalID	BigInt	Unique identifier for the proposal submission.
title	Varchar	The official title of the proposed research project.
submissionDate	DateTime	Timestamp of when the proposal was submitted.
status	Varchar	Current state ('Draft', 'Pending', 'Approved', 'Rejected').
version	Float	Tracks revision history (e.g., 1.0, 1.1) to avoid duplicates.
requested_amount	Decimal	Total grant budget requested by the researcher (RM).

pdf_file	Varchar	File path to the uploaded PDF proposal document.
researcher_id	BigInt	Foreign key linking to the Researcher who owns this proposal.

3.3.6 Data Structure 6: Grant

Attribute	Data Type	Description
grantID	BigInt	Unique identifier for the approved grant.
totalAllocatedAmount	Decimal	The final approved budget amount allocated to the project (RM).
status	Varchar	Current condition ('Active', 'Completed', 'Needs Intervention').
startDate	Date	The date the grant funding becomes available.
endDate	Date	The deadline for the research project completion.

proposal_id	BigInt	Foreign key linking to the original approved Proposal.

3.3.7 Data Structure 7: Budget

Attribute	Data Type	Description
budgetID	BigInt	Unique identifier for the budget tracking sheet.
totalSpent	Decimal	Cumulative sum of all approved expenditures (RM).
grant_id	BigInt	Foreign key linking to the associated Grant.

3.3.8 Data Structure 8: ExpendituresDetails

Attribute	Data Type	Description
expID	BigInt	Unique identifier for the financial transaction.
date	Date	The exact date the expense was incurred.
itemDescription	Varchar	Description of the item or service purchased.

amount	Decimal	The cost of the specific transaction (RM).
receiptFile	Varchar	File path to the uploaded proof of purchase or invoice.
budget_id	BigInt	Foreign key linking to the parent Budget sheet.

3.3.9 Data Structure 9: ProgressReport

Attribute	Data Type	Description
reportID	BigInt	Unique identifier for the report.
submissionDate	DateTime	Timestamp of when the report was filed.
content	Text	Detailed narrative description of progress or HOD feedback.
milestonesAchieved	Text	Short summary or bullet points of key tasks completed.
proposal_id	BigInt	Foreign key linking to the associated Proposal/Grant.

3.3.10 Data Structure 10: Evaluation

Attribute	Data Type	Description
evaluationID	BigInt	Unique identifier for the evaluation record.
score	Integer	Numerical rating assigned by the reviewer (e.g., 0-100).
feedback	Text	Detailed comments justifying the score and decision.
dateEvaluated	DateTime	Timestamp of when the review was completed.
reviewer_id	BigInt	Foreign key linking to the Reviewer.
proposal_id	BigInt	Foreign key linking to the Proposal being evaluated.

…

4 Behavioral Modeling

4.1 Sequence Diagrams

4.1.1 Researcher Use Case 1

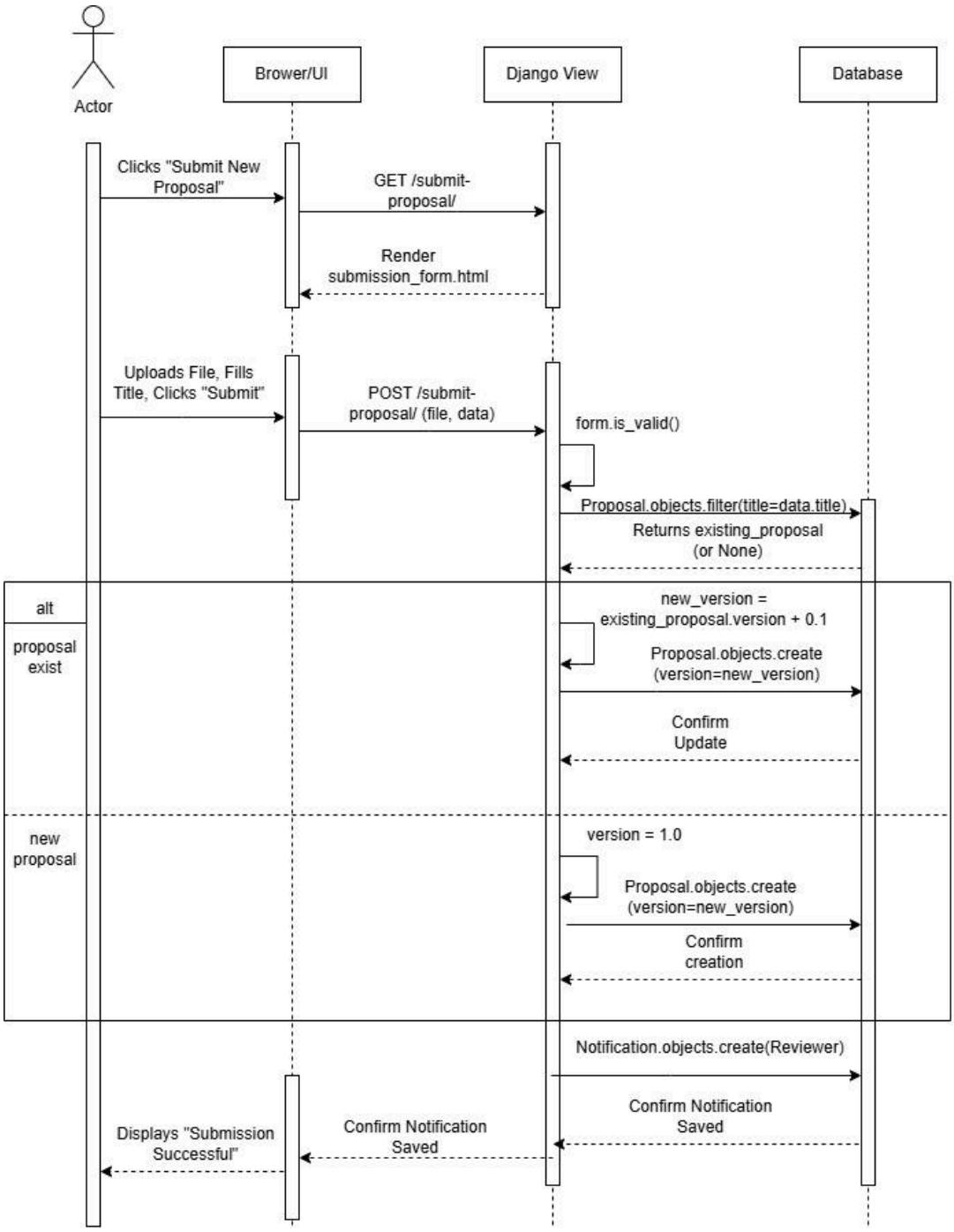
This sequence diagram illustrates the process of a **Researche**r submitting a new proposal or updating an existing one. The interaction begins when the **Actor** submits a form containing the proposal file and metadata via the **Browser/U**I.

1. The Django View receives the **POST** request and first validates the form data.
2. To enforce **Document Version Control** (as defined in the functional requirements), the **View** queries the **Database** to check if a proposal with the same title already exists.

ALT:

- **If it exists:** The system retrieves the current version number and increments it (e.g., from 1.0 to 1.1) to create a new version record.
- **If it is new:** The system creates a new entry with version 1.0.

3. Once the proposal is saved, the **View** triggers a second **Database** operation to create a **Notification** for the **Reviewer**.
4. Finally, the system redirects the user back to the dashboard with a success message.

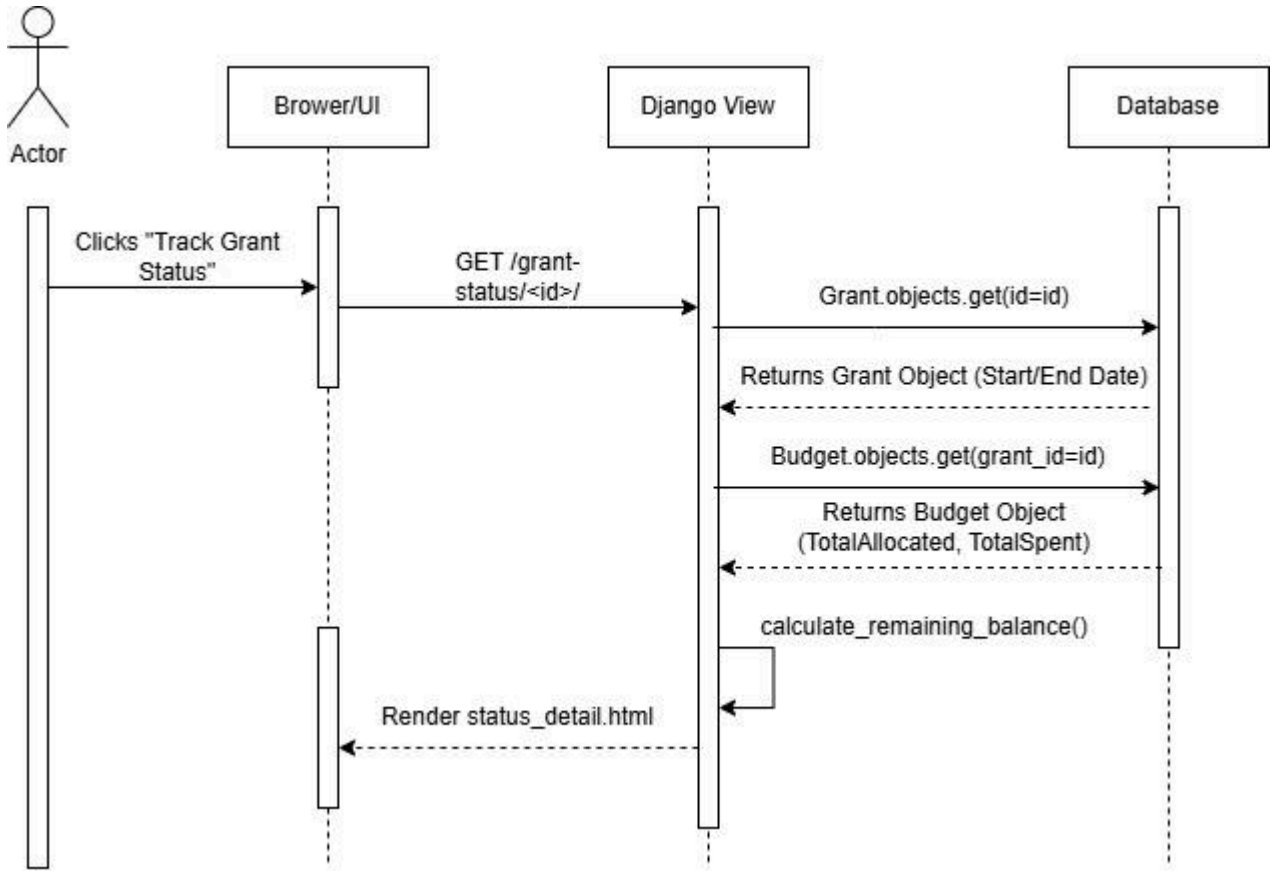


4.1.2 Use Case 2

This diagram details how the system retrieves and calculates financial data for a specific grant. The **Actor** selects a grant from their list to view detailed metrics.

1. The **Browser/UI** sends a **GET** request to the **Django View** with the specific grant_id.
2. The View performs two distinct **Database** lookups:
    - First, it retrieves the **Grant** entity to get the start/end dates and total allocation.
    - Second, it retrieves the associated Budget entity.

3. The **View** logic then performs a calculation to determine the remainingBalance (Total Allocated minus Total Spent) to ensure the data displayed is up-to-date.
4. This calculated data is passed to the template, and the **Browser/UI** renders the status bar and budget utilization graphs for the **Researcher**.

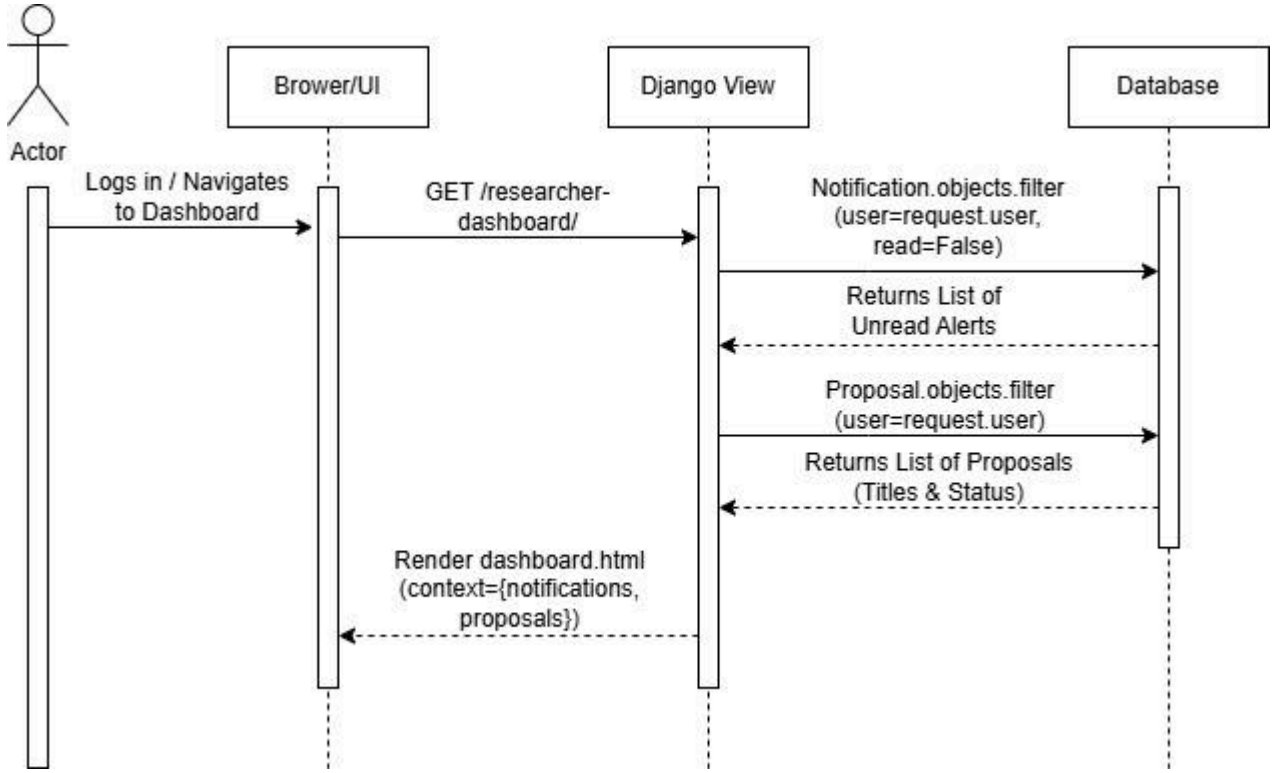


4.1.3 Use Case 3

This diagram explains the initialization of the **Researcher's** main landing page. This use case aggregates multiple data points to provide a "single pane of glass" view for the user.

1. When the **Actor** logs in or navigates to the home page, the **Browser/UI** sends a request to the researcher_dashboard **View**.
2. The **Django View** executes multiple queries to the Database to build the page context:
    - **Fetch Notifications:** It retrieves all unread alerts (e.g., deadlines or approval decisions) for the current user.
    - **Fetch Proposals:** It retrieves a summary list of the user's active proposals and their current status (e.g., "Pending", "Approved").

3. The **View** combines these datasets into a context object and renders the dashboard.html template, which is then displayed to the **Actor**.



4.1.4 Use Case 4 Reviewer（Evaluation/Scoring Proposals）

Phase 1: Initial Page Load (GET Request)

- **Request Initiation:** The process begins when the **Reviewer** clicks on the "Evaluate" button in the **Browser/UI**.
- **Django View Execution:** The Browser sends a GET evaluate_proposal(request, proposal_id) call to the **Django View**.
- **Data Retrieval:** The View performs a get_object_or_404(Proposal, pk=proposal_id) query to the **Database** to verify the proposal exists.
- **Response:** The Database returns the specific Proposal Data.
- **Rendering:** The Django View renders the evaluate_proposal.html template with the form and context, sending it back to the Browser/UI for the Reviewer to see.

Phase 2: Submission and Validation (Alt Fragment)

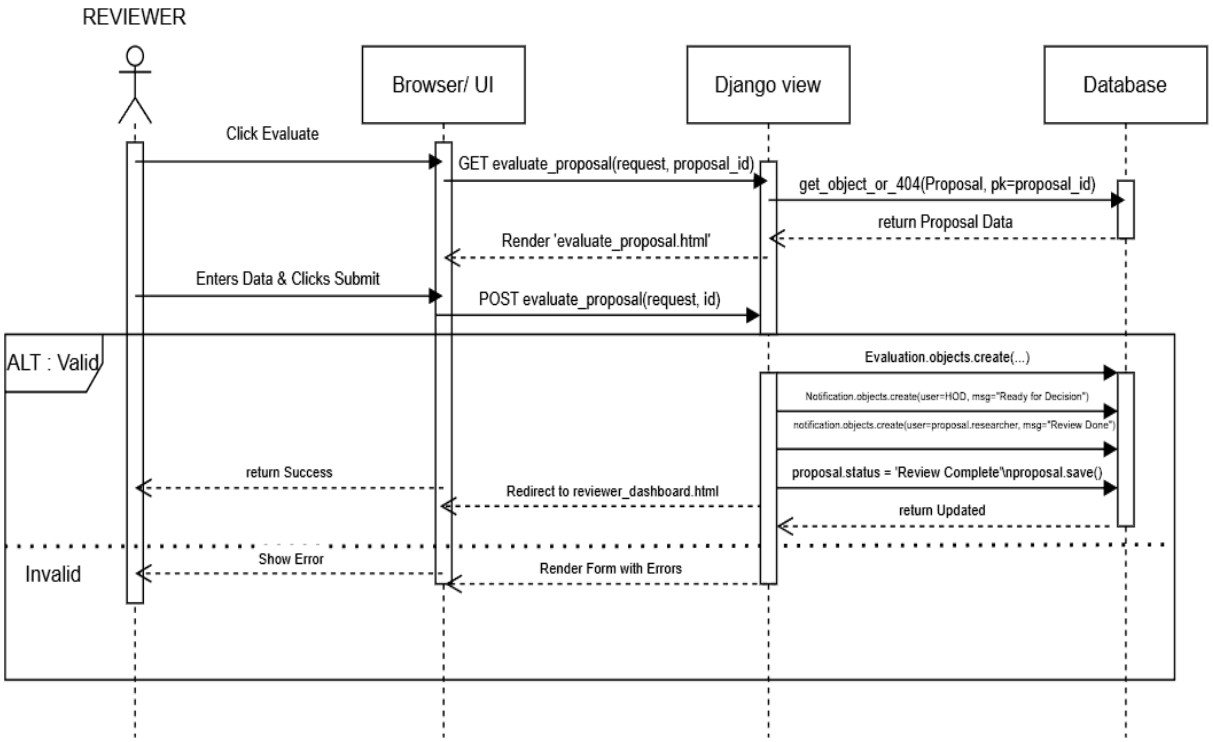
The Reviewer fills out the form and submits it. The system then enters a validation block (ALT: Valid) with two distinct paths:

Path A: Successful Submission (Valid Data)

- **Submission:** The Reviewer enters the Score and Comments, then clicks "Submit".
- **POST Request:** The Browser/UI sends a POST evaluate_proposal(request, id) request to the Django View.
- **Database Creation (Evaluation):** Since the form is valid, the View calls Evaluation.objects.create(...) to save the feedback in the **Database**.
- **Notifications:** The View triggers two creation events in the Database:
    1. Notification.objects.create(...) to alert the **HOD** ("Ready for Decision").
    2. notification.objects.create(...) to alert the **Researcher** ("Review Done").
- **Status Update:** The View updates the proposal status by calling proposal.status = 'Review Complete' followed by proposal.save().
- **Completion:** The Database confirms the update ("return Updated"), and the Django View redirects the user to reviewer_dashboard.html with a success message.

Path B: Invalid Submission (Error)

- **Validation Failure:** If the data entered is incomplete or incorrect (e.g., missing score), the validation check fails.
- **Error Handling:** The Django View re-renders the form specifically with the error details included ("Render Form with Errors").
- **User Feedback:** The Browser/UI displays the "Show Error" message to the Reviewer, allowing them to correct the input.



4.1.5 Use Case 5 (Approve/Reject Grant Proposals)

Phase 1: Initial Review (GET Request)

- **Request Initiation**: The process begins when the **Actor (HOD)** clicks on "Review and Approve" in the **Browser/UI**.
- **Django View Execution**: The UI sends an approve_proposal(request, proposal_id) call to the **Django View**.
- **Data Retrieval**: The View performs a get_object_or_404(Proposal, pk=proposal_id) query to the **Database**.
- **Response**: The Database returns the specific Proposal data and Reviewer feedback.
- **Rendering**: The Django View renders the approve_form.html template with the retrieved context and sends it back to the Browser/UI for the HOD to see.

Phase 2: Final Decision (Alt Fragment)

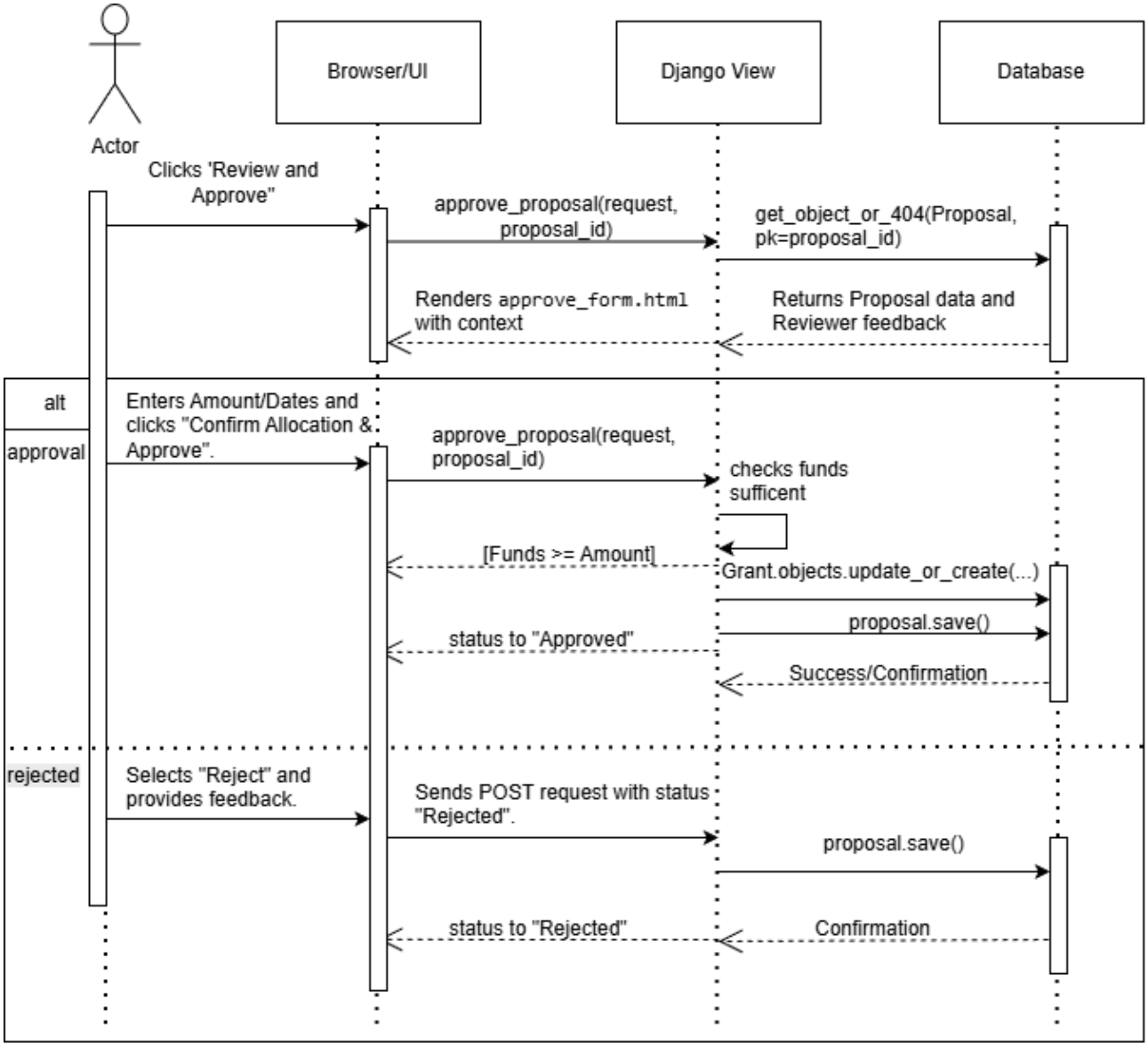
The HOD then makes a decision, which triggers one of two distinct paths:

Path A: Proposal Approval

- **Submission**: The HOD enters the allocation amount and dates, then clicks "Confirm Allocation & Approve".
- **Validation**: The Browser/UI sends a POST request to the Django View, which internally "checks funds sufficient".
- **Conditional Logic**: If the guard condition **[Funds >= Amount]** is met, the system proceeds with the following database updates:
    - **Grant Creation**: Calls Grant.objects.update_or_create(...).
    - **Status Update**: Calls proposal.save() to update the record.
- **Confirmation**: The Database returns a "Success/Confirmation" to the View, which then notifies the UI that the status is now "Approved".

Path B: Proposal Rejection

- **Submission**: The HOD selects "Reject" and provides written feedback.
- **POST Request**: The Browser/UI sends a POST request with the status "Rejected" to the Django View.
- **Database Update**: The View calls proposal.save() to finalize the rejection in the database.
- **Final Confirmation**: After the Database confirms the change, the Django View returns the updated status "Rejected" to the UI.


Actor
Browser/UI
Django View
Database
Clicks 'Review and Approve"
approve_proposal(request, proposal_id)
get_object_or_404(Proposal, pk=proposal_id)
Renders approve_form.html with context
Returns Proposal data and Reviewer feedback
alt
approval
Enters Amount/Dates and clicks "Confirm Allocation & Approve".
approve_proposal(request, proposal_id)
checks funds sufficent
[Funds >= Amount]
Grant.objects.update_or_create(...)
proposal.save()
status to "Approved"
Success/Confirmation
rejected
Selects "Reject" and provides feedback.
Sends POST request with status "Rejected".
proposal.save()
status to "Rejected"
Confirmation

4.1.6 Use Case 6 (Grant Allocation & Budget Tracking)

Phase 1: Accessing the Dashboard (GET Request)

- **Request Initiation**: The sequence begins when the **Actor (HOD)** clicks on the "Budget Tracking Dashboard" button in the **Browser/UI**.
- **Data Retrieval**: The Browser/UI sends an HTTP GET request to the Django View at the URL /hod/budget-dashboard/.
- **Database Query**: The **Django View** executes Grant.objects.all() to retrieve the list of projects from the **Database**.
- **Rendering**: After the Database returns the project budget list, the Django View renders the tracking_dashboard.html template and sends it back to the Browser/UI for display.

Phase 2: Budget Tracking (First Alt Fragment)

When the HOD decides to monitor a specific project, the following sub-sequence occurs:

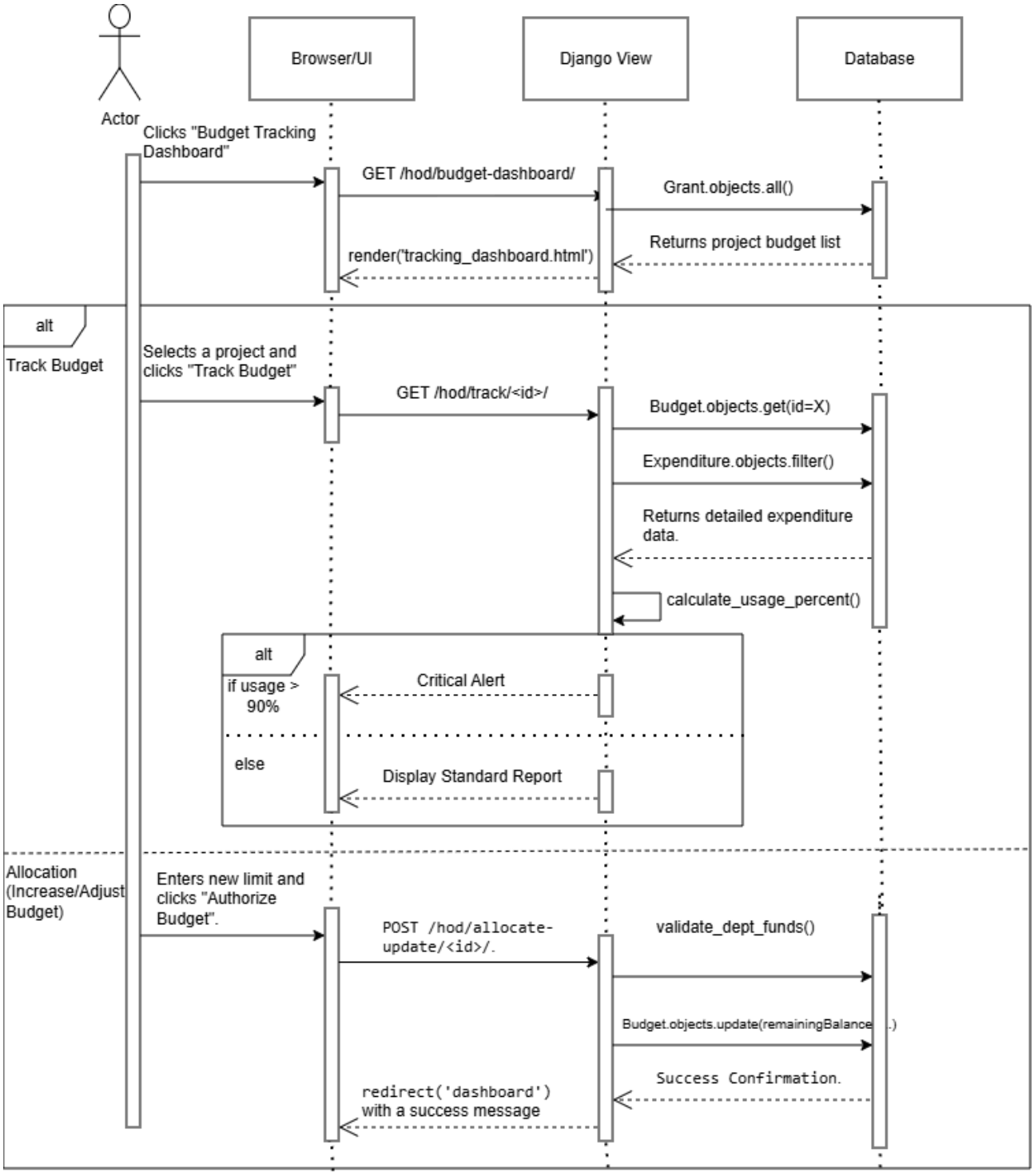
- **Project Selection**: The HOD selects a specific project and clicks "Track Budget".
- **Detailed Request**: The Browser/UI sends a GET request to /hod/track/<id>/.
- **Data Fetching**: The Django View fetches details from the Database using Budget.objects.get(id=X) and Expenditure.objects.filter().
- **Internal Calculation**: Once the detailed expenditure data is returned, the View runs an internal method, calculate_usage_percent(), to determine the current spending status.
- **Threshold Logic**:
    - **Critical Alert**: If the calculated usage is **greater than 90%**, the View sends a "Critical Alert" message to the UI.
    - **Standard Report**: Otherwise, it renders a "Standard Report" for the HOD.

Phase 3: Allocation Update (Second Alt Fragment)

If the HOD chooses to adjust the project's funding, the final part of the sequence is triggered:

- **Limit Entry**: The HOD enters a new budget limit and clicks the "Authorize Budget" button.
- **Update Submission**: The Browser/UI sends a POST request to /hod/allocate-update/<id>/.
- **Validation**: The Django View calls validate_dept_funds() to ensure the department has sufficient resources.
- **Database Update**: The View executes Budget.objects.update(remainingBalance=...) to save the new financial data.

- **Completion**: After receiving a "Success Confirmation" from the Database, the Django View redirects the HOD back to the dashboard with a success message.



4.1.7 Use Case 7 (Monitor Research Dashboard (KPIs))

Phase 1: Initial Dashboard Generation (GET Request)

- **User Action**: The process begins when the **Actor** (HOD) navigates to the "Department Analytics / KPI Dashboard" via the **Browser/UI**.
- **Request Initiation**: The **Browser/UI** sends an HTTP GET request to the Django View at the URL /hod/analytics-dashboard/.
- **Data Retrieval**: The **Django View** interacts with the **Database** by calling Executes aggregate queries to pull department-wide performance data.
- **Database Response**: The **Database** returns the aggregated data points to the View.
- **Data Processing**: The **Django View** then performs an internal calculation, calculate_performance_metrics(), to prepare the data for visualization.
- **Rendering**: The View renders the analytics_dashboard.html template and sends it to the **Browser/UI**, which displays the interactive dashboard to the HOD.

Phase 2: Action Selection (Frame Fragment)

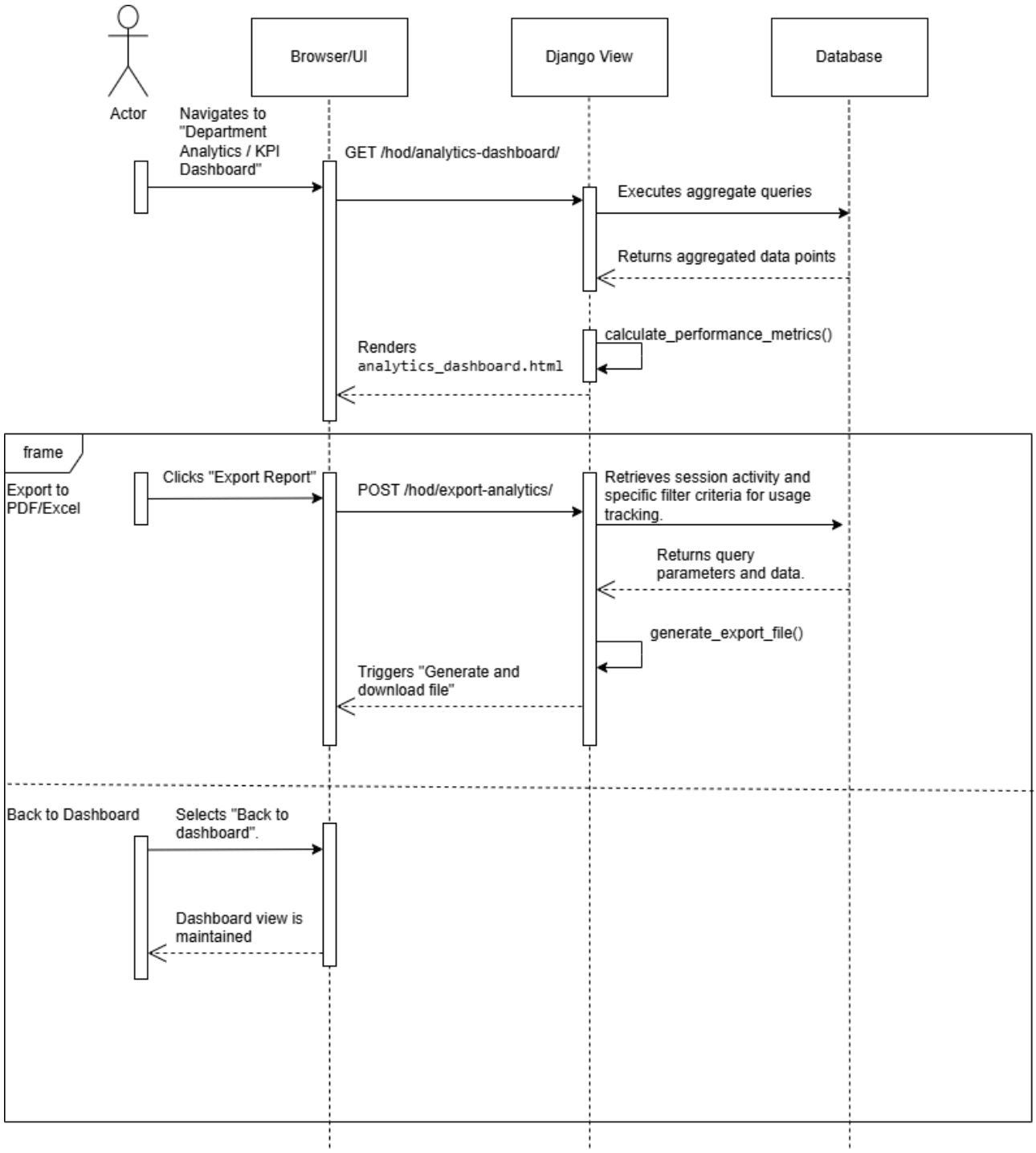
Once the dashboard is displayed, the HOD has two distinct options represented in the frame:

Option A: Export to PDF/Excel

- **User Action**: The HOD clicks the **"Export Report"** button.
- **POST Submission**: The **Browser/UI** sends an HTTP POST request to /hod/export-analytics/.
- **Context Retrieval**: The **Django View** communicates with the **Database** to retrieve session activity and specific filter criteria to ensure usage tracking.
- **Data Preparation**: The **Database** returns the query parameters and raw data to the View.
- **File Generation**: The **Django View** executes the internal function generate_export_file().
- **Delivery**: The View sends a signal back to the **Browser/UI**, which triggers a "Generate and download file" prompt for the user.

Option B: Back to Dashboard

- **User Action**: Alternatively, the HOD selects the **"Back to dashboard"** option.
- **Return Call**: The sequence concludes with a simple return message stating that the **"Dashboard view is maintained"** in the **Browser/UI**, with no further database interaction required.


Actor
Browser/UI
Django View
Database
Navigates to "Department Analytics / KPI Dashboard"
GET /hod/analytics-dashboard/
Executes aggregate queries
Returns aggregated data points
calculate_performance_metrics()
Renders analytics_dashboard.html
frame
Export to PDF/Excel
Clicks "Export Report"
POST /hod/export-analytics/
Retrieves session activity and specific filter criteria for usage tracking.
Returns query parameters and data.
generate_export_file()
Triggers "Generate and download file"
Back to Dashboard
Selects "Back to dashboard".
Dashboard view is maintained

4.1.8 Use Case 8 (Monitor Research Project Progress)

Phase 1: Project List Retrieval (GET Request)

- **Request Initiation**: The process begins when the **Actor (HOD)** navigates to the project monitoring dashboard.
- **Data Fetching**: The **Browser/UI** sends an HTTP GET request to /hod/monitor-projects/.
- **Database Query**: The **Django View** executes Project.objects.all() to retrieve the full list of projects.
- **Rendering**: The **Database** returns the project list along with high-level milestones, which the View uses to render the project_list.html template for the HOD.

Phase 2: Detailed Project Review (GET Request)

- **Selection**: The HOD selects a specific project from the list for a more detailed review.
- **Detailed Request**: The **Browser/UI** sends a GET request for /hod/project-detail/<id>/.
- **Metric Retrieval**: The **Django View** communicates with the **Database** to retrieve specific Key Performance Indicators (KPIs), recent progress reports, and any adherence issues.
- **Display**: The **Database** returns the project timeline and completion percentage, which is then displayed to the HOD on a detailed dashboard.

Phase 3: Decision and Intervention (Frame Fragment)

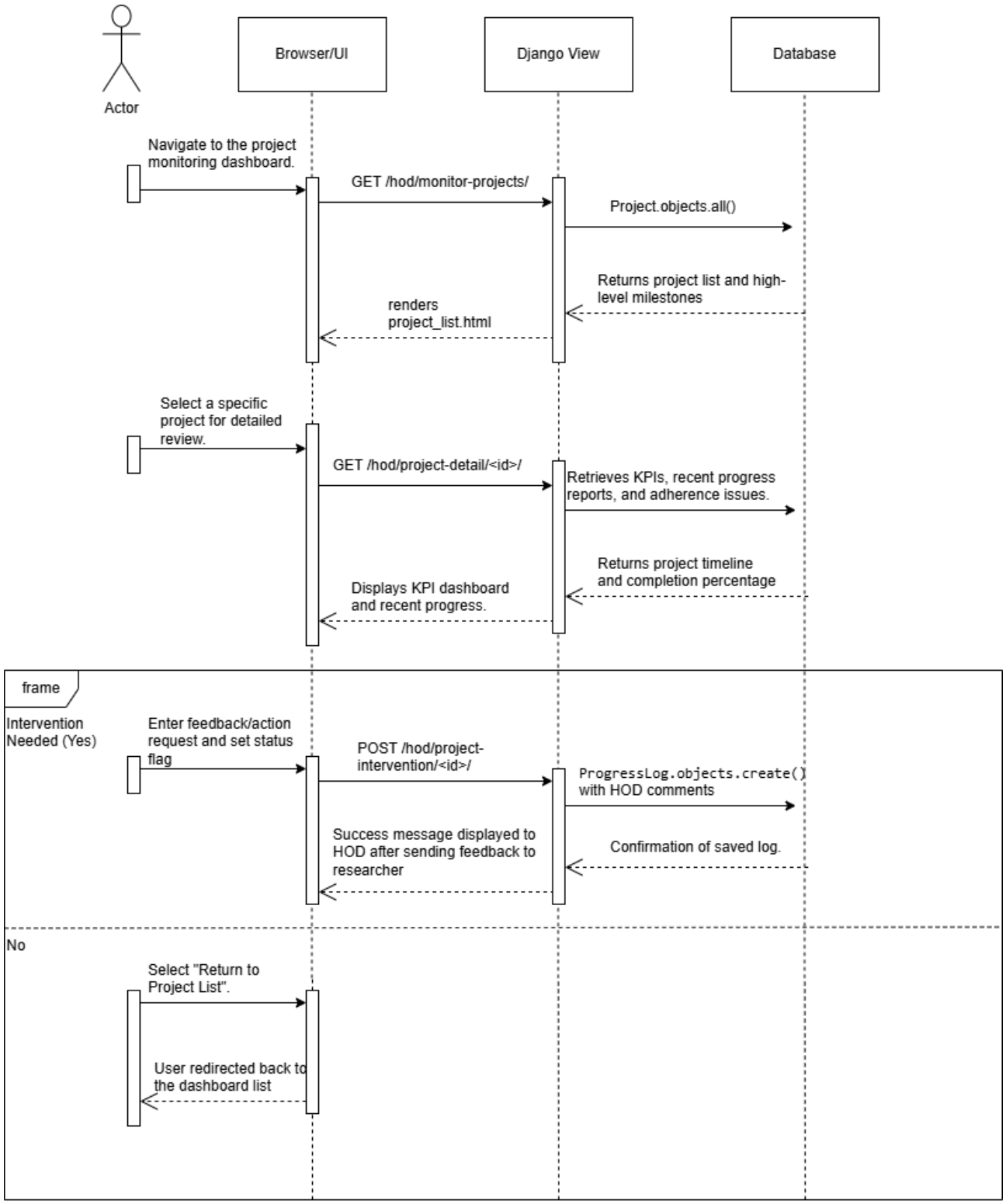
Following the review, the HOD decides whether the project requires direct intervention, leading to two possible paths:

Path A: Intervention Needed (Yes)

- **Action Entry**: The HOD enters feedback or an action request and sets a status flag.
- **POST Submission**: The **Browser/UI** sends a POST request to /hod/project-intervention/<id>/.
- **Logging**: The **Django View** executes ProgressLog.objects.create() to store the HOD's comments in the system.
- **Confirmation**: Upon receiving a confirmation of the saved log from the **Database**, the View displays a success message to the HOD indicating the feedback has been sent to the researcher.

Path B: No Intervention (No)

- **Exit**: The HOD selects "Return to Project List".
- **Redirect**: The **Browser/UI** concludes the sequence by redirecting the HOD back to the primary dashboard list without further database updates.


Actor
Browser/UI
Django View
Database
Navigate to the project monitoring dashboard.
GET /hod/monitor-projects/
Project.objects.all()
Returns project list and high-level milestones
renders project_list.html
Select a specific project for detailed review.
GET /hod/project-detail/<id>/
Retrieves KPIs, recent progress reports, and adherence issues.
Returns project timeline and completion percentage
Displays KPI dashboard and recent progress.
frame
Intervention Needed (Yes)
Enter feedback/action request and set status flag
POST /hod/project-intervention/<id>/
ProgressLog.objects.create() with HOD comments
Success message displayed to HOD after sending feedback to researcher
Confirmation of saved log.
No
Select "Return to Project List".
User redirected back to the dashboard list

4.2 State Diagram

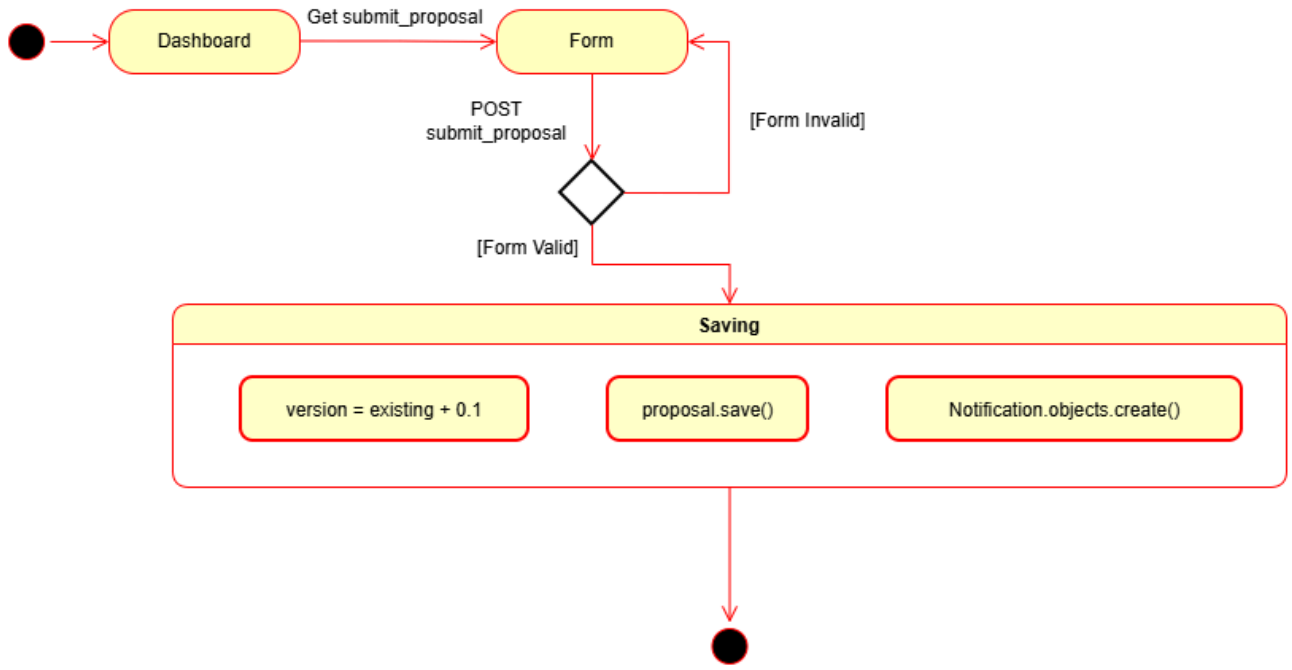
4.2.1 State Diagram 1（Upload Research Proposals and Reports）

Description of States:

Researcher Dashboard (Initial State): The starting point where the researcher views their current projects. A GET request for the upload form initiates the transition.
Uploading Form: An active state where the user selects files and fills in metadata.

- **Guard Condition [Form invalid]:** If validation fails (e.g., missing fields or wrong file format), the system reverts to this state to display errors.
- **Guard Condition [Form valid]:** Triggered by a POST request, moving the process to the backend.

Saving Proposal/Report: A transient backend state where the system executes file.save() and writes the initial record to the database.
Updating System Log: The system automatically generates an audit trail entry or updates the researcher's activity log to reflect the new submission.
Submission Complete (Final State): The terminal state confirming the upload is successful. The user is typically redirected back to the dashboard with a success message.



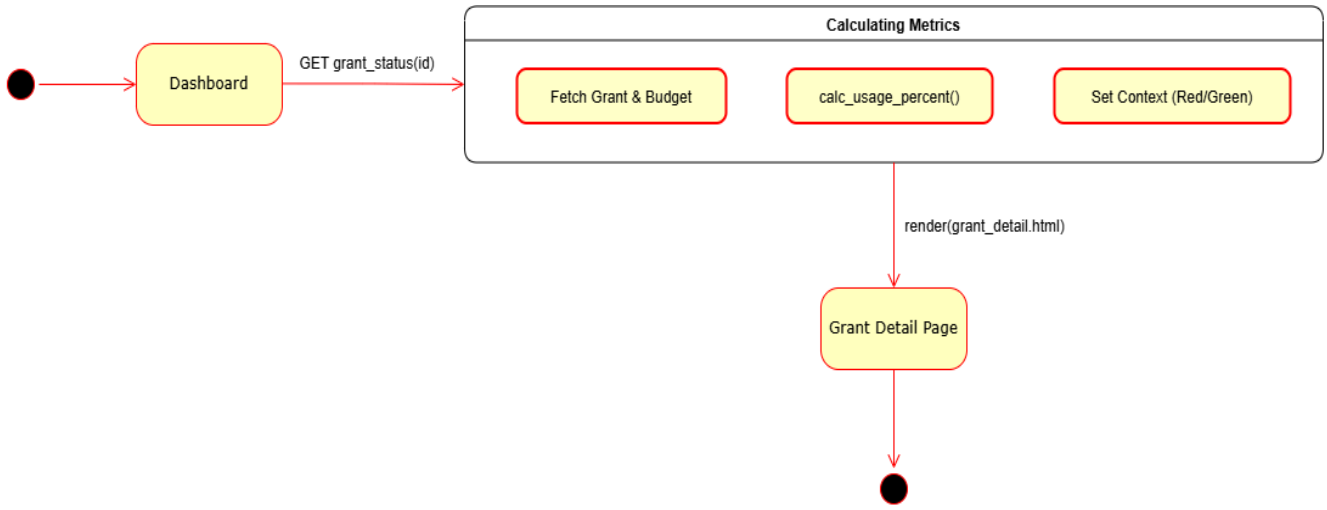
4.2.2 State Diagram 2（Track Grant Status & Budget Usage）

Description of States:

Grant Overview (Initial State): Displays a high-level list of all grants assigned to the user.
Fetching Grant Data: Once a specific grant is selected, the system enters this state to query the database for all related transactions, allocations, and expenses.
Calculating Budget Metrics: A computational state where the system applies business logic to determine:

- Total Spent vs. Remaining Balance.
- Burn rate or percentage of budget utilized.
- Flagging any over-expenditures.

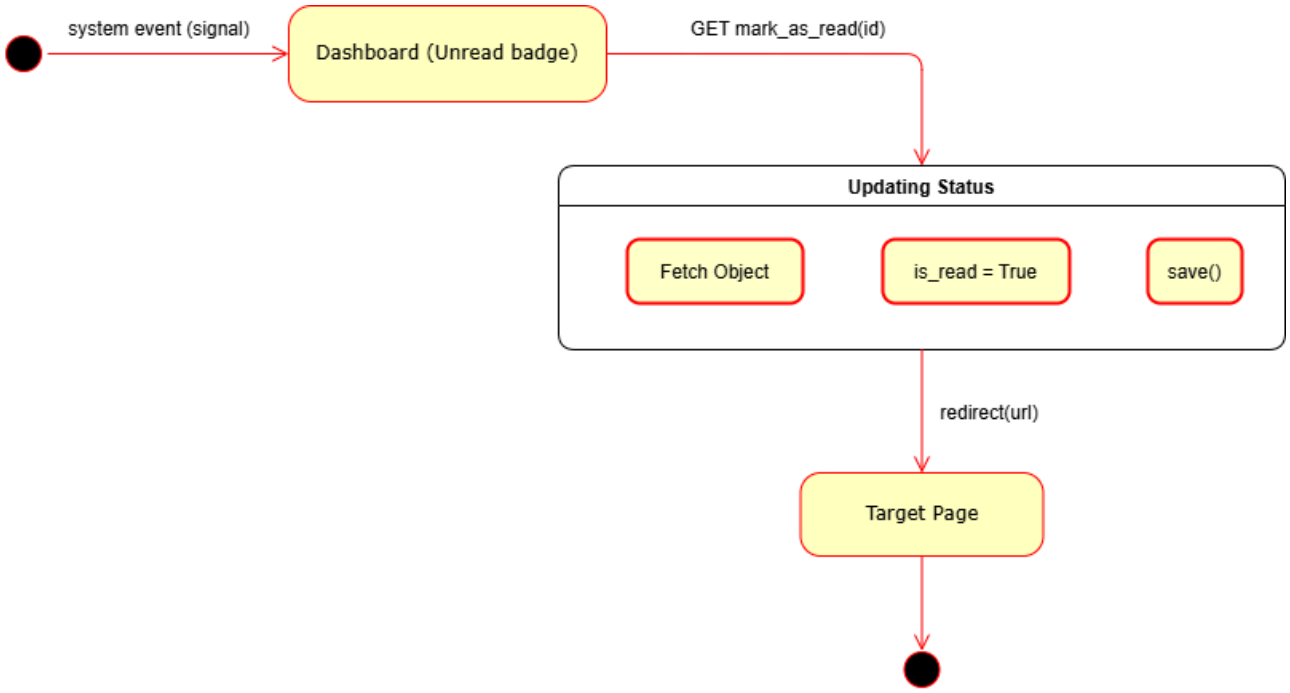
Displaying Grant Details (Final State): The "Grant Detail Page" where the calculated metrics are rendered into charts or tables for the user. From here, the user can navigate back to the overview.



4.2.3 State Diagram 3（Receive Notifications）

Description of States:

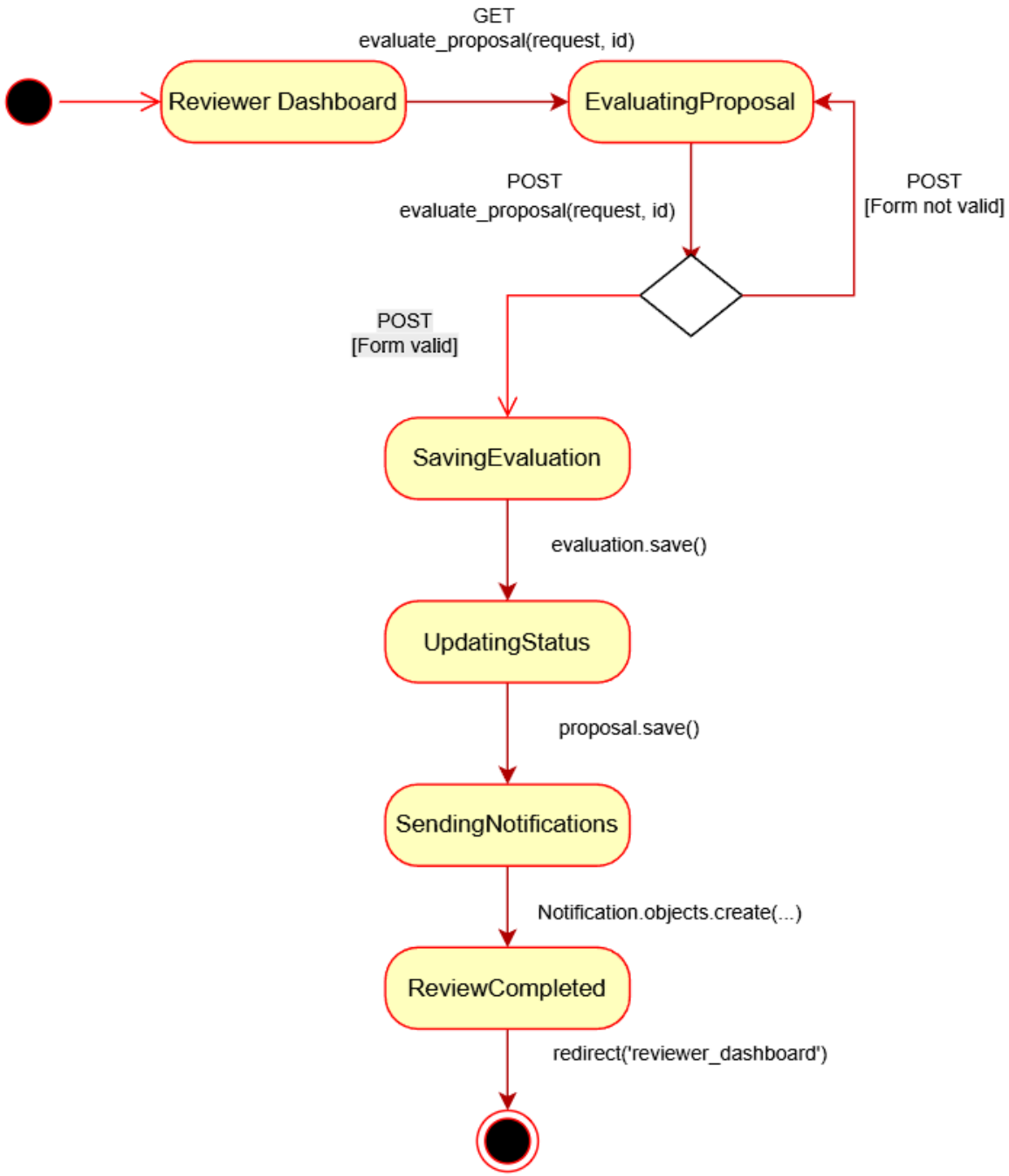
User Dashboard (Initial State): The dashboard displays a notification badge (e.g., an unread count). The state changes when a user clicks the notification icon or a specific alert.
Marking As Read: A backend state where the system executes notification.status = 'read'. This ensures the UI badge updates and the notification no longer appears as "new."
Redirecting To Resource: The system identifies the target_url or object ID associated with the notification (e.g., a specific proposal that was just reviewed).
Resource View (Final State): The terminal state where the user arrives at the relevant page (the "target page"). The notification cycle ends here as the user is now positioned to take action on the actual item.



4.2.4 State Diagram 4（Evaluation/Scoring Proposals）

Description of States:

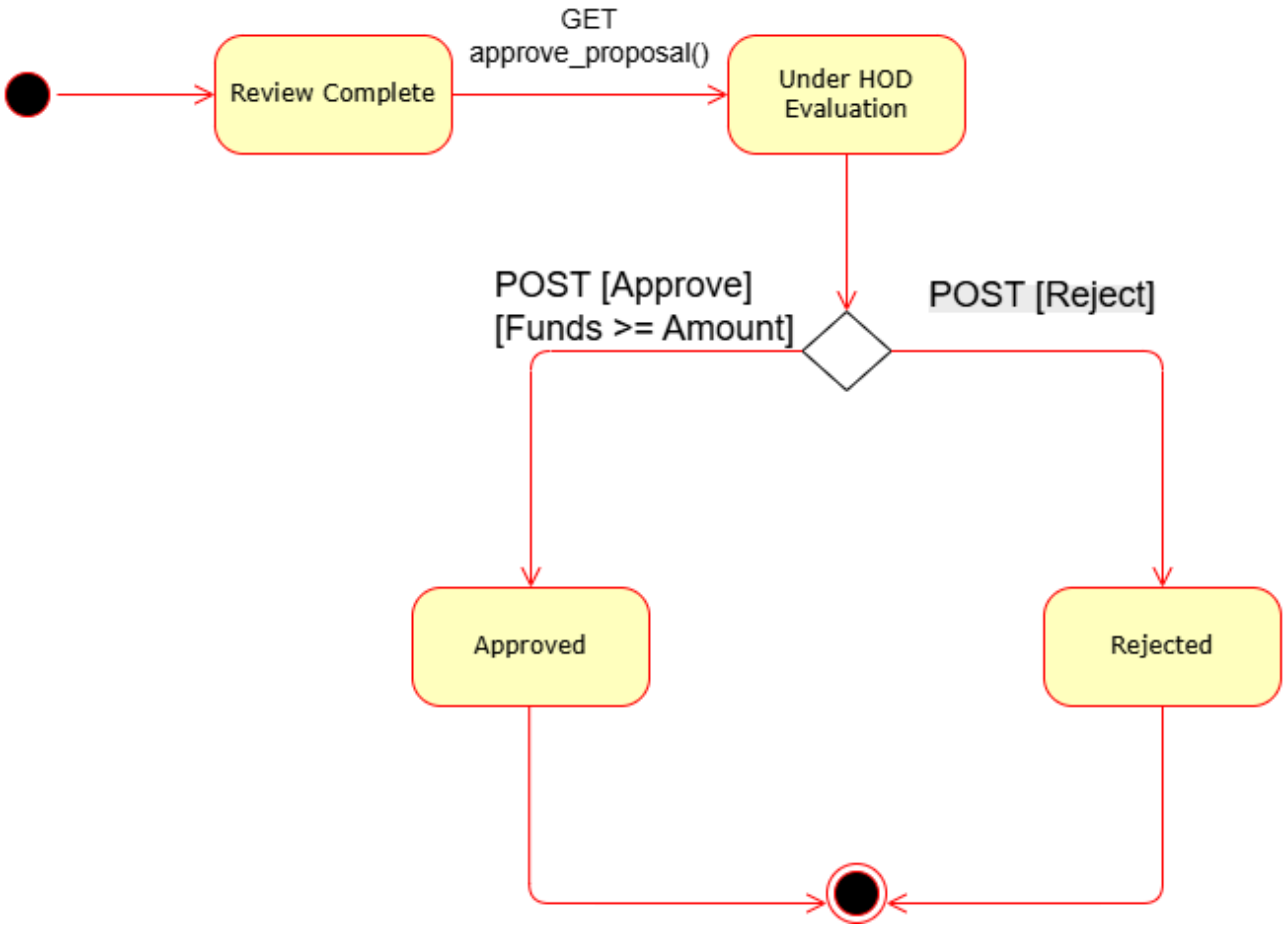
1. **Reviewer Dashboard (Initial State):** This state represents the reviewer's entry point into the system. It displays a list of assigned proposals awaiting evaluation. The process remains in this state until a `GET` request is initiated to evaluate a specific proposal.
2. **EvaluatingProposal:** An active state where the reviewer interacts with the scoring interface. The system remains in this state while the user inputs qualitative and quantitative data. It includes a validation loop: if the Guard Condition `[Form not valid]` is met upon submission, the system reverts to this state to display error messages.
3. **SavingEvaluation:** A transient backend state triggered once the Guard Condition `[POST][Form valid]` is satisfied. In this state, the system invokes `evaluation.save()` to commit the reviewer's scores and feedback to the database.
4. **UpdatingStatus:** In this state, the system automatically updates the metadata of the proposal itself via `proposal.save()`. This transition ensures that the proposal is marked as "Evaluated" in the global tracking system, moving it out of the reviewer's active queue.
5. **SendingNotifications:** A functional state where the system triggers `Notification.objects.create(...)`. This automated action alerts relevant stakeholders (such as the Lead Researcher or HOD) that the evaluation has been finalized and is ready for the next phase of the workflow.
6. **ReviewCompleted (Final State):** This terminal state signifies that all scoring and background processing for the proposal are finished. Reaching this state triggers a redirect back to the `reviewer_dashboard`, effectively concluding the evaluation lifecycle for that specific record.


GET
evaluate_proposal(request, id)
Reviewer Dashboard
EvaluatingProposal
POST
evaluate_proposal(request, id)
POST
[Form not valid]
POST
[Form valid]
SavingEvaluation
evaluation.save()
UpdatingStatus
proposal.save()
SendingNotifications
Notification.objects.create(...)
ReviewCompleted
redirect('reviewer_dashboard')

4.2.5 State Diagram 5 (Approve/Reject Grant Proposals)

Description of States:

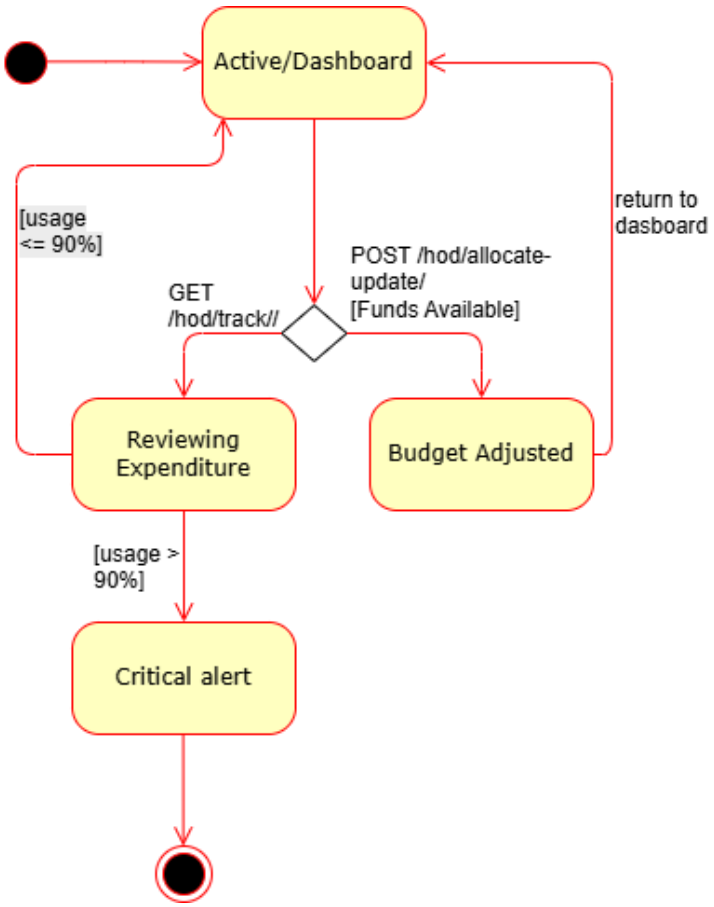
1. **Review Complete (Initial State):** This state represents a proposal that has successfully passed the reviewer phase. All required evaluations have been submitted, and the proposal is strictly read-only for the researcher while waiting in the HOD's queue.
2. **Under HOD Evaluation:** The proposal enters this active state when the HOD opens the decision interface. In this state, the system aggregates financial data (Department Reserve) and qualitative data (Reviewer Scores) to facilitate a decision. The proposal remains in this state until a definitive POST action is triggered.
3. **Approved (Final State):** This terminal state signifies that the grant has been authorized. Reaching this state implies that the **Guard Condition** [Funds >= Amount] was satisfied, meaning the department had sufficient budget to cover the grant allocation. Once in this state, the funds are deducted, and the project becomes active.
4. **Rejected (Final State):** This terminal state signifies that the HOD has denied the funding request. This state is reached immediately if the HOD selects the "Reject" action. No funds are deducted, and the workflow for this specific proposal ends here.



4.2.6 State Diagram 6 (Grant Allocation & Budget Tracking)

Description of States:

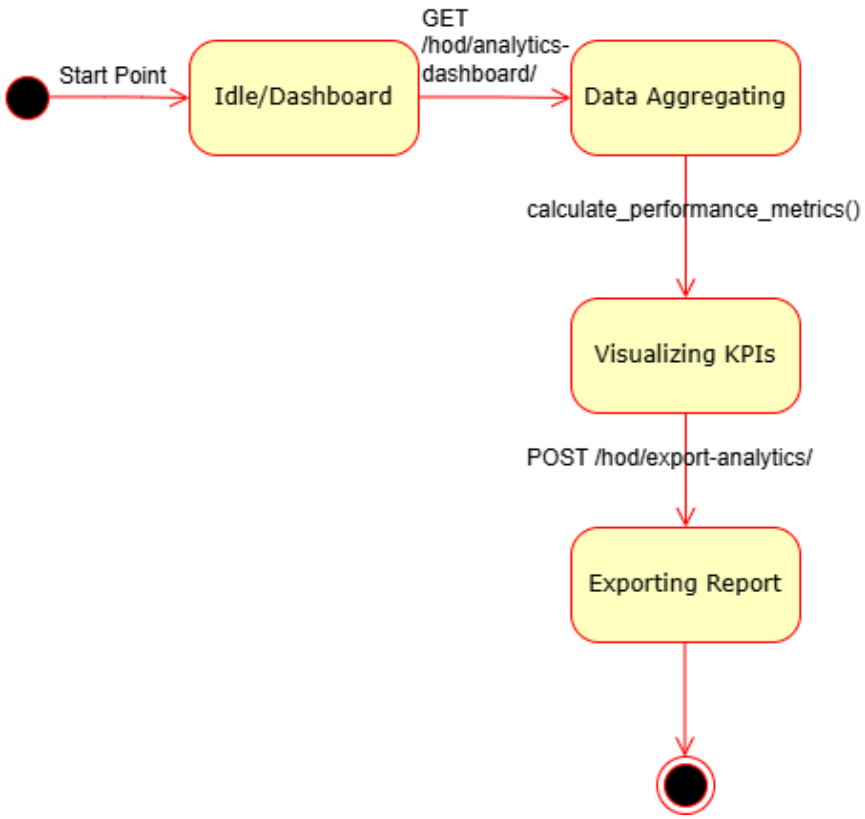
1. **Active/Dashboard (Initial State):** This is the default monitoring state where the HOD oversees all active grants. The system remains in this state, displaying high-level summaries, until the HOD selects a specific grant for detailed financial review or modification.
2. **Reviewing Expenditure:** The system transitions to this state when the HOD initiates a GET /hod/track/ request to view a specific grant's ledger. In this state, the system automatically evaluates the budget health:
    - If the usage is safe ([<= 90%]), the workflow loops back, allowing the HOD to return to the Dashboard.
    - If the usage is high ([> 90%]), the system triggers a transition to the **Critical Alert** state.
3. **Budget Adjusted:** This state signifies that a fund allocation (top-up) has been successfully processed. It is reached via a POST action, conditional on the [Funds Available] guard check. After the adjustment is recorded, the system automatically transitions back to the Active/Dashboard state.
4. **Critical Alert (Terminal State):** This is a specialized warning state entered strictly when budget usage exceeds 90%. It serves as a system-enforced halt to notify the HOD that the specific grant requires immediate financial intervention.



4.2.7 State Diagram 7 (Monitor Research Dashboard (KPIs))

Description of States:

1. **Idle/Dashboard (Initial State):** This is the resting state where the system waits for user interaction. The HOD is on the main menu or a different dashboard view, and the analytics module is inactive until specifically invoked.
2. **Data Aggregating:** The system enters this transient processing state immediately after receiving the GET /hod/analytics-dashboard/ request. In this state, the backend logic queries the database to gather raw data on budget utilization, proposal counts, and success rates before any visualization can occur.
3. **Visualizing KPIs:** This is the primary display state reached after the calculate_performance_metrics() function completes. Here, the aggregated data is rendered into graphical formats (Doughnut Charts for budget, Bar Charts for status) for the HOD to review. The system remains in this state to allow the user to analyze the information.
4. **Exporting Report (Final State):** This state is triggered when the HOD initiates a POST /hod/export-analytics/ request. The system compiles the currently visualized data into a downloadable format (e.g., PDF or CSV). Once the download is generated, the specific analytics workflow concludes, leading to the terminal node.



5 Architecture Design

5.1 Software Architecture

The **RGMS** is designed using the **Layered Architecture** style, as defined in our design curriculum. This structure separates the system into distinct layers of responsibility, ensuring that the User Interface is decoupled from the Business Logic and Data Storage. We have implemented this using the Three-Tier pattern within the **Django** Framework:

- **User Interface Layer:** (Presentation): Handles user interactions (Templates).
- **Application Layer:** Contains the business logic and subsystems.
- **Core/Data Layer:** Manages persistence (Database).

This architectural style was chosen to ensure tight integration between the data model and business logic while maintaining a clear separation of concerns. As illustrated in the Main Architecture Diagram, the system is organized into three distinct logical layers hosted within a centralized Cloud Server Environment:

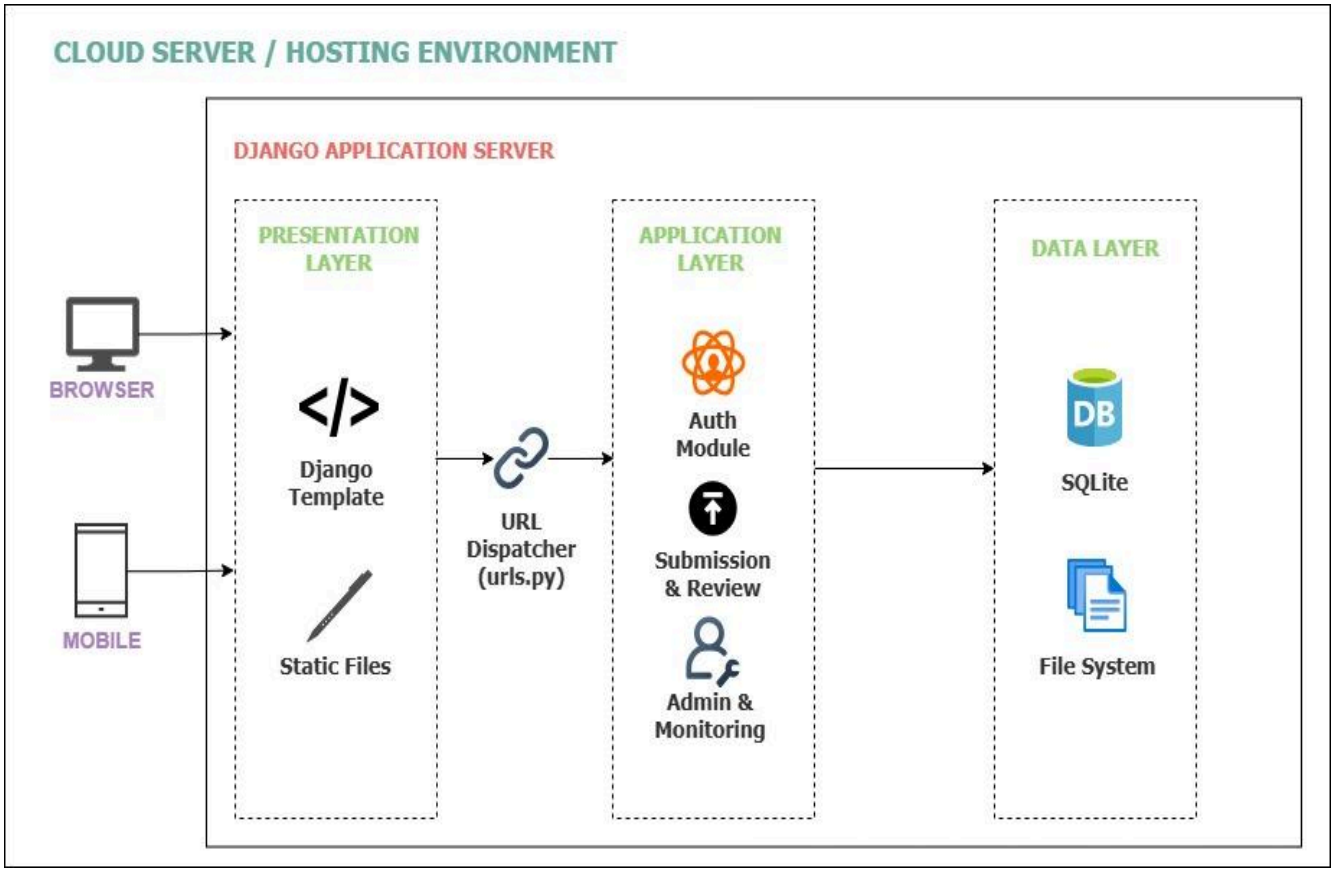
1. **Presentation Layer (Client-Side Interface):** Located on the client-facing side, this layer handles all user interactions. It utilizes the **Django Template** Engine to dynamically **render** HTML, CSS, and JavaScript. This layer is responsible for presenting data to the user and collecting input via forms (e.g., proposal submissions), which are then routed to the application layer via the **URL Dispatcher.**

2. **Application Layer (Business Logic):** This central layer acts as the system's **"brain,"** processing requests and enforcing business rules. It is composed of modular components corresponding to the system's key functional areas:

    - **Authentication Module:** Manages user identity, ensuring strict role-based access control for **Researchers, Reviewers, and HODs**.

    - **Subsystem Modules:** Encapsulates specific workflows, including the Researcher Subsystem (Proposal Submission), Reviewer Subsystem (Evaluation), and Admin Subsystem (Grant Allocation).

3. **Data Layer (Persistence):** The backend storage layer relies on a Relational **Database** (PostgreSQL or SQLite) to maintain structured records for Users, Proposals, and Budgets. Additionally, a dedicated **File System storage** is utilized to securely manage unstructured data, specifically the uploaded research proposal documents (PDFs).


CLOUD SERVER / HOSTING ENVIRONMENT
DJANGO APPLICATION SERVER
PRESENTATION LAYER
APPLICATION LAYER
DATA LAYER
BROWSER
MOBILE
Django Template
Static Files
URL Dispatcher (urls.py)
Auth Module
Submission & Review
Admin & Monitoring
SQLite
File System

5.1.1 Subsystem 1

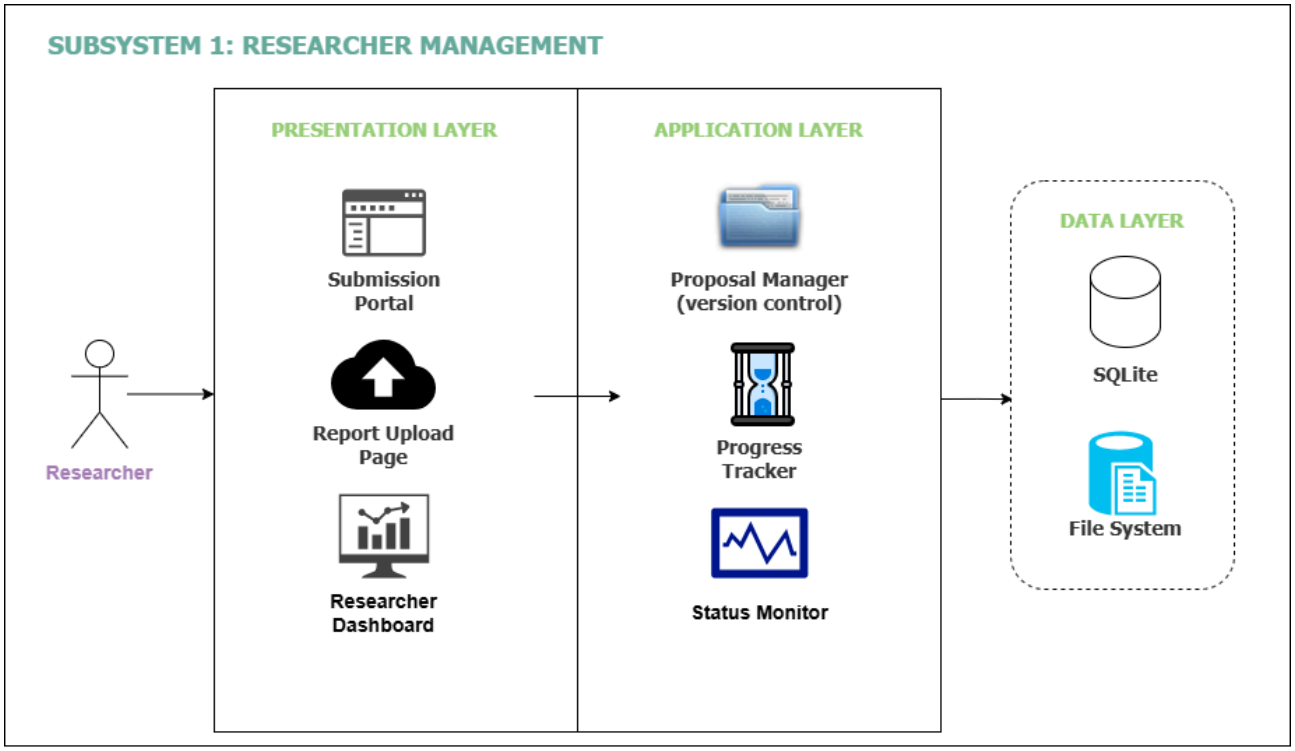
This **subsystem** encapsulates all workflows initiated by the **Researcher** actor. It is designed to handle the **"Entry"** and **"Monitoring**" phases of the grant lifecycle. The subsystem follows the application's three-tier structure, dividing responsibilities between the User Interface (UI) and the underlying Logic Modules.
Key Functional Modules:

- **1. Proposal Manager (Logic):** This is the core input module for the subsystem. It processes data received from the **Submission Portal UI**. Its primary responsibility is to handle the creation of new grant proposals and manage **Document Version Control**. When a researcher updates a proposal, this manager checks existing records in the database and automatically increments the version number (e.g., from 1.0 to 1.1) to ensure data integrity is preserved.

- **2. Progress Tracker (Logic):** This module manages the post-award phase. It interfaces with the **Report Upload Page** to allow **researchers** to submit milestone updates. It validates that the report is attached to an active grant before saving the **ProgressReport** entity to the **database.**

- **3. Status Monitor (Logic):** This module powers the **Researcher Dashboard**. It is a read-only retrieval engine that aggregates data from multiple sources to provide a "single pane of glass" view. It queries the **Notification System** for unread alerts and the **Budget** table to calculate and display real-time financial metrics (Total Allocated vs. Total Spent).

Data Flow: Interaction begins when the **Researcher** inputs data via the **Presentation Layer**. The specific Logic Module (e.g., Proposal Manager) validates this input and commits it to the **Data Layer**. Conversely, when the **Researcher** requests to view their status, the **Status Monitor** retrieves the relevant records from the **Database** and passes them back to the **UI** for display.



Subsystem 2

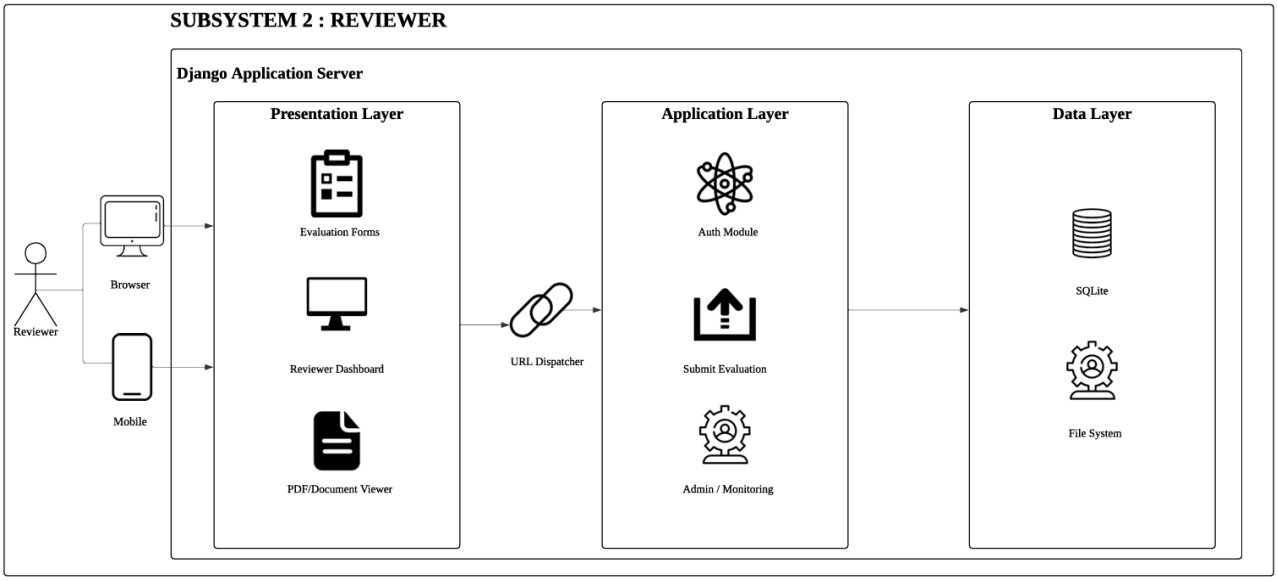
This **subsystem** encapsulates the specialized workflows performed by th**e Reviewer actor** within the Research Grant Management System. It is responsible for the systematic assessment of submitted proposals and the hand-off of recommendations to the Head of Department (HOD). Like the Researcher's subsystem, this follows a three-tier architecture to separate evaluation interfaces from backend logic.

Key Functional Modules:

1. **Evaluation Manager (Logic)**: This is the core engine for the review process. It processes inputs from the Evaluation Forms in the Presentation Layer. Its primary responsibility is to store quantitative scores and qualitative feedback in the Database. It ensures that an evaluation is "locked" once submitted to maintain the integrity of the review.
2. **Assignment Engine (Logic)**: This module powers the Reviewer Dashboard. It filters the Database to display only the specific grant proposals assigned to the logged-in Reviewer. It works closely with the Auth Module to verify that a user has the correct "Reviewer" permissions before granting access to sensitive proposal documents.
3. **HOD Dispatcher (Logic):** This module handles the "Submit Evaluation" workflow. Once a Reviewer completes their assessment, the Dispatcher updates the proposal's status (e.g., from "Under Review" to "Review Completed"). It packages

the reviewer's recommendation and triggers a notification to the HOD's dashboard, ensuring a seamless transition between the two roles.

Data Flow: The workflow begins when the Reviewer accesses the Reviewer Dashboard via a Browser or Mobile device. The Assignment Engine retrieves pending proposals from the Data Layer and displays them. When the Reviewer submits an assessment via the Evaluation Forms, the Evaluation Manager validates the data and saves it to the Database. Finally, the HOD Dispatcher updates the proposal status, making the evaluation results visible to the HOD for final decision-making.



5.2.1 Subsystem 3

This **subsystem** encapsulates all workflows initiated by the **HOD (Head of Department)** actor. It is designed to handle the **"Governance"**, **"Financial Oversight"**, and **"Strategic Monitoring"** phases of the grant lifecycle. The subsystem follows the application's three-tier structure, dividing responsibilities between the administrative User Interface (UI) and the underlying Logic Modules.

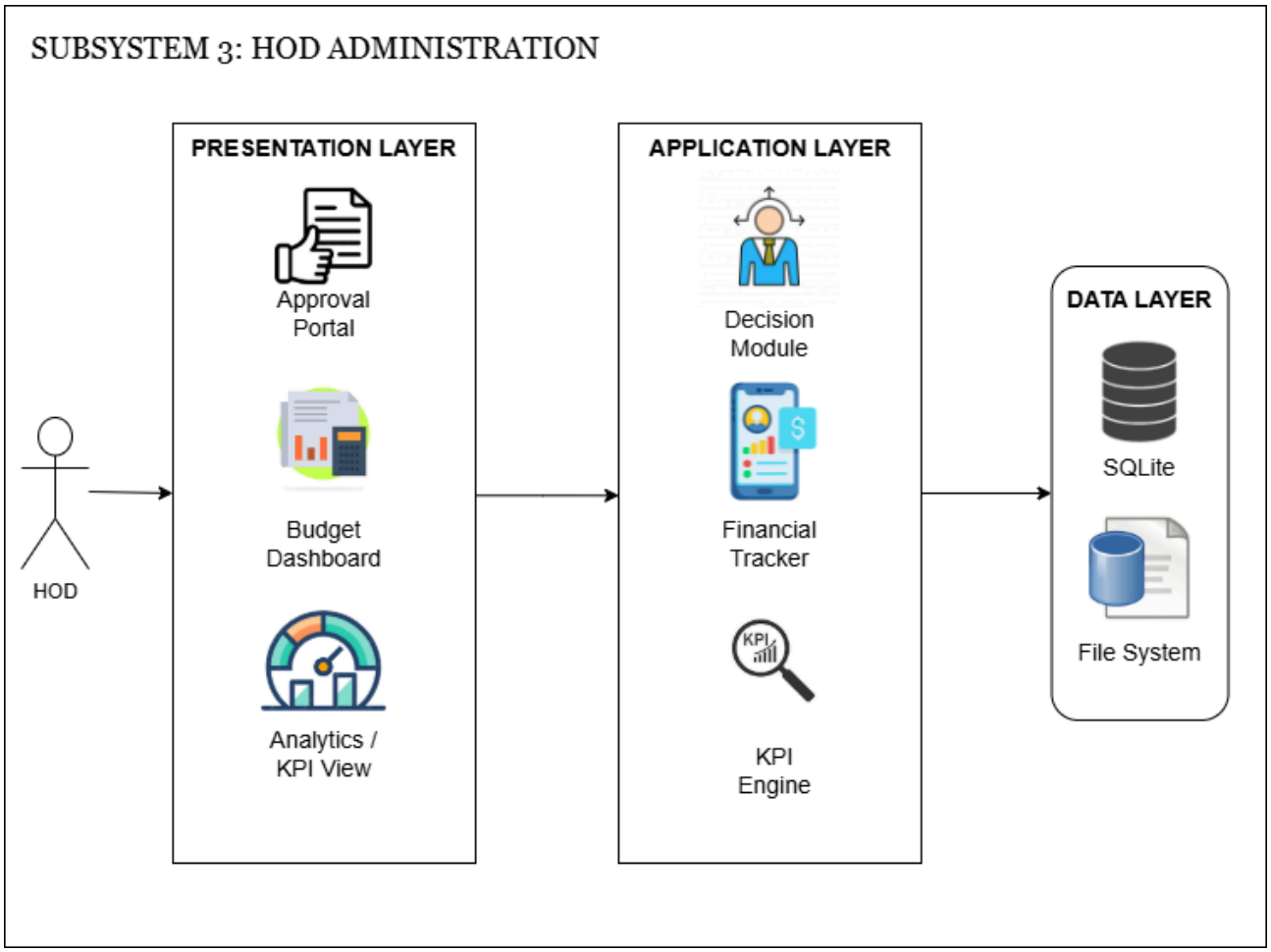
Key Functional Modules:

1. **Decision Module (Logic):** This is the primary control module for grant initiation. It processes requests from the **Approval Portal UI**. Its primary responsibility is to execute the "Approve/Reject" logic for new proposals. When an HOD approves a proposal, this module triggers the creation of the **Grant** entity and ensures the initial funding status is strictly validated against available department resources.
2. **Financial Tracker (Logic):** This module manages the ongoing fiscal health of active grants. It interfaces with the **Budget Dashboard** to allow the HOD to adjust allocations and monitor spending. It contains the specific business rule for the **"90%**

Threshold Alert", automatically flagging projects that are nearing budget exhaustion and preventing unauthorized overspending before committing updates to the **Database**.

3. **KPI & Analytics Engine (Logic):** This module powers the **Department Analytics Dashboard**. It is a high-level aggregation engine that queries multiple tables (Proposals, Progress Reports, Budgets) to calculate performance metrics. It handles the data processing for "Burn Rates" and "Completion Percentages" and manages the **Export** function to generate PDF/Excel reports for external stakeholders.

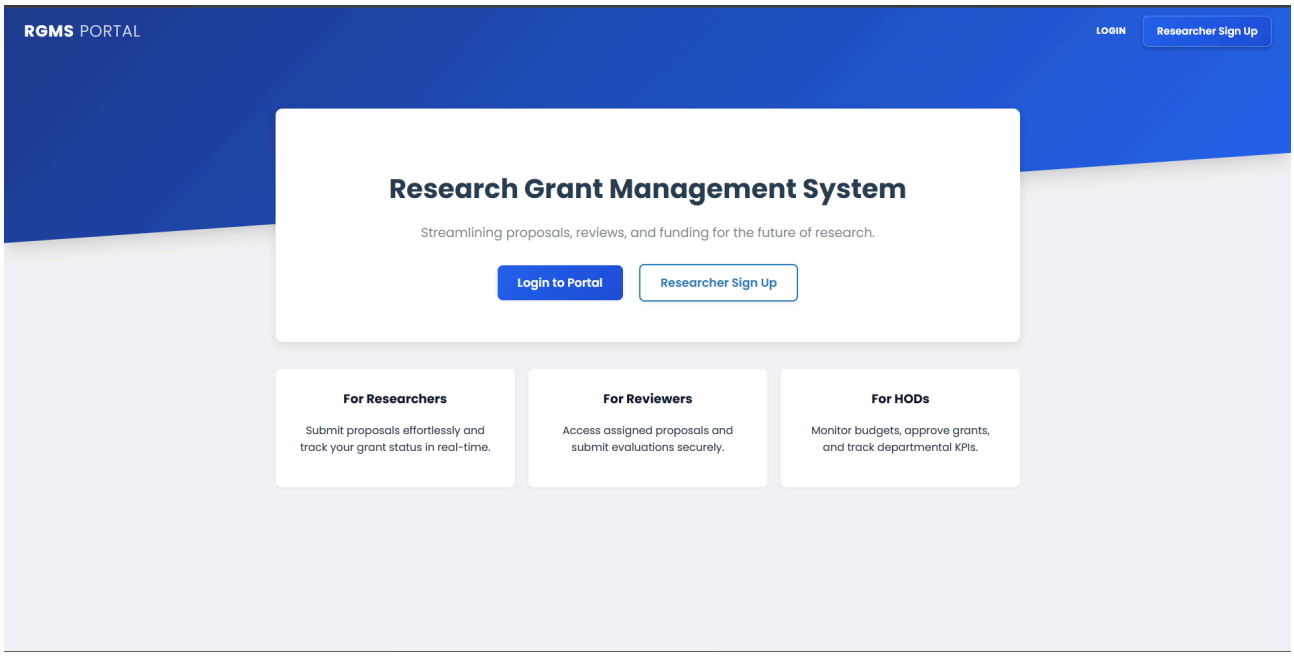
Data Flow: Interaction begins when the **HOD** inputs a decision or request via the **Presentation Layer**. The specific Logic Module (e.g., Decision Module) validates this input (checking funds or permissions) and commits the transaction to the **Data Layer**. Conversely, when the HOD requests an analytics report, the **KPI Engine** aggregates the relevant records from the **Database**, processes the metrics, and passes them back to the **UI** for visualization.



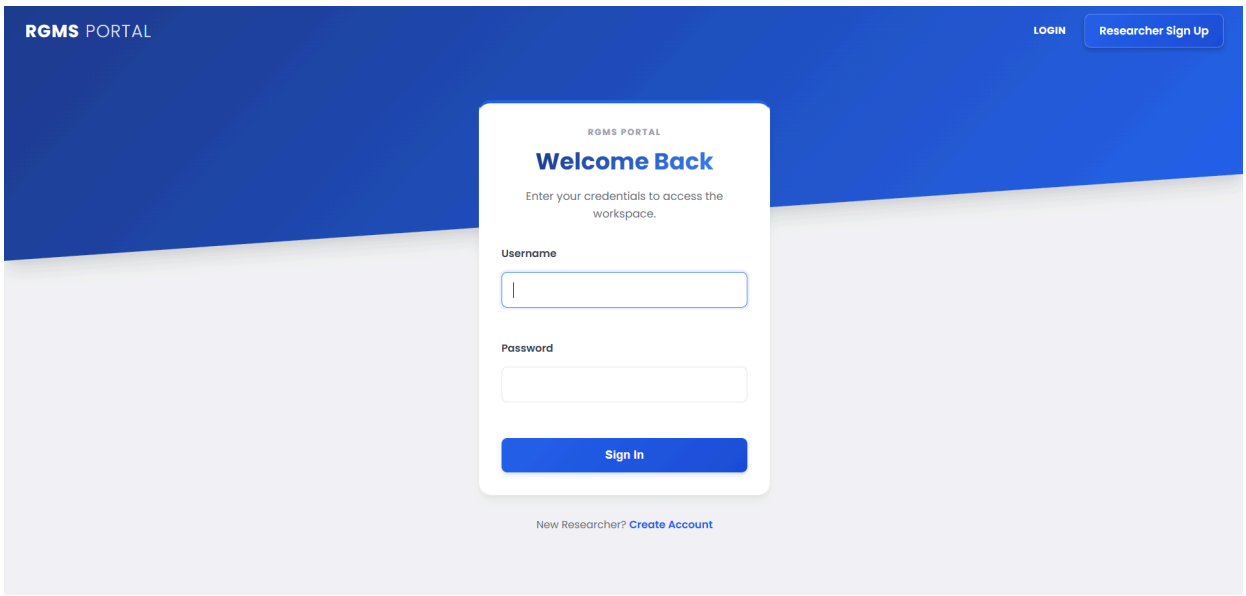
6 Interface Design

6.1 Main Screens

The image below shows the **landing page** all users will first land on. From here users can either navigate to the login **portal** or **sign up** as **researcher**.



The image below shows the **login page** for all users. Users who already have an account can login from here while users who have not signed up, can proceed to **sign up**.

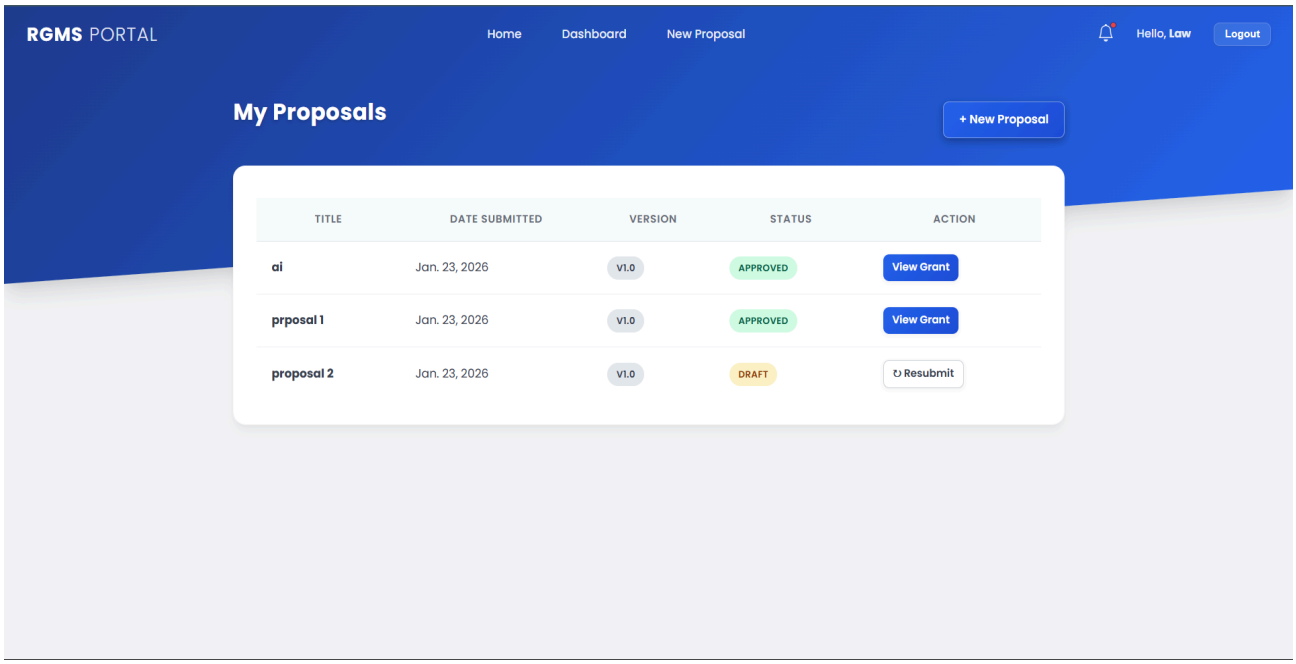


6.2 Subsystem 1: Researcher Subsystem

6.2.1 Researcher Dashboard

Researcher Dashboard The primary landing page that aggregates the Researcher's activities. The interface features a prominent Hero section with a skewed geometric blue background.

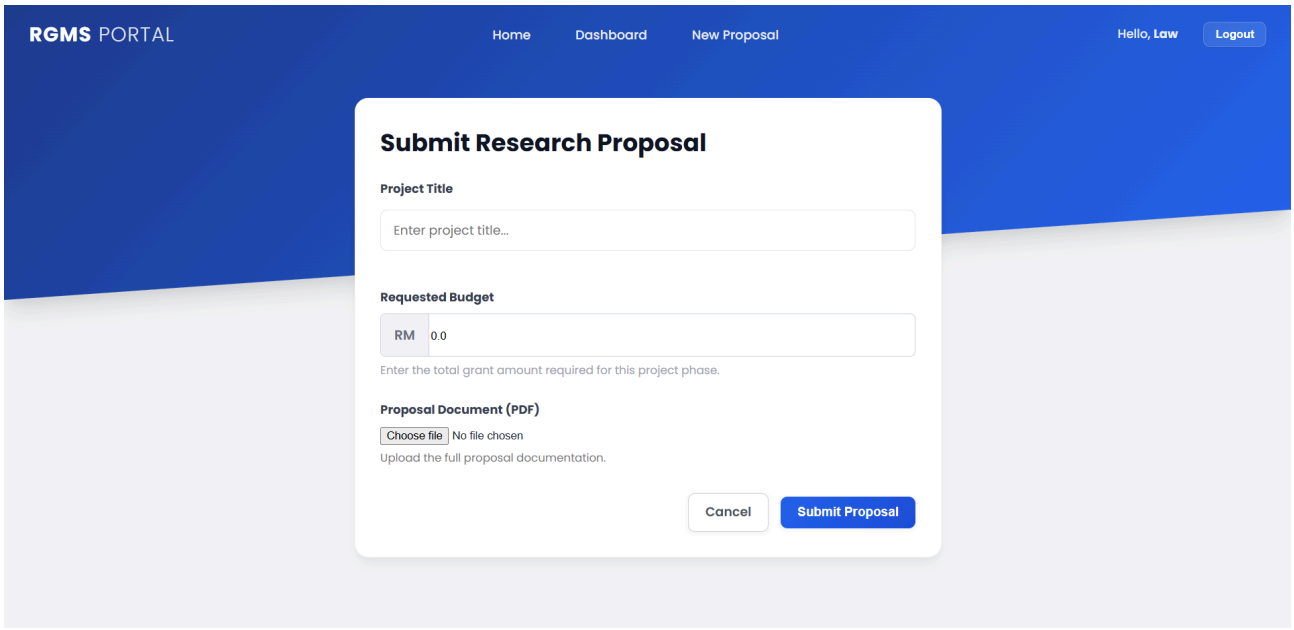
- **KPI Grid:** Uses a 4-column CSS grid (.info-grid) to display high-level metrics (e.g., Active Grants, Pending Proposals).
- **Summary Table:** A clean, spaced table layout listing recent proposals with color-coded status badges (.badge-success for Approved, .badge-warning for Pending).



6.2.2 Submit Proposal or Resubmit Proposal

Submit Proposal Form A standardized data-entry interface designed for minimal friction.

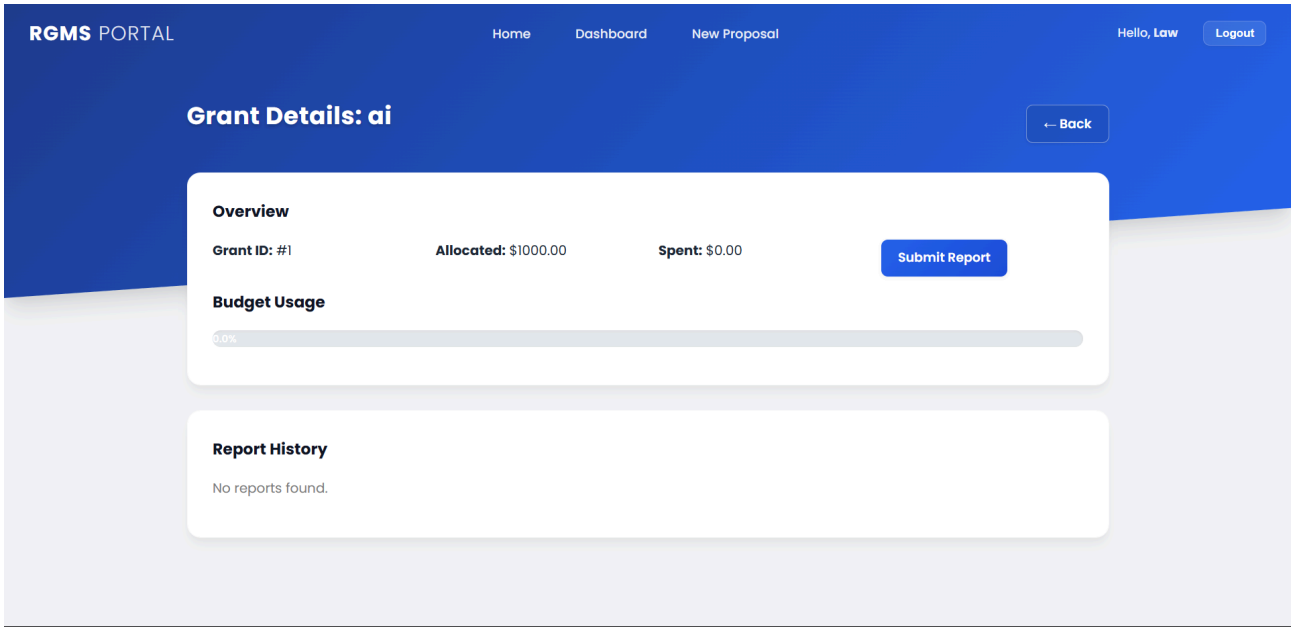
- **Form Layout**: Clean form groups with subtle focus-state borders that guide the user's attention.
- **Document Upload:** Includes a dedicated file input field for the Research Proposal PDF.
- **Action Prompts**: A prominent, gradient-styled Primary Button confirms the submission.



6.2.3 Grant Details & Financial View

Grant Details & Financial View A read-only information hub for an approved grant. This interface uses the custom .card component to organize complex data.

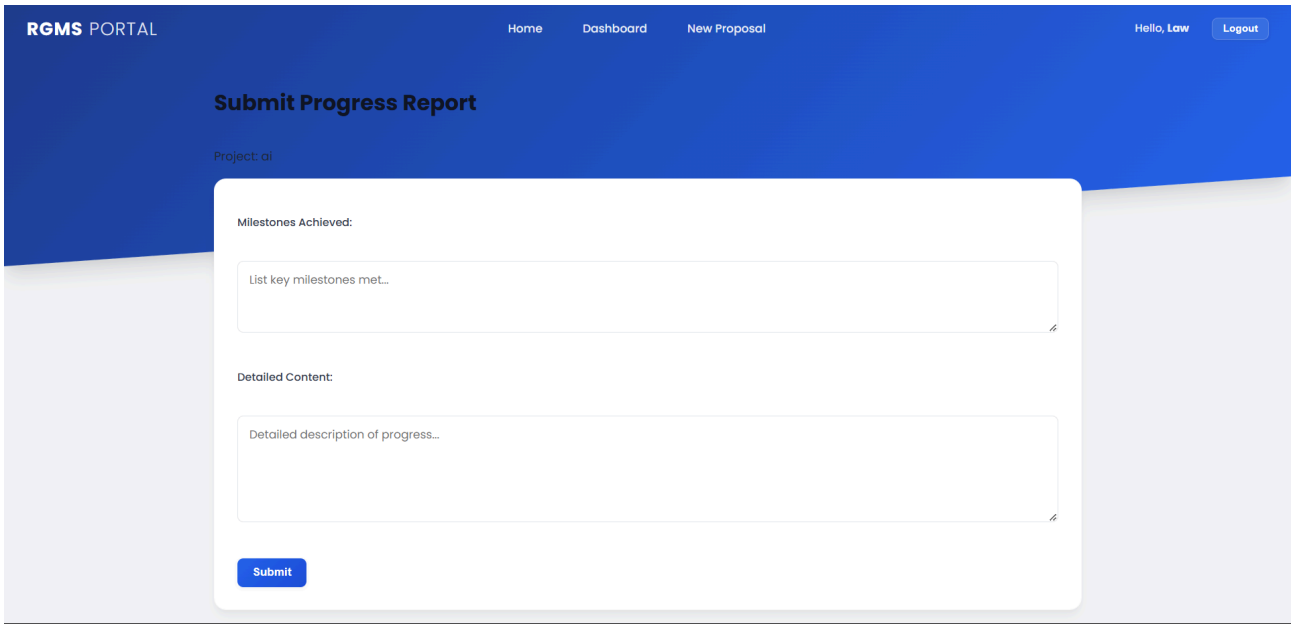
- **Budget Tracker:** Integrates native HTML5/CSS progress bars (.progress-fill) that dynamically display budget usage. Bars change color (Green, Yellow, Red) based on expenditure thresholds.
- **Expenditure Log:** A tabular view detailing date, item description, and receipt proofs for individual expenses.



6.2.4 Submit Progress Report

Submit Progress Report Form An interface tailored for qualitative data collection, allowing the researcher to update the department on their research milestones.

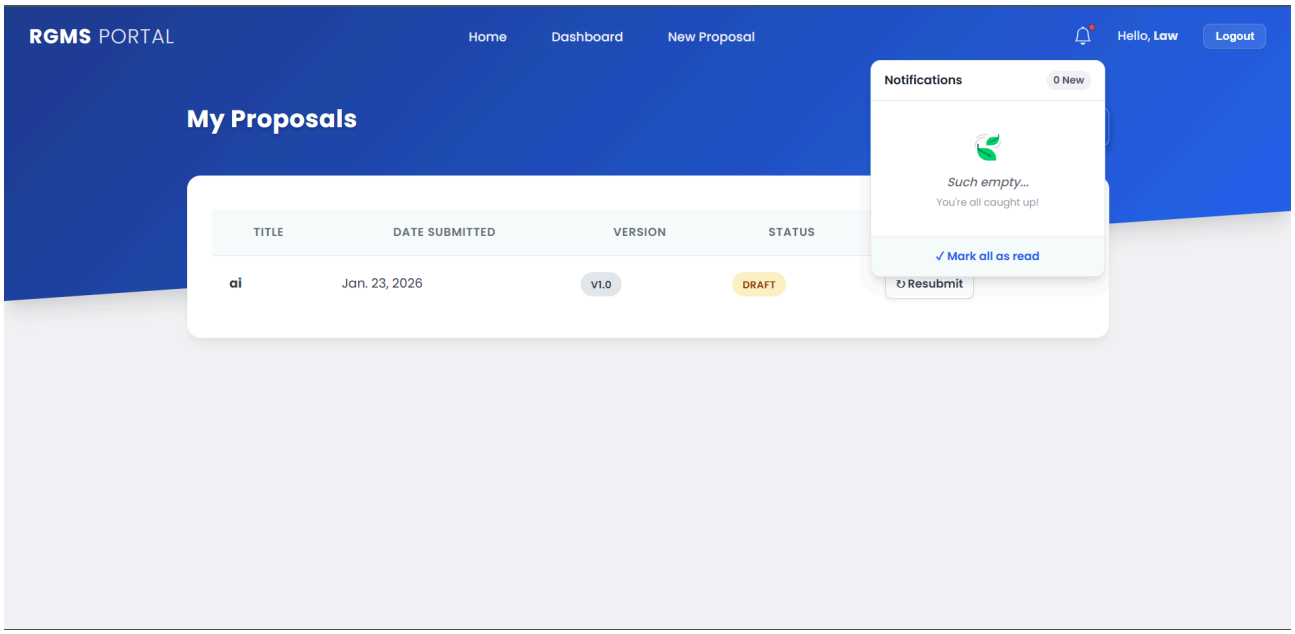
- **Narrative Input:** Features large, resizable text areas (.custom-textarea) for detailed milestone descriptions.
- **Context Preservation:** Displays the Grant Title and current timeline at the top of the card so the researcher retains project context while writing.



6.2.5 Notification System

Notification System Integrated directly into the navigation bar, the Notification Bell acts as the central hub for grant updates (e.g., "Proposal Approved", "Report Reviewed").

- **Visual Cue:** A custom red indicator dot (notif-badge) alerts the researcher to unread system messages.
- **Interactive Dropdown:** Clicking the bell reveals a floating, card-styled menu powered by pure CSS animations.
- **Empty State:** When no notifications are present, the system displays an illustrated "Such empty... 🍃" state, reducing visual clutter and confirming to the user that they are caught up.



6.3 Subsystem 2: Reviewer Subsystem

Interface Description: Reviewer Dashboard

Screen Name: Reviewer Dashboard

User Role: Reviewer

Purpose: To provide a streamlined list of all research proposals assigned to the reviewer for evaluation.

Key Interface Components:

1. **Hero Header:** A blue-themed header featuring the "RGMS Portal" branding and a personalized welcome message (e.g., "Welcome, Ijf") to confirm the user session.

2. **Proposal Evaluation Table:**

   Data Columns: Displays essential metadata including Proposal ID, Title, and Researcher Name.

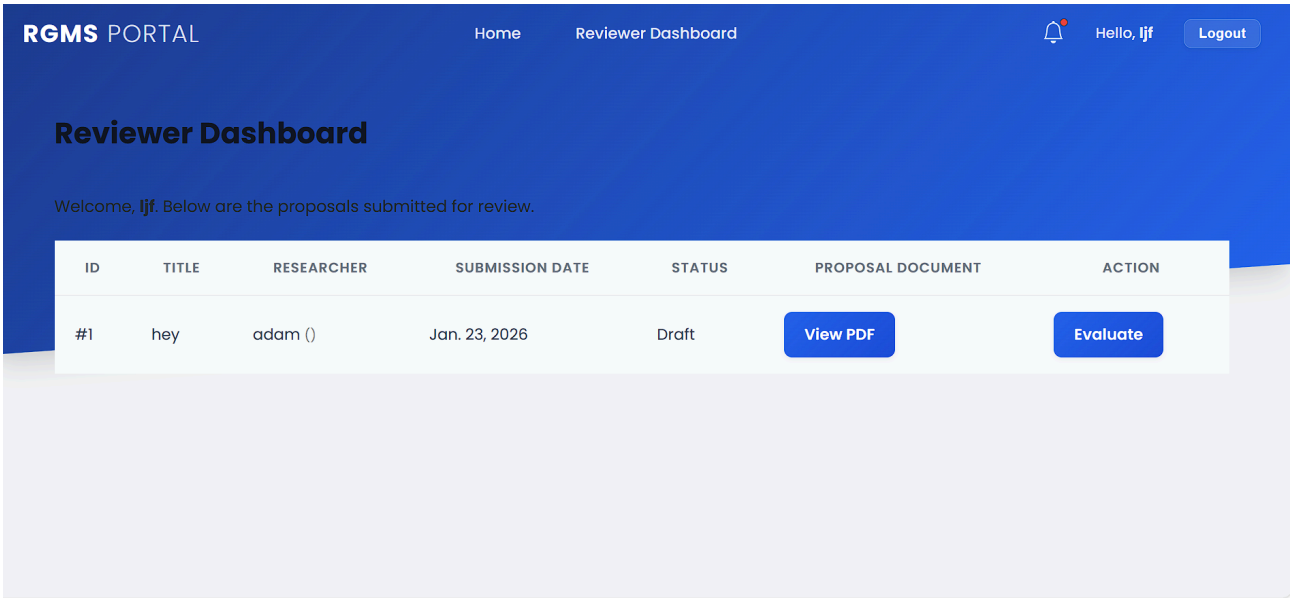
   Submission Tracking: Shows the Submission Date (e.g., Jan 23, 2026) and current Status (e.g., "Draft") to help reviewers prioritize their workload.

3. **Action Hub:**

   View PDF: A primary blue button that allows the reviewer to open and read the proposal document without leaving the application environment.

   Evaluate: A secondary action button that triggers the transition to the formal assessment form

4. **Empty State Logic:** When no tasks are assigned, the table displays a "No proposals available for review" message, preventing UI clutter.



Interface Description: Proposal Evaluation Form

Screen Name: Evaluate Proposal Screen

User Role: Reviewer

Purpose: To capture the reviewer's quantitative score and qualitative feedback for a specific research submission.

Key Interface Components:

1. **Contextual Header:** The screen title dynamically updates to include the proposal title (e.g., "Evaluate Proposal: hey"), ensuring the reviewer knows exactly which project they are scoring.

2. **Researcher Attribution:** Clearly displays the "Submitted by" field to maintain transparency during the review process.

3. **Assessment Inputs:**

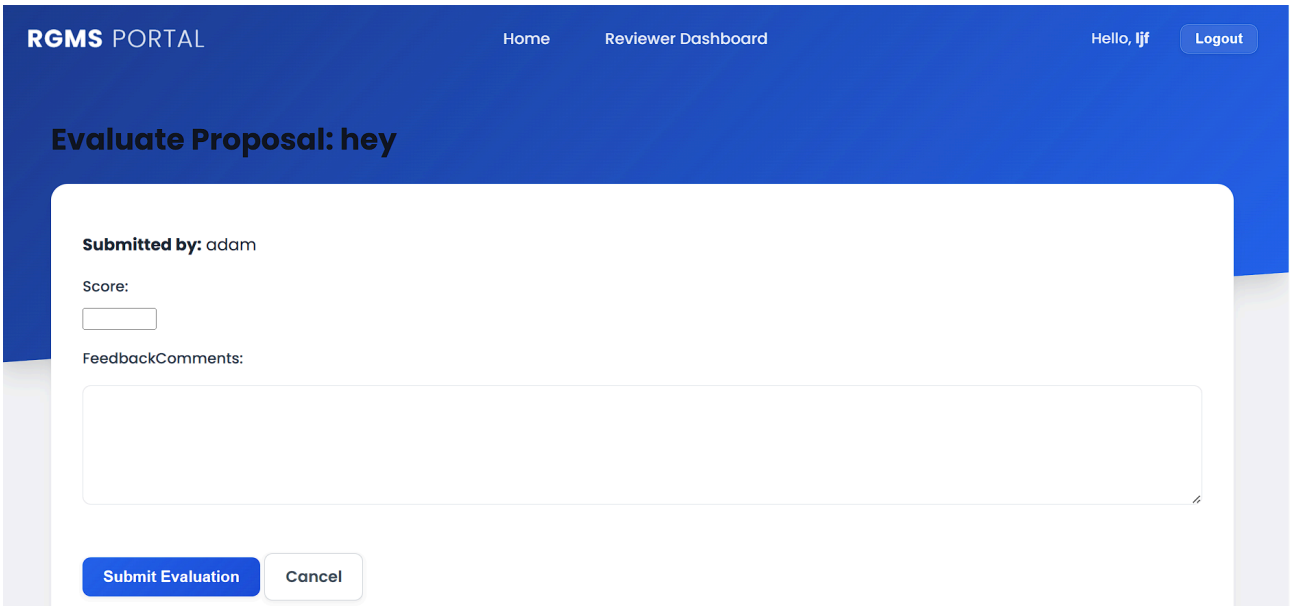
   Score Field: A dedicated numeric input box for the reviewer to provide a quantitative rating based on institutional rubrics.

   Feedback/Comments Box: A large, expandable textarea labeled "FeedbackComments" designed for detailed peer-review notes, critiques, and suggestions for the researcher.

4. **Form Submission Controls:**

   Submit Evaluation: A high-visibility blue button to finalize the review and commit the data to the system.

   Cancel: A neutral button that allows the user to exit the form and return to the dashboard without saving changes, serving as a safety mechanism against accidental edits.



6.4 Subsystem 3 Screens 9 (HOD)

6.4.1 Interface Description: HOD Management Dashboard

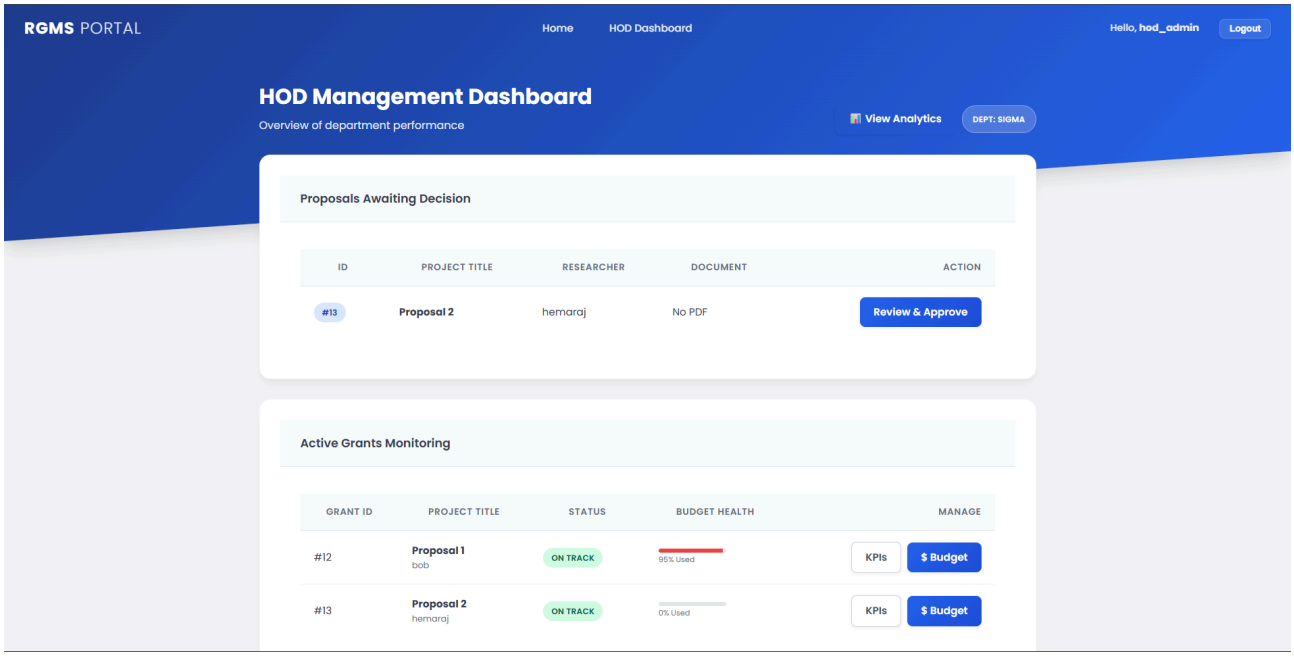
Screen Name: HOD Dashboard

User Role: Head of Department (HOD)

Purpose: To provide a centralized overview of departmental research activities, allowing the HOD to manage incoming proposals and monitor the financial and operational health of active grants.

Key Interface Components:

1. **Hero Header & Global Controls:**
    - **Department Identity:** The top section clearly identifies the user's jurisdiction with a specific badge (e.g., "DEPT: SIGMA"), ensuring the HOD knows which department's data is being viewed.
    - **Analytics Access:** A prominent **"View Analytics"** button is positioned in the header, providing single-click access to the graphical reports (Budget Utilization Doughnut Chart & Success Rate Bar Chart).
    - **Navigation:** Consistent top-bar navigation allows quick switching between Home and the Dashboard, with a user profile and logout option.
2. **Pending Proposals Section (Action Queue):**
    - This card displays a tabular list of **"Proposals Awaiting Decision"**.
    - **Data Columns:** Displays critical info including Proposal ID , Project Title , Researcher Name , and Document availability.
    - **Primary Interaction:** The **"Review & Approve"** button directs the HOD to the detailed approval form for that specific proposal.
3. **Active Grants Monitoring (Project Oversight):**
    - A comprehensive table tracking **"Active Grants Monitoring"** for projects that have already been approved.
    - **Status Indicators:** Uses color-coded badges (e.g., Green for "ON TRACK") to give an immediate visual status of project health.
    - **Visual Budget Health:** Features a dynamic **Progress Bar** under the "Budget Health" column.
        - **Logic:** The bar visualizes the percentage of funds used. It automatically changes color to **Red** (Critical) when usage exceeds 90% (as seen in the "95% Used" example), alerting the HOD to potential overspending without needing to click into the details.
    - **Management Actions:** Provides two distinct action buttons for every grant:
        - **KPIs:** To review project milestones and timelines.
        - **$ Budget:** To view the financial ledger and perform fund top-ups



6.4.2 Interface Description: Approve and Grant Allocation

Screen Name: Grant Approval Decision

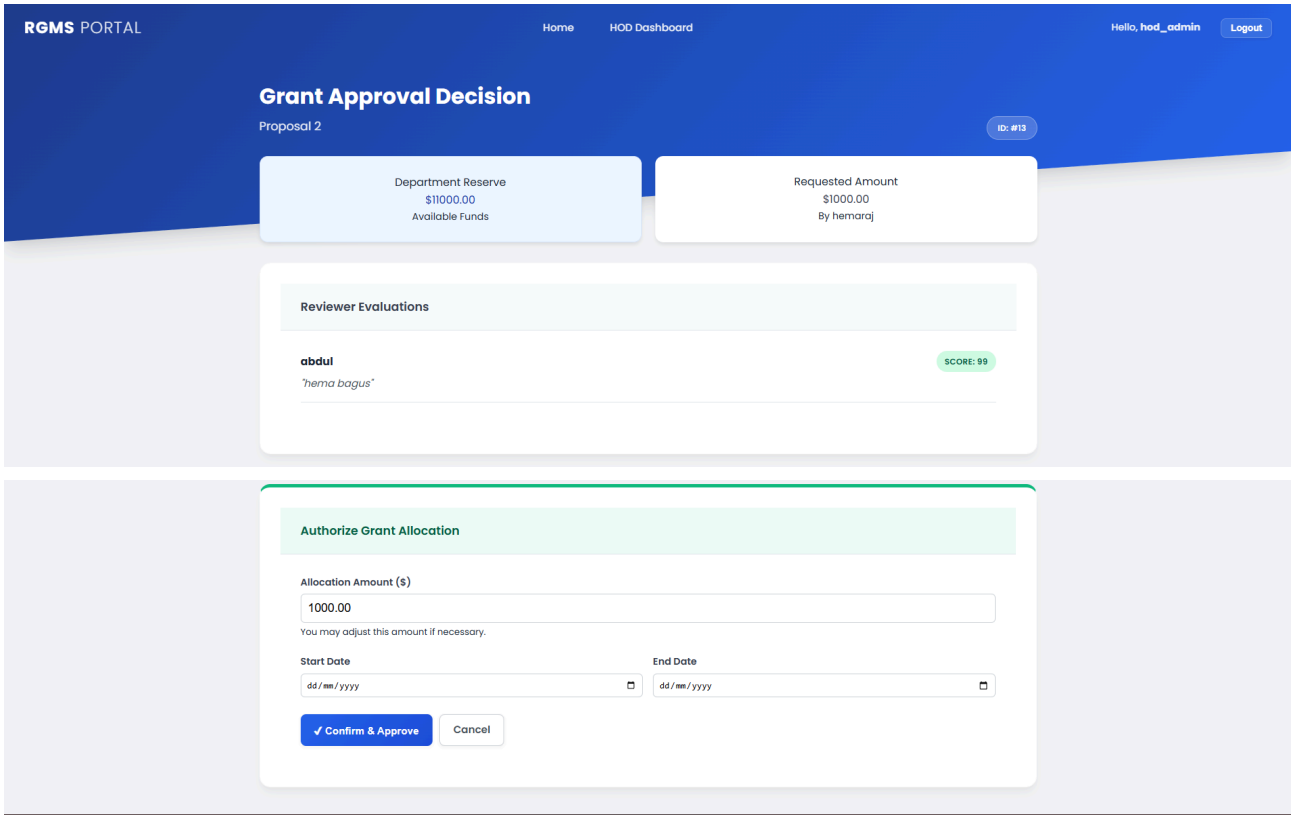
User Role: Head of Department (HOD)

Purpose: To enable the HOD to review reviewer evaluations, assess budget feasibility, and formally authorize the allocation of funds for a specific research proposal.

Key Interface Components:

1. **Financial Context Header (Top Grid):**
    - **Purpose:** To provide immediate financial awareness before a decision is made.
    - **Component:** Two prominent "KPI Cards" displayed side-by-side.
        - **Department Reserve:** Displays the total available funds (e.g., "$11,000.00") in the department's account.
        - **Requested Amount:** Displays the specific amount requested by the researcher (e.g., "$1,000.00").
    - **Design Logic:** Placing these figures adjacent to each other allows the HOD to instantly verify if the department can afford the grant without navigating to a separate budget page.
2. **Evaluation Summary (Middle Section):**
    - **Section Title:** "Reviewer Evaluations".
    - **Content:** Aggregates qualitative and quantitative feedback from the review phase.
    - **Data Points:** Displays the reviewer's username (e.g., "abdul"), their specific comments (e.g., "hema bagus"), and the assigned **Score Badge** (e.g., "SCORE: 99").
    - **Visual Cue:** High scores are highlighted with green badges to quickly signal quality proposals.

3. **Authorization Form (Bottom/Action Section):**
    - **Section Title:** "Authorize Grant Allocation".
    - **Visual Distinction:** This card features a distinct **Green Top Border** to visually signify that this is the "Action Zone" for final approval.
    - **Editable Allocation:** The "Allocation Amount ($)" field is pre-filled with the requested amount ($1000.00) but remains editable, giving the HOD the flexibility to approve partial funding if necessary.
    - **Timeline Controls:** Mandatory "Start Date" and "End Date" pickers to define the grant's active duration.
    - **Primary Action:** A **"Confirm & Approve"** button that commits the transaction and creates the grant record.



6.4.3 Interface Description: Project KPI & Monitoring Review

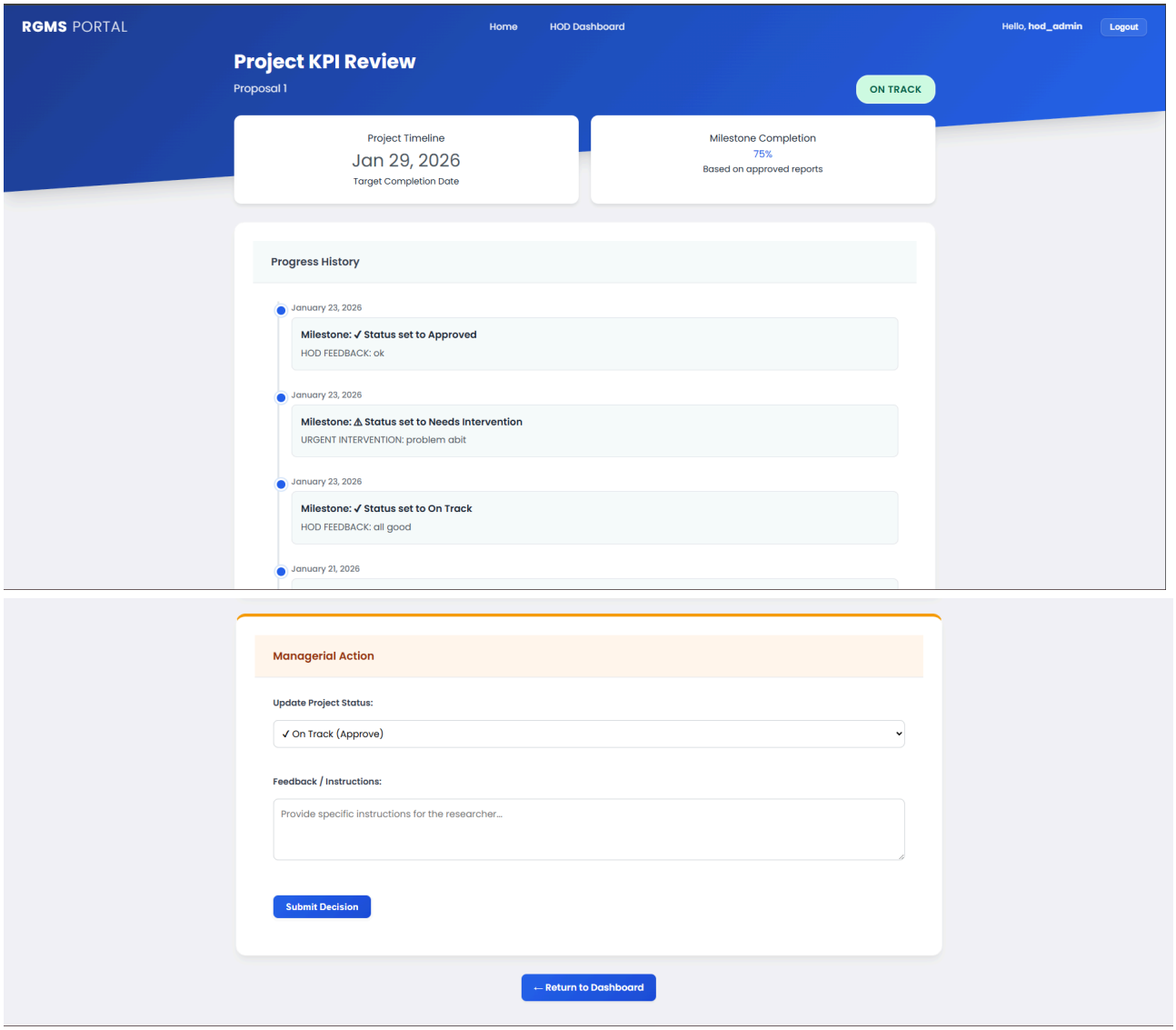
Screen Name: Project Monitoring Detail (project_monitoring_detail.html)

User Role: Head of Department (HOD)

Purpose: To provide a detailed historical and real-time view of a specific grant's progress, allowing the HOD to review milestone achievements and issue managerial directives.

Key Interface Components:

1. **Project Status Header:**
    - **Project Identity:** Clearly displays the project title (e.g., "Proposal 1") and the current status badge (e.g., "ON TRACK").
    - **Visual Context:** Uses the standard HOD blue hero header for consistency, ensuring the user knows they are in a management view.
2. **Performance Metrics (KPI Grid):**
    - **Timeline Tracking:** Displays the "Project Timeline" with the target completion date (e.g., "Jan 29, 2026"), helping the HOD assess if the researcher is behind schedule.
    - **Completion Metric:** A large numeric indicator for "Milestone Completion" (e.g., "75%"). This provides an instant quantitative measure of progress based on approved reports.
3. **Progress History (Vertical Timeline):**
    - **Component:** A chronological, vertical timeline on the left side of the screen.
    - **Data Points:** Each entry acts as a historical log, displaying:
        - **Date:** When the report/action occurred (e.g., "January 23, 2026").
        - **Event:** The specific milestone or status change (e.g., "Milestone: Status set to Needs Intervention").
        - **Feedback/Context:** Displays the content of the report or the HOD's previous feedback (e.g., "URGENT INTERVENTION: problem abit").
    - **Design Logic:** This timeline format allows the HOD to trace the "story" of the project—seeing how it went from "On Track" to "Needs Intervention" and back again.
4. **Managerial Action Panel (Intervention Form):**
    - **Purpose:** To give the HOD direct control over the project's lifecycle.
    - **Visual Distinction:** This card uses an **Orange/Amber Header** ("Managerial Action") to differentiate it from the informational cards, signaling that this is a control panel.
    - **Status Control:** A dropdown menu allows the HOD to force a status update (e.g., "✔ On Track (Approve)" or "⚠ Needs Intervention").
    - **Feedback Loop:** A text area for "Feedback/Instructions" enables the HOD to send specific directives directly to the researcher.
    - **Primary Action:** The "Submit Decision" button commits these changes to the system.


RGMS PORTAL
Home
HOD Dashboard
Hello, hod_admin
Logout
Project KPI Review
Proposal 1
ON TRACK
Project Timeline
Jan 29, 2026
Target Completion Date
Milestone Completion
75%
Based on approved reports
Progress History
January 23, 2026
Milestone: ✓ Status set to Approved
HOD FEEDBACK: ok
January 23, 2026
Milestone: ⚠ Status set to Needs Intervention
URGENT INTERVENTION: problem abit
January 23, 2026
Milestone: ✓ Status set to On Track
HOD FEEDBACK: all good
January 21, 2026
Managerial Action
Update Project Status:
✓ On Track (Approve)
Feedback / Instructions:
Provide specific instructions for the researcher...
Submit Decision
← Return to Dashboard

6.4.4 Interface Description: Project KPI & Monitoring Review

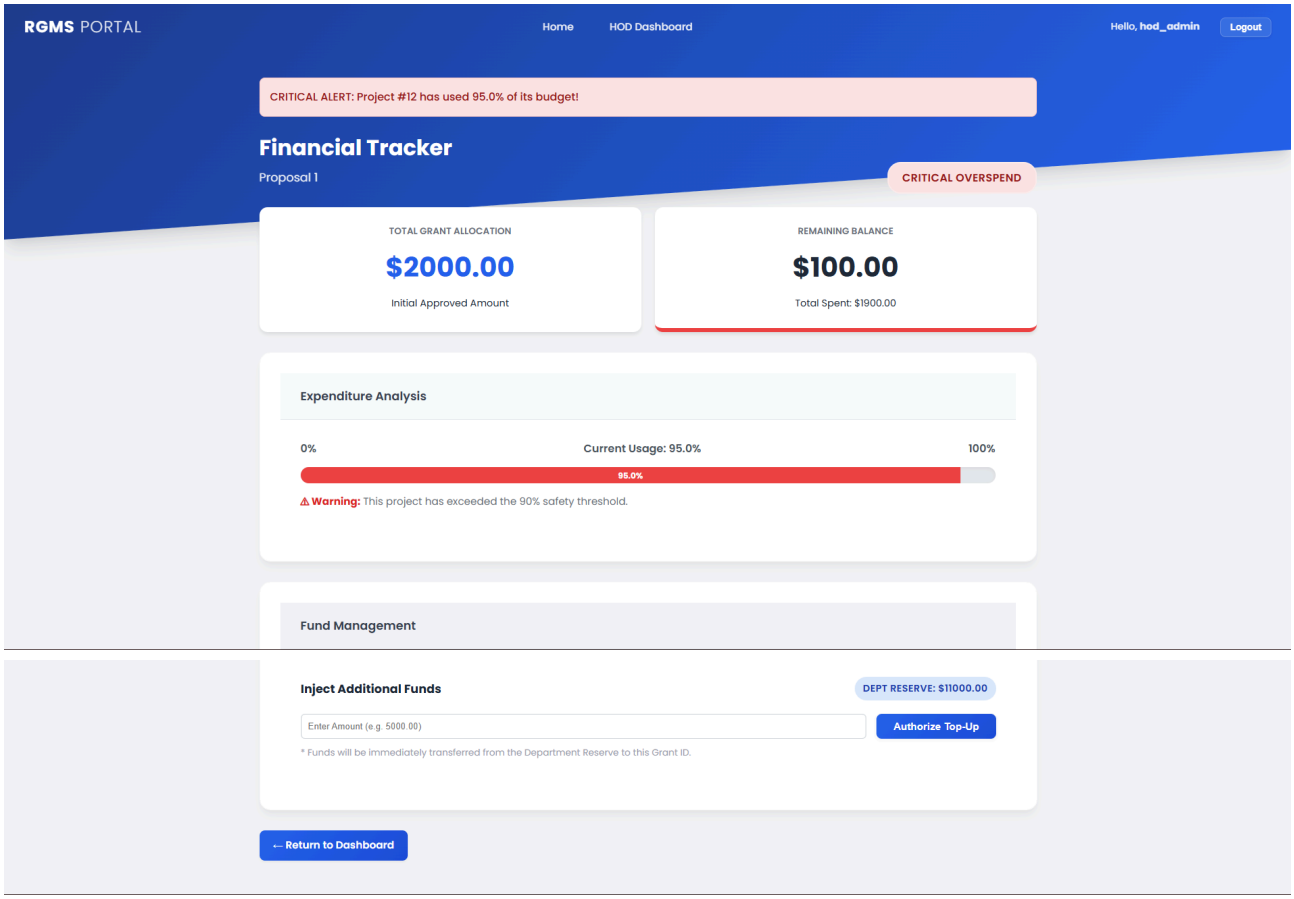
Screen Name: Budget Detail View

User Role: Head of Department (HOD)

Purpose: To provide granular oversight of a specific grant's financial health, alert the HOD to critical overspending, and enable emergency funding interventions (top-ups).

Key Interface Components:

1. **Critical Alert System:**
    - **Banner Alert:** If a project exceeds the 90% spending threshold, a prominent **Red Alert Banner** appears at the top ("CRITICAL ALERT: Project #12 has used 95.0% of its budget!").
    - **Visual Status:** A "CRITICAL OVERSPEND" badge in the header instantly signals that this project is in a danger zone.
2. **Financial KPI Cards (Grid Layout):**
    - **Total Grant Allocation:** Displays the original approved amount (e.g., "$2000.00").
    - **Remaining Balance:** Displays the live funds left (e.g., "$100.00").
    - **Visual Logic:** The "Remaining Balance" card dynamically applies a **Red Bottom Border** (class: border-critical) when funds are low, reinforcing the urgency visually.
3. **Expenditure Analysis (Progress Tracking):**
    - **Component:** A visual progress bar tracking "Current Usage" against a 0% - 100% scale.
    - **Logic:** The bar turns **Solid Red** when usage exceeds 90% (e.g., "95.0%").
    - **Diagnostic Text:** A warning message below the bar explicitly states the rule violation: "⚠ Warning: This project has exceeded the 90% safety threshold".
4. **Fund Management Panel (Action Zone):**
    - **Context:** Displays the "Dept Reserve" balance (e.g., "$11,000.00") so the HOD knows how much money is available to give.
    - **Input Control:** A precise numeric input field ("Enter Amount") allows the HOD to specify the exact top-up value.
    - **Primary Action:** The **"Authorize Top-Up"** button executes the transfer immediately.
    - **Disclaimer:** Clear instructional text confirms that funds will be deducted from the Department Reserve.


RGMS PORTAL
Home
HOD Dashboard
Hello, hod_admin
Logout
CRITICAL ALERT: Project #12 has used 95.0% of its budget!
Financial Tracker
Proposal 1
CRITICAL OVERSPEND
TOTAL GRANT ALLOCATION
$2000.00
Initial Approved Amount
REMAINING BALANCE
$100.00
Total Spent: $1900.00
Expenditure Analysis
0%
Current Usage: 95.0%
100%
95.0%
⚠ Warning: This project has exceeded the 90% safety threshold.
Fund Management
Inject Additional Funds
DEPT RESERVE: $11000.00
Enter Amount (e.g. 5000.00)
Authorize Top-Up
* Funds will be immediately transferred from the Department Reserve to this Grant ID.
← Return to Dashboard

6.4.5 Interface Description: Department Analytics & Performance Reports

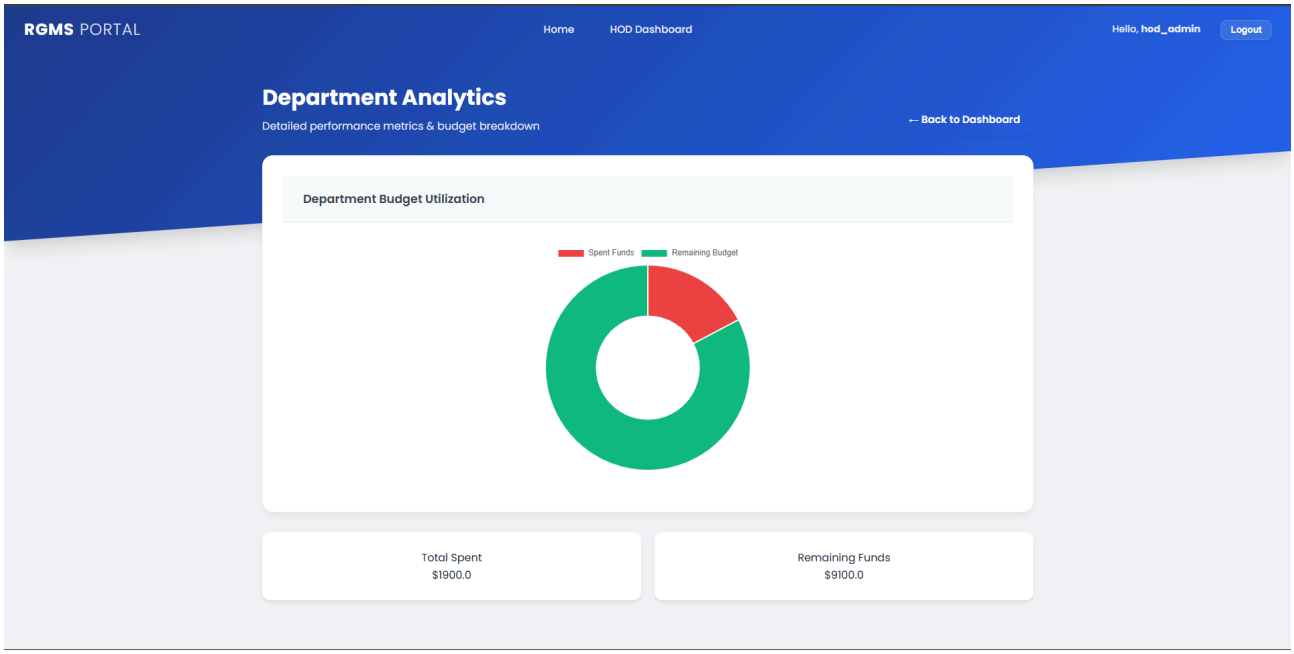
Screen Name: Department Analytics View

User Role: Head of Department (HOD)

Purpose: To provide high-level visual intelligence regarding the department's financial efficiency and proposal success rates, allowing for data-driven strategic planning.

Key Interface Components:

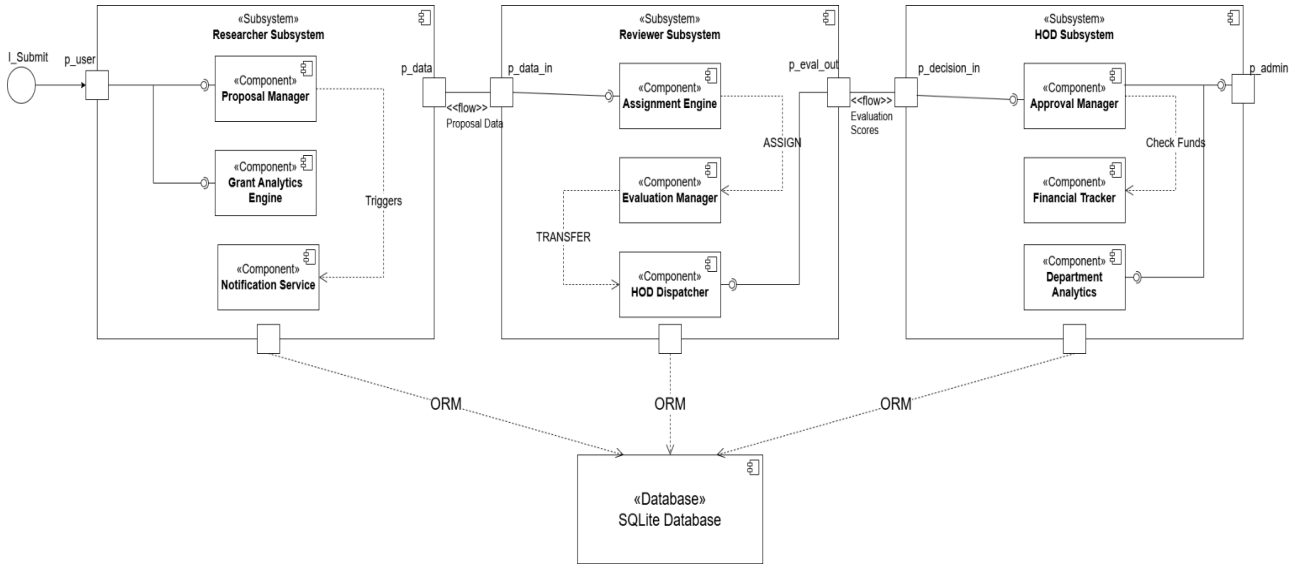
1. **Navigational Header:**
    - **Context:** The header clearly labels the page "Department Analytics" with a subtitle "Detailed performance metrics & budget breakdown," setting the expectation for statistical content.
    - **Back Navigation:** A prominent "← Back to Dashboard" button allows the HOD to quickly return to the main operational view after reviewing the data.
2. **Visual Analytics (Charts Section):**
    - **Budget Utilization Chart:** A Doughnut Chart visualizing the ratio of **"Spent Funds" (Red)** vs. **"Remaining Budget" (Green)**.
        - **Design Rationale:** The use of red/green color coding provides an immediate "traffic light" assessment of financial health. A large red segment warns of budget depletion at a glance.
    - **Proposal Success Metrics:** (Implicit in layout/code, though partially cut off in this specific crop) A Bar Chart comparing approved versus rejected proposals to track department research output quality.
3. **Summary KPI Cards (Bottom Grid):**
    - **Total Spent:** A dedicated card displaying the aggregate expenditure across all active grants (e.g., "$1900.0"). The text is color-coded **Red** to signify money flowing out.
    - **Remaining Funds:** A card displaying the available balance (e.g., "$9100.0"). The text is color-coded **Green** to signify available resources.


RGMS PORTAL
Home
HOD Dashboard
Hello, hod_admin
Logout
Department Analytics
Detailed performance metrics & budget breakdown
← Back to Dashboard
Department Budget Utilization
Spent Funds
Remaining Budget
Total Spent
$1900.0
Remaining Funds
$9100.0

...

7 Component Design

7.1 Component Diagram



1. Description of Main Components

The Research Grant Management System (RGMS) is structured into three distinct subsystems (Packages) that communicate via a shared repository architecture (SQLite). Each subsystem encapsulates specific business logic modules.

A. Researcher Subsystem

This package handles the initiation of the grant lifecycle.

- **Proposal Manager:** The core logic module responsible for the creation, editing, and submission of grant proposals. It handles input validation and ensures data integrity before persistence.
- **Grant Analytics Engine:** A read-only module that queries the database to visualize the status of submitted proposals and track milestones for the researcher.
- **Notification Service:** An internal utility that triggers status alerts (e.g., "Proposal Received") to the user upon successful submission.

B. Reviewer Subsystem

This package serves as the quality assurance layer.

- **Assignment Engine:** A logic module that retrieves unassigned proposals and links them to available reviewers based on workload or expertise.
- **Evaluation Manager:** Handles the input of quantitative scores and qualitative feedback. It implements the grading logic and calculates the final average score.
- **HOD Dispatcher:** A gateway component that packages completed evaluations and flags them as "Ready for Decision," effectively transferring control to the HOD subsystem.

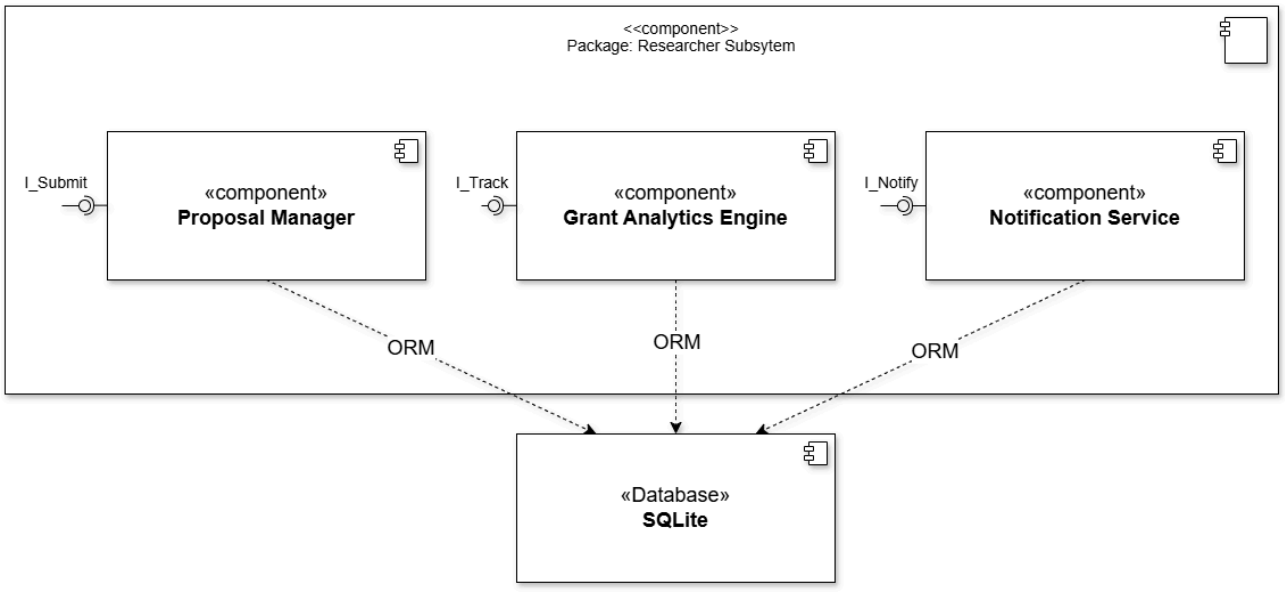
C. HOD Subsystem

This package handles high-level decision-making and financial governance.

- **Approval Manager:** The decision engine that allows the HOD to accept or reject proposals based on the aggregated scores provided by the Reviewer subsystem.
- **Financial Tracker:** A risk-management module that monitors department funds. It enforces the "90% Budget Alert" rule and processes fund top-ups.
- **Department Analytics:** A reporting module that aggregates data to generate visual charts (Doughnut/Bar charts) regarding budget utilization and success rates.
- **Monitoring Engine:** Tracks the operational progress of active grants (e.g., Timeline adherence) after approval.

7.2 Main Component 1 (Subsystem 1: Researcher)

7.2.1 Component Diagram



7.2.2 Main Components

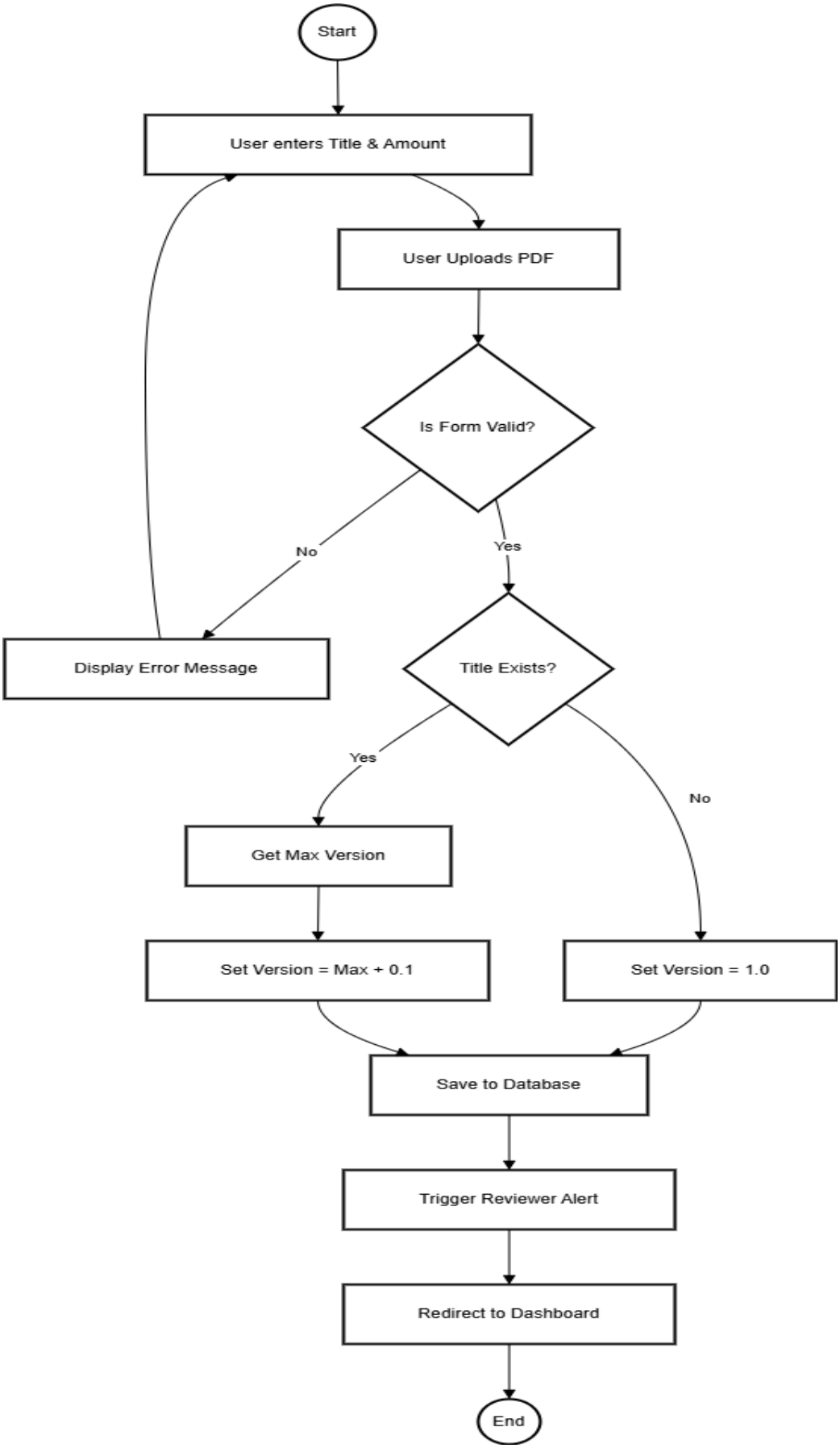
Component Name	*Type*	*Description*	*Use Cases Covered*
Proposal Manager	*Function Module*	*Manages the submission lifecycle. It handles form validation, file uploads (PDFs), and enforces **Version Control** (e.g., v1.0 -> v1.1) to prevent duplicate entries.*	*UC 1 (Upload Proposals)*
Grant Analytics Engine	*Function Module*	*Tracks the financial health of active grants. It calculates the **Utilization Percentage** (Spent / Allocated) and triggers visual alerts (Red/Green) based on the 90% threshold.*	*UC 2 (Track Grant Status)*
Notification Service	*Function Module*	*Acts as the communication bridge. It listens for system events (Approvals/Deadlines) to generate alerts and manages the **Read/Unread** state for the dashboard badge.*	*UC 3 (Receive Notifications)*

7.2.3 Component 1: Proposal Manager

Description: The Proposal Manager is responsible for handling the submission lifecycle. It validates user inputs, manages file uploads (PDFs), and critically, **enforces version control**. It detects if a title already exists and automatically increments the version number (e.g., v1.0 —> v1.1) instead of creating duplicates.

Pseudocode:

```
ALGORITHM submit_proposal(user, title, file, amount):
    1. VALIDATE form inputs (check if title is empty, file is PDF)
    2. IF form is invalid:
       RETURN error_message

    3. SEARCH Database for existing proposals by this 'user' and 'title'

    4. IF existing_proposal FOUND:
       a. FETCH the highest version number (max_version)
       b. SET new_version = max_version + 0.1
       c. SET status = "Pending"
    5. ELSE:
       a. SET new_version = 1.0
       b. SET status = "Draft"

    6. CREATE new Proposal record:
       - Version = new_version
       - File = uploaded_file
       - Status = status

    7. SAVE to Database
    8. TRIGGER Notification to Reviewer
    9. REDIRECT to Dashboard
END ALGORITHM
```

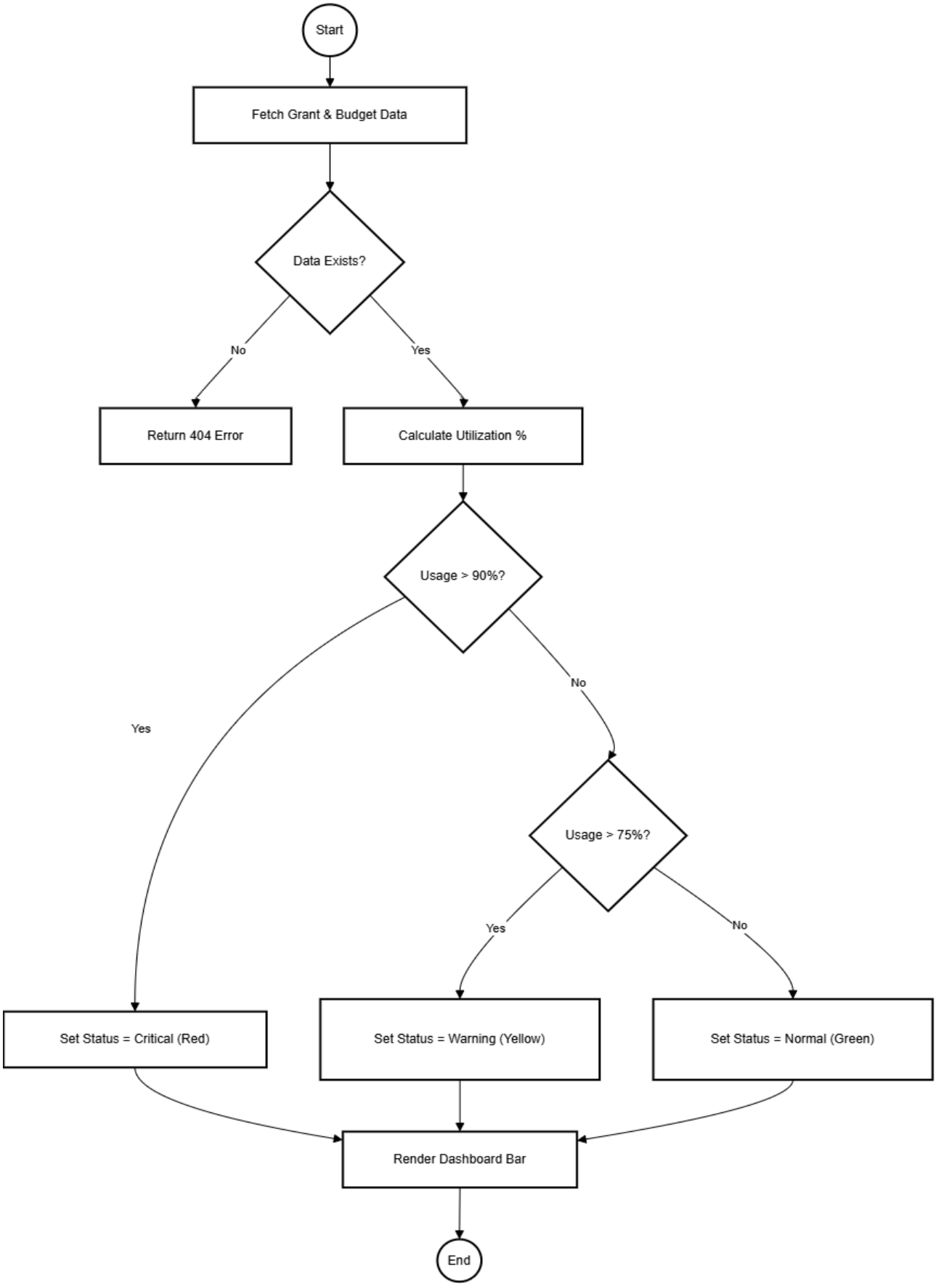

Start
User enters Title & Amount
User Uploads PDF
Is Form Valid?
No
Yes
Display Error Message
Title Exists?
Yes
No
Get Max Version
Set Version = Max + 0.1
Set Version = 1.0
Save to Database
Trigger Reviewer Alert
Redirect to Dashboard
End

7.2.4 Component 2: Grant Analytics Engine

Description: The Grant Analytics Engine is responsible for visualizing the financial health of a project. It fetches the raw financial data (Total Allocated vs. Total Spent), calculates the **Utilization Percentage**, and determines the **Health State** (Normal vs. Critical). It dynamically updates the UI elements (Green vs. Red progress bars) based on these calculations.

Pseudocode:

```
ALGORITHM track_budget(grant_id):
    1. FETCH Grant object WHERE id = grant_id
    2. FETCH Budget object associated with Grant

    3. IF Grant or Budget not found:
       RETURN 404_Error

    4. CALCULATE Utilization:
       percentage = (Budget.totalSpent / Grant.totalAllocated) * 100

    5. DETERMINE Health State:
       IF percentage >= 90:
          SET alert_level = "CRITICAL"
          SET color = "Red"
       ELSE IF percentage >= 75:
          SET alert_level = "WARNING"
          SET color = "Yellow"
       ELSE:
          SET alert_level = "NORMAL"
          SET color = "Green"

    6. RENDER "grant_detail.html" with:
       - usage_percent
       - alert_level
       - progress_bar_color
END ALGORITHM
```

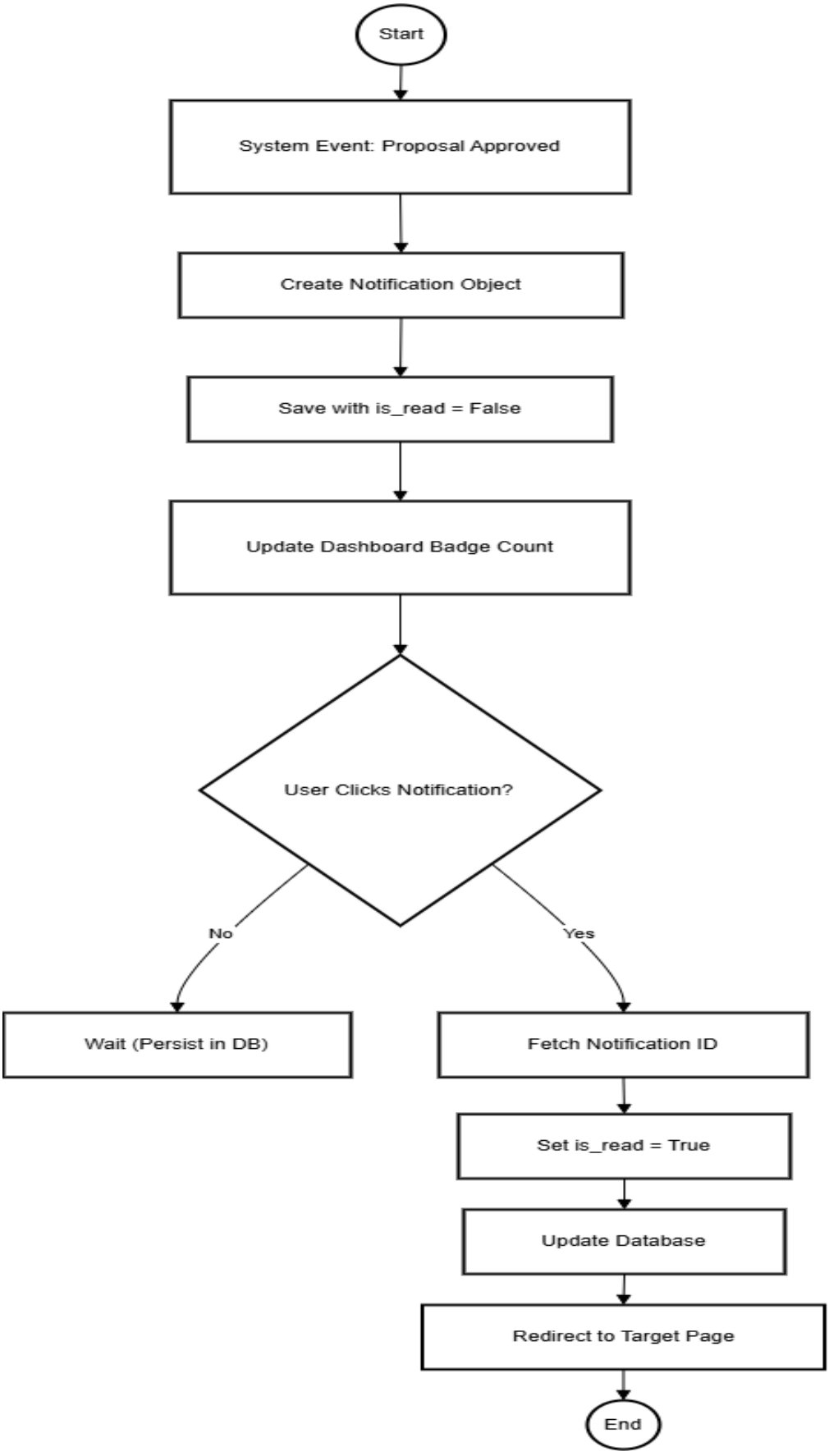

Start
Fetch Grant & Budget Data
Data Exists?
No
Yes
Return 404 Error
Calculate Utilization %
Usage > 90%?
Yes
No
Usage > 75%?
Yes
No
Set Status = Critical (Red)
Set Status = Warning (Yellow)
Set Status = Normal (Green)
Render Dashboard Bar
End

7.2.5 Component 3: Notification Service

Description: The Notification Service acts as the communication bridge. It has two main responsibilities: **Listening** for system events (Signals) to generate alerts, and **Managing** the read/unread state when a user interacts with them. It ensures the "Badge Count" on the dashboard is always accurate.

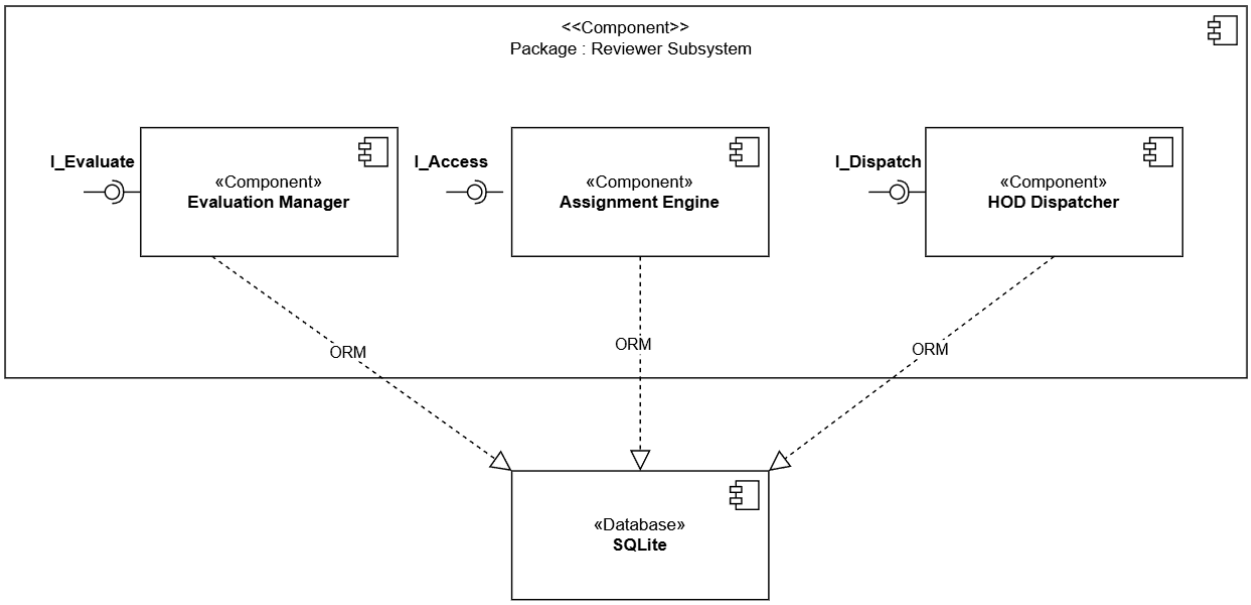
Pseudocode:

```
ALGORITHM mark_as_read(notification_id):
    1. RECEIVE request when user clicks "Bell Icon" or "Message"

    2. FETCH Notification object WHERE id = notification_id

    3. UPDATE object:
       is_read = True
       read_at = Current_Timestamp

    4. SAVE changes to Database

    5. DECREMENT User's Badge Count (Global Context)

    6. EXTRACT target_url from Notification (e.g., link to specific Grant)

    7. REDIRECT User to target_url
END ALGORITHM
```


Start
System Event: Proposal Approved
Create Notification Object
Save with is_read = False
Update Dashboard Badge Count
User Clicks Notification?
No
Yes
Wait (Persist in DB)
Fetch Notification ID
Set is_read = True
Update Database
Redirect to Target Page
End

7.2 Component 2 (Subsystem 2: Reviewer)

7.2.1 Component diagram



7.2.2 Main Components

The Reviewer subsystem is architected into three distinct functional modules located within the grants application. These components handle the assessment lifecycle of submitted proposals, from initial assignment to the systematic evaluation and final hand-off of recommendations to the Head of Department (HOD).

Component Name	Type	Description	Use Cases Covered
Evaluation Manager	Function Module	Processes qualitative and quantitative feedback. It ensures evaluations are saved securely and "locked" once submitted to maintain integrity.	UC 4 (Evaluation/Scoring Proposals)
Assignment Engine	Function Module	Get the global database to present reviewers with their specific assigned proposals. It verifies permissions via the Auth Module before granting access to documents	UC 4 (View researcher documents)
HOD Dispatcher	Function Module	Handles the "Submit	UC 4 (Give Feedback)

		Evaluation" workflow by updating proposal statuses and triggering automated notifications to the HOD.	

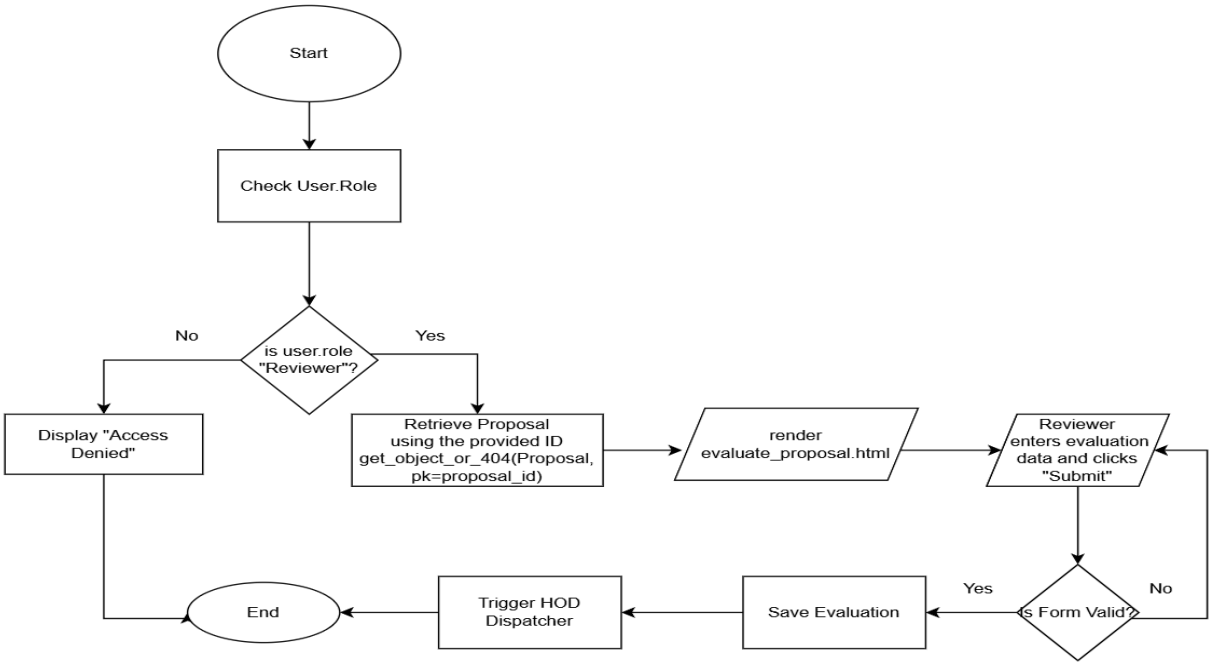
7.2.3 Component 1: Evaluation Manager

Description: This manager is responsible for capturing the reviewer's assessment. It validates that the score and comments meet system constraints before committing the data to the database.

Pseudocode:

ALGORITHM evaluate_proposal(user, proposal_id, score, comments):
1. VERIFY user.role equals "Reviewer"
   ELSE RETURN "Access Denied"
2. RETRIEVE Proposal record from Database using proposal_id
3. VALIDATE form inputs (e.g., score is integer, comments are present)
4. IF form is valid:
   a. CREATE new Evaluation record linked to Proposal and Reviewer
   b. SAVE Evaluation to Database
   c. CALL HOD Dispatcher to update status
5. ELSE:
   a. RENDER evaluation form with validation errors

END ALGORITHM



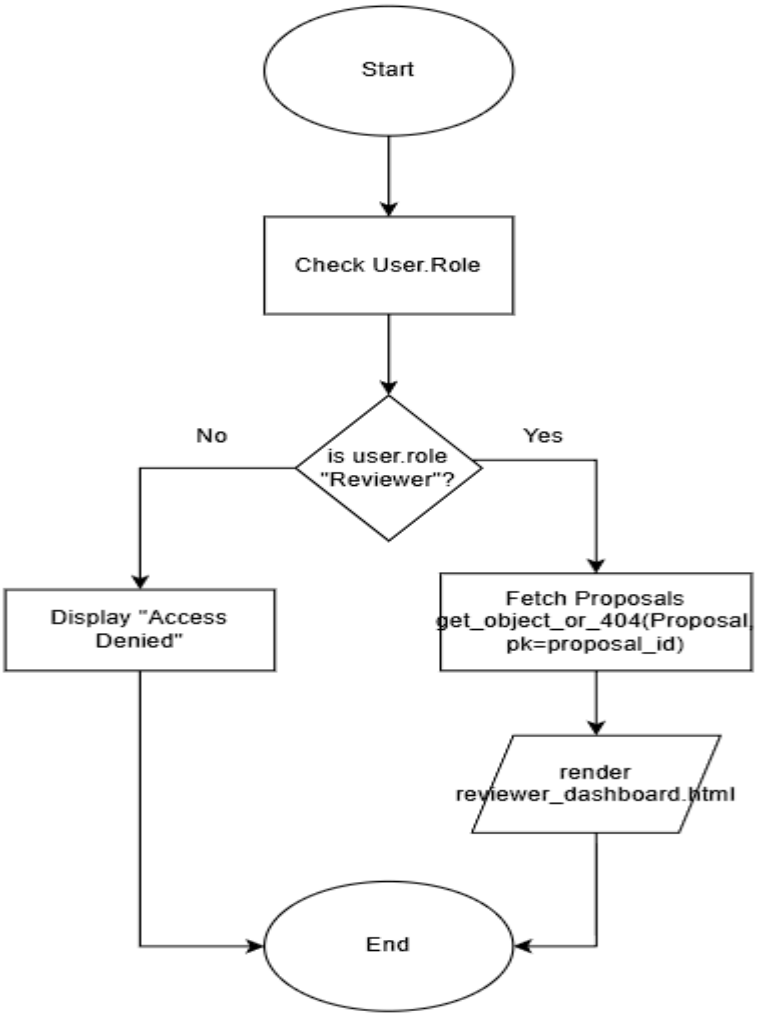
7.2.4 Component 2: Assignment Engine

Description: This module serves as the primary engine for the Reviewer Dashboard. It aggregates and displays all proposals that are ready for review, filtering versions to ensure only valid submissions are assessed.

Pseudocode:

ALGORITHM load_reviewer_dashboard(user):
1. VERIFY user.role equals "Reviewer"
2. RETRIEVE list of all Proposals from Database
3. RENDER "reviewer_dashboard.html" with the filtered proposal list

END ALGORITHM



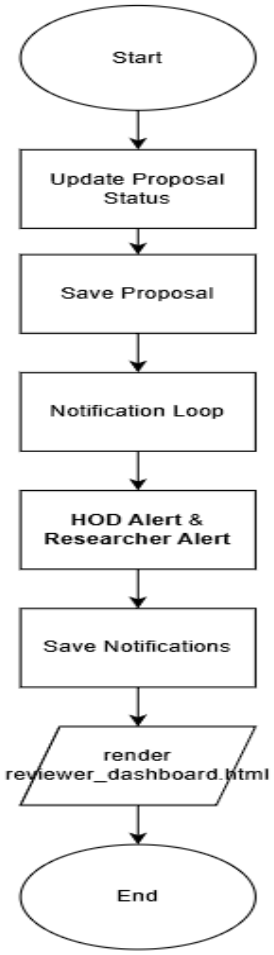
7.2.5 Component 3: HOD Dispatcher

Description: The HOD Dispatcher manages the workflow transition once an evaluation is submitted. It bridges the gap between the Reviewer and HOD roles by updating the proposal state and notifying stakeholders.

Pseudocode:

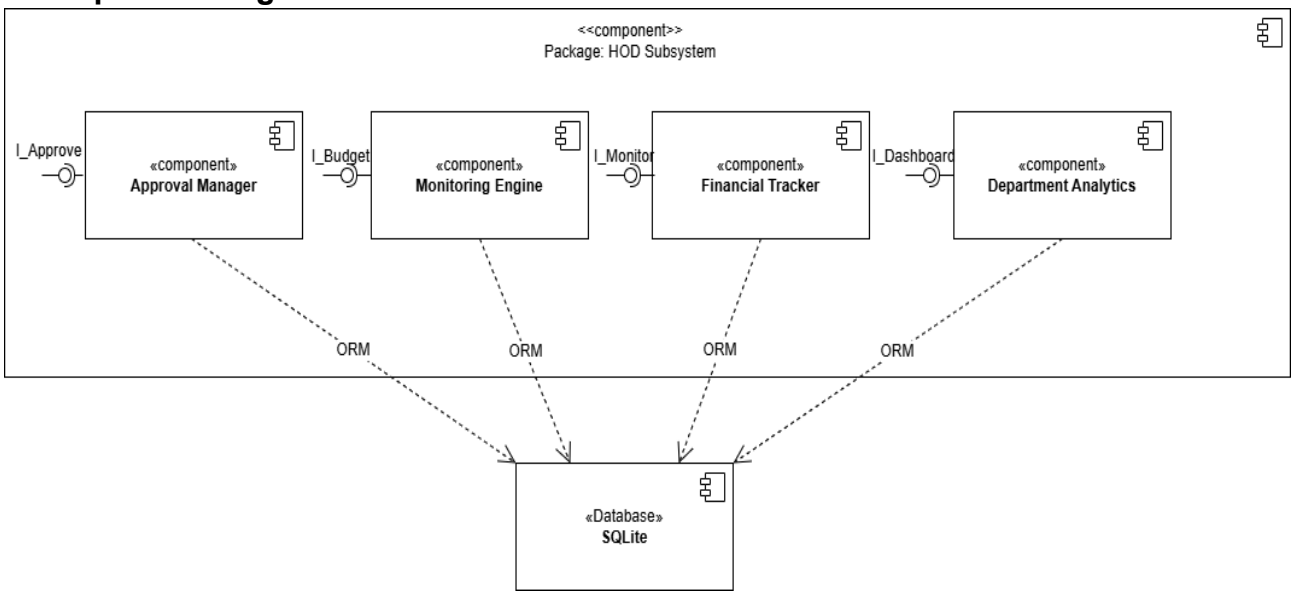
ALGORITHM finalize_review_transition(proposal):
1. UPDATE proposal.status to "Review Complete"
2. SAVE updated proposal record to Database
3. TRIGGER Notifications:
   a. CREATE Notification for HOD ("Ready for Decision")
   b. CREATE Notification for Researcher ("Review Done")
4. RETURN Success and Redirect to Reviewer Dashboard

END ALGORITHM



7.3 Component 3 (Subsystem 3: HOD)

7.3.1 Component diagram



7.3.2 Main Components

The Head of Department (HOD) subsystem is architected into three distinct functional modules located within the grants application. These components handle the lifecycle of a grant from approval to monitoring and financial oversight.

Component Name	Type	Description	Use Cases Covered
Approval Manager	Function Module	Manages the decision process, validates department funds, and initializes the grant entity.	UC 5 (Approve/Reject)
Financial Tracker	Function Module	Tracks real-time expenditures, calculates remaining balances, and processes budget top-ups.	UC 6 (Track Budget)
Monitoring Engine	Function Module	Aggregates project health metrics (KPIs) and retrieves detailed progress reports.	UC 7 (Monitor Research Progress)
Dashboard	Function Module	Serves as the central	UC 8 (Monitor

Manager		analytical hub for the HOD; aggregates departmental data from proposals and grants to calculate real-time performance metrics (KPIs) and provides administrative tools for exporting visual reports.	Department Analytics)

7.3.3 Component 1: Approval Manager

Description: The Approval Manager is the core decision-making module of the HOD subsystem. It handles the transition of a researcher's proposal into an active, funded grant. Its primary responsibility is to act as a financial gatekeeper by validating that the department has sufficient funds before any grant is authorized. It ensures data integrity by linking the grant to the original proposal and initializing the financial tracking records (Budget) simultaneously.

Pseudocode:

ALGORITHM ApproveProposal(user, proposal_id, input_amount, dates)

1. VERIFY user.role equals "HOD"

   ELSE RETURN "Access Denied"

2. RETRIEVE Proposal(proposal_id) AND HOD_User(current_user)

3. IF HTTP Method is POST (Submission):

   a. PARSE allocated_amount from form input

   b. VALIDATE Funds:

      IF allocated_amount > hod_user.department_budget:

         RETURN Error("Insufficient Department Funds")

c. CREATE or UPDATE Grant:

SET grant.proposal = proposal

SET grant.totalAllocatedAmount = allocated_amount

SET grant.dates = input_dates

SAVE Grant

d. IF Grant was newly created:

DEDUCT allocated_amount FROM hod_user.department_budget

CREATE Budget Record (linked to Grant, total_spent = 0.00)

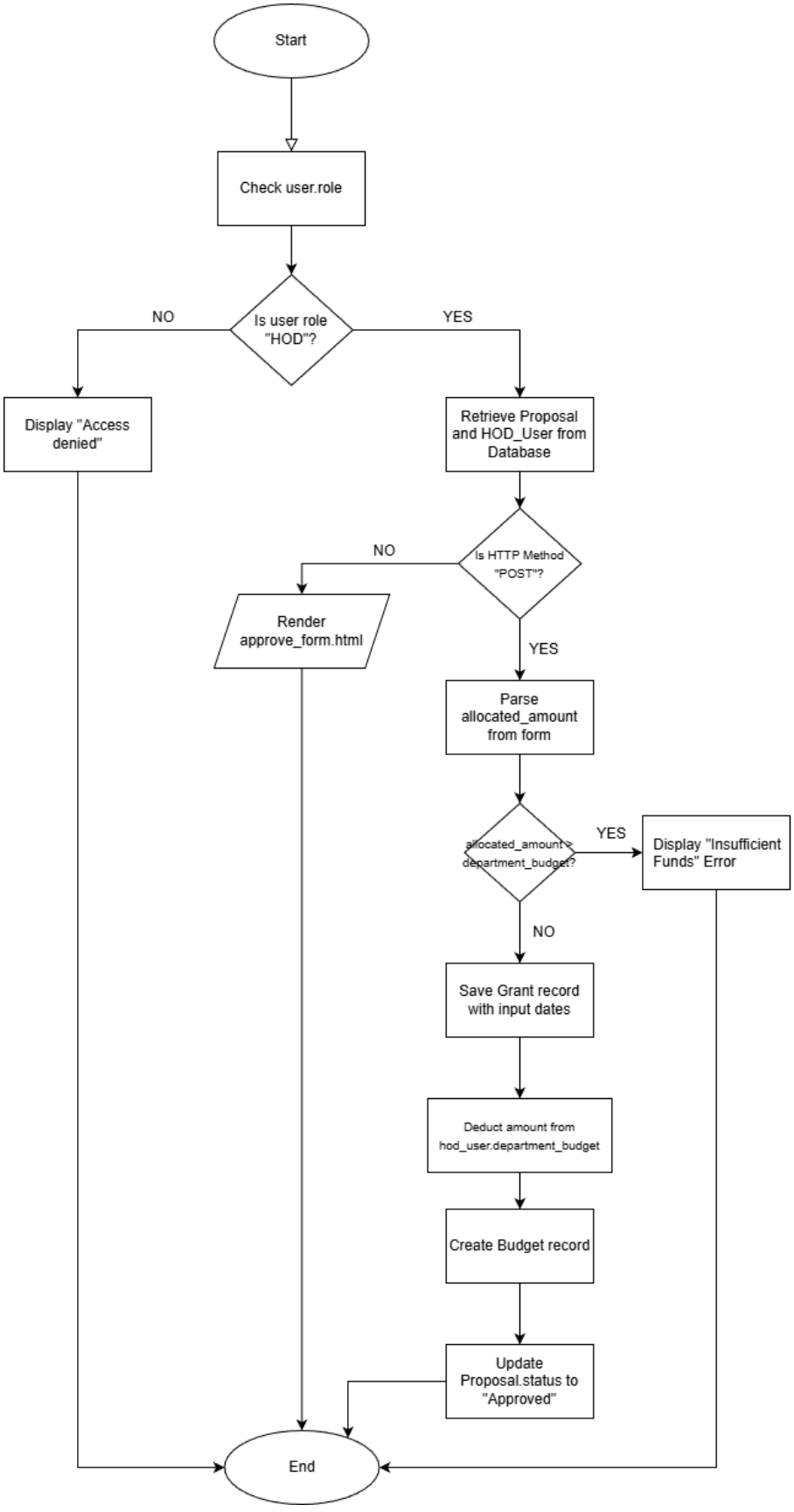
UPDATE Proposal.status = "Approved"

e. RETURN Success Message AND Redirect to Dashboard

4. ELSE (GET Request):

RENDER "approve_form.html" with Proposal details and Current Department Budget

END ALGORITHM


Start
Check user.role
Is user role "HOD"?
NO
YES
Display "Access denied"
Retrieve Proposal and HOD_User from Database
Is HTTP Method "POST"?
NO
YES
Render approve_form.html
Parse allocated_amount from form
allocated_amount > department_budget?
YES
Display "Insufficient Funds" Error
NO
Save Grant record with input dates
Deduct amount from hod_user.department_budget
Create Budget record
Update Proposal.status to "Approved"
End

7.3.4 Component 2: Financial Tracker

Description: The Financial Tracker provides continuous fiscal oversight for active research grants. It calculates real-time metrics, such as the remaining balance and the percentage of funds consumed. A core business rule of this component is the "Critical Threshold" monitor, which triggers a visual alert when project spending exceeds 90% of the total allocation. Additionally, it handles "Top-Up" requests, allowing the HOD to inject further department funds into a project if required.

Pseudocode:

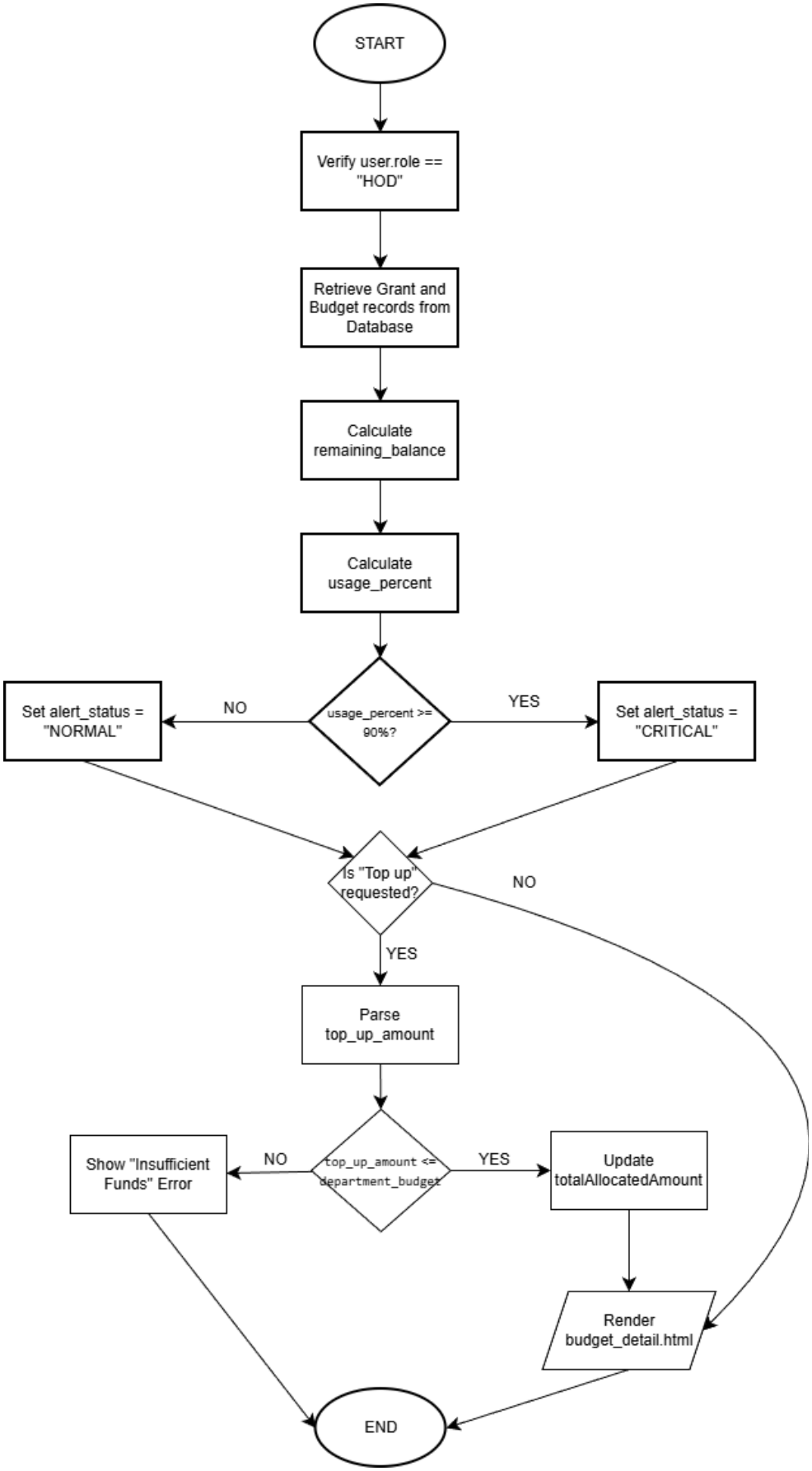
```
ALGORITHM TrackBudget(user, grant_id)
  1. VERIFY user.role equals "HOD"

  2. RETRIEVE Grant(grant_id) AND Budget(grant_id)

  3. CALCULATE Financial Metrics:
     SET total_spent = Budget.total_spent
     SET remaining = Grant.totalAllocatedAmount - total_spent
     SET usage_percent = (total_spent / Grant.totalAllocatedAmount) * 100

  4. MONITOR Threshold:
     IF usage_percent >= 90:
        SET alert_status = "CRITICAL"
     ELSE:
        SET alert_status = "NORMAL"

  5. IF Top-Up Requested (POST):
     a. PARSE top_up_amount
     b. IF top_up_amount <= hod_user.total_department_budget:
        UPDATE Grant.totalAllocatedAmount += top_up_amount
        UPDATE hod_user.total_department_budget -= top_up_amount
        SAVE records
```

c. ELSE:

RETURN Error("Insufficient Department Funds")

6. RENDER "budget_detail.html" WITH metrics and alert_status

END ALGORITHM

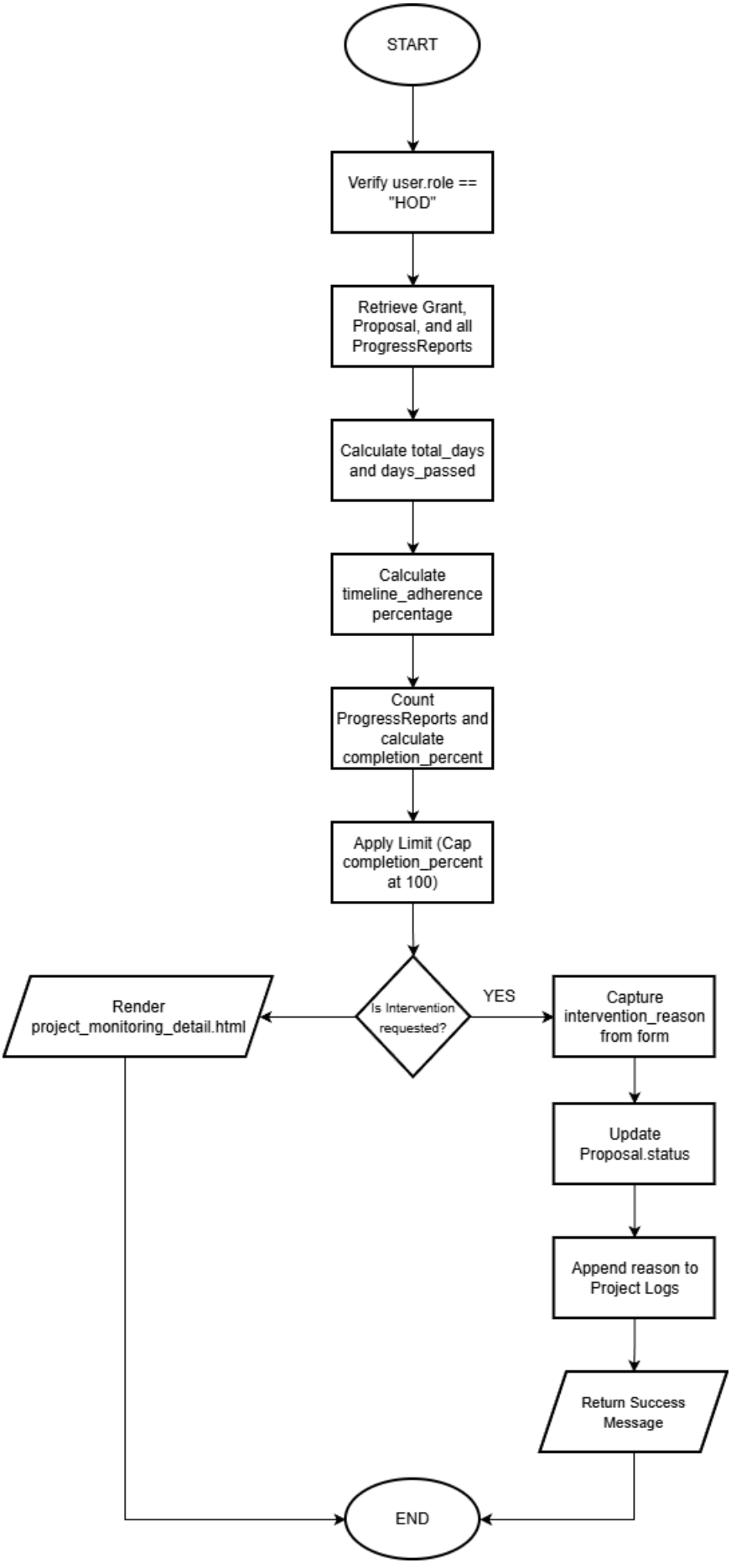

START
Verify user.role == "HOD"
Retrieve Grant and Budget records from Database
Calculate remaining_balance
Calculate usage_percent
usage_percent >= 90%?
NO
YES
Set alert_status = "NORMAL"
Set alert_status = "CRITICAL"
Is "Top up" requested?
NO
YES
Parse top_up_amount
top_up_amount <= department_budget
NO
YES
Show "Insufficient Funds" Error
Update totalAllocatedAmount
Render budget_detail.html
END

7.3.5 Component 3: Monitoring Engine

Description: The Monitoring Engine is responsible for the qualitative and quantitative oversight of research projects. It aggregates data from progress reports to evaluate project health through two primary Key Performance Indicators (KPIs): Timeline Adherence and Completion Percentage. This component allows the HOD to perform "Deep Dive" monitoring by reviewing individual report contents and provides a mechanism for "HOD Intervention" if a project is falling behind schedule or failing to meet milestones.

Pseudocode:

```
ALGORITHM MonitorProject(user, grant_id)
  1. VERIFY user.role equals "HOD"

  2. RETRIEVE Grant, associated Proposal, and List<ProgressReport>

  3. CALCULATE Timeline KPI:
     total_days = Grant.endDate - Grant.startDate
     days_passed = CurrentDate - Grant.startDate
     SET timeline_adherence = (days_passed / total_days) * 100

  4. CALCULATE Completion KPI:
     SET report_count = ProgressReport.count()
     SET completion_percent = report_count * 25  // Assuming 4 milestones
     LIMIT completion_percent TO 100

  5. IF Intervention Requested (POST):
     a. CAPTURE intervention_reason from form
     b. UPDATE Proposal.status = "Under Intervention"
     c. APPEND reason to Project Logs
     d. RETURN Success

  6. RENDER "project_monitoring_detail.html" WITH kpi_metrics and report_list

END ALGORITHM
```

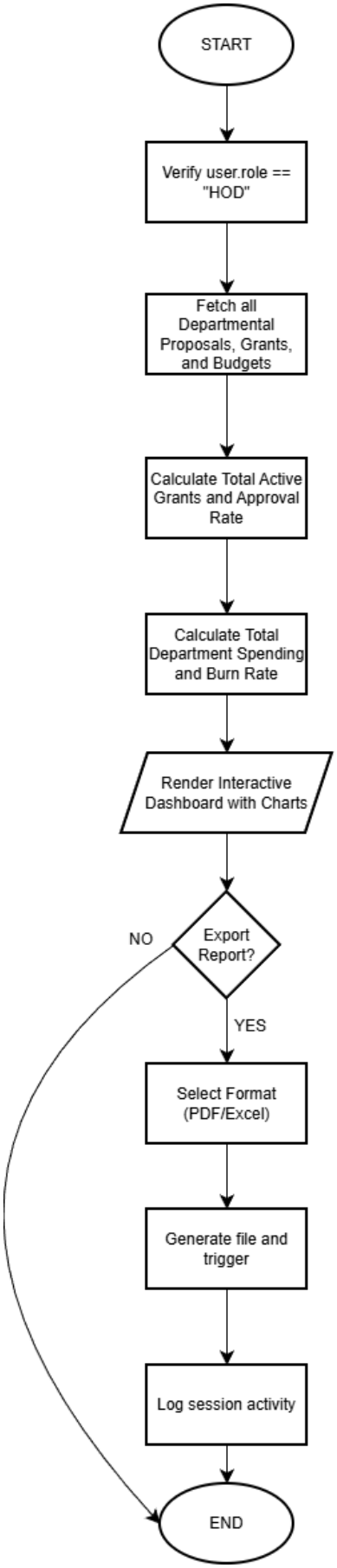

START
Verify user.role == "HOD"
Retrieve Grant, Proposal, and all ProgressReports
Calculate total_days and days_passed
Calculate timeline_adherence percentage
Count ProgressReports and calculate completion_percent
Apply Limit (Cap completion_percent at 100)
Is Intervention requested?
YES
Render project_monitoring_detail.html
Capture intervention_reason from form
Update Proposal.status
Append reason to Project Logs
Return Success Message
END

7.3.6 Component 4: Dashboard Manager (Department Analytics)

Description: The Dashboard Manager is the analytical hub of the HOD subsystem. It aggregates data from all proposals and active grants to provide a "helicopter view" of departmental performance. Its primary role is to calculate global metrics—such as the total budget burn rate and proposal success rates—and provide administrative tools for exporting these insights into formal reports (PDF/Excel).

Pseudocode:

```
ALGORITHM DashboardManager(user)
  1. VERIFY user.role equals "HOD"
     ELSE RETURN "Access Denied"

  2. RETRIEVE All Proposals AND All Grants linked to Department

  3. CALCULATE Department Metrics:
     SET total_active_grants = Count(Grants)
     SET approval_rate = (Count(Approved_Proposals) / Count(Total_Proposals)) * 100
     SET total_dept_spent = Sum(All_Budget.total_spent)
     SET total_dept_allocated = Sum(All_Grant.totalAllocatedAmount)
     SET overall_burn_rate = (total_dept_spent / total_dept_allocated) * 100

  4. IF Export Requested (POST):
     a. PARSE format (PDF or Excel)
     b. COMPILE metrics into report template
     c. TRIGGER file download
     d. LOG export activity in session logs

  5. RENDER "hod_dashboard.html" WITH aggregated_metrics AND charts
END ALGORITHM
```


START
Verify user.role == "HOD"
Fetch all Departmental Proposals, Grants, and Budgets
Calculate Total Active Grants and Approval Rate
Calculate Total Department Spending and Burn Rate
Render Interactive Dashboard with Charts
Export Report?
NO
YES
Select Format (PDF/Excel)
Generate file and trigger
Log session activity
END

8 Deployment Design

8.1 Deployment Diagram

The deployment diagram for your **Research Grant Management System (RGMS)** provides a comprehensive view of how your Django application is physically and logically distributed. It strictly follows the **three-tier architecture** described in your design specification, separating the user interface, business logic, and data storage.

1. Client Node: User Workstation (<<device>>)

This node represents the hardware (PC, Laptop, or Mobile) used by the system's actors: **Researchers**, **Reviewers**, and **HODs**.

- **Web Browser Environment:** The system requires a modern browser (Chrome, Safari, Firefox) that supports HTML5 and ES6 standards.
- **RGMS UI Artifact:** Inside the browser, the **Presentation Layer** is active. It consists of dynamic templates rendered by the **Django Template Engine**.
- **Communication:** All requests are sent via a **URL Dispatcher** over the **HTTP/HTTPS protocol** to the central server.

---

2. Application Tier: Django Cloud Server (<<device>>)

This is the centralized hosting environment that acts as the "brain" of the RGMS.

- **Django Runtime:** The server operates within a **Python 3.x** and **Django 4.x** execution environment.
- **Modular Subsystems (Inner Artifacts):** This layer contains the specific logic modules that handle the system's workflows:
    - **Auth Module:** Manages role-based access control (RBAC), ensuring that an HOD sees the department dashboard while a Researcher sees their own proposals.
    - **Submission & Review Subsystems:** These artifacts handle the **Proposal Manager** (for version control) and the **Evaluation Manager** (for reviewer feedback).
    - **Admin & Monitoring Subsystems:** These artifacts power the **Financial Tracker** and **KPI Engine**, which handle budget updates and performance metrics.

---

3. Data Tier: Persistence & Storage (<<device>>)

The data layer is hosted on a SQL-compliant engine. As indicated in your architecture, this tier is split into two specialized storage environments to handle different types of data.

A. Relational Database (Inner Box 1)

- **Execution Environment:** The system is designed for **SQLite** during development but is prepared for **PostgreSQL** in production.

- **Artifacts (Grant_Records.db):** This stores all structured entities defined in your Data Dictionary, such as User, Proposal, Grant, Budget, and Evaluation records.
- **Relationship:** It communicates with the Application Tier using the **Django ORM**.

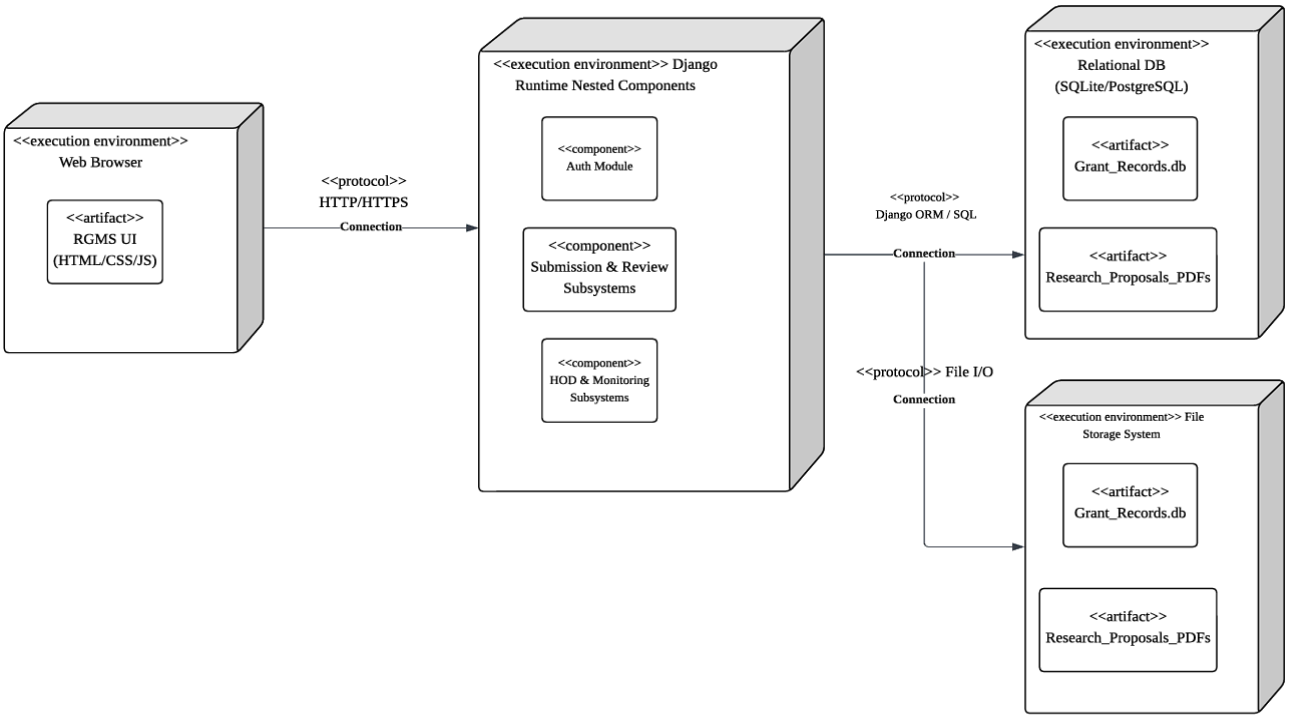
B. File Storage System (Inner Box 2)

- **Execution Environment:** A dedicated server-side file system.
- **Artifacts (Research_Proposals_PDFs):** This specifically manages unstructured data—the actual **PDF proposal documents** uploaded by researchers.
- **Relationship:** It is accessed via **File I/O** commands triggered by the Proposal Manager whenever a document is uploaded or viewed.

Summary of System Readiness

This deployment design confirms that the RGMS is ready for implementation because:

1. **Scalability:** The separation of the Database and File System allows for high-resolution chart rendering (via Matplotlib/Seaborn) without slowing down record lookups.
2. **Integrity:** The automated **Document Version Control** logic is integrated into the deployment flow, preventing duplicate submissions.
3. **Governance:** The **Financial Tracker** logic ensures that the HOD can monitor budget health in real-time, with critical alerts triggered at 90% usage



9 Summary

The design and documentation of the **Research Grant Management System (RGMS)** represent a complete blueprint for an enterprise-level academic tool. The project follows a **Layered Three-Tier Architecture** style, ensuring that the presentation, business logic, and data storage remain logically distinct yet highly integrated.

Design Approach

- **Tiered Structure**: User interactions are handled via a dynamic **Presentation Layer** (Django Templates), business rules are enforced by the **Application Layer** (Python logic), and records are persisted in the **Data Layer** (Relational DB/File System).
- **Actor-Centric Design**: The system is logically divided into three primary modules—**Researcher**, **Reviewer**, and **HOD**—each with specialized interfaces and permissions to ensure **Role-Based Access Control (RBAC)**.

Readiness for Implementation

The system is ready for the development phase based on the following readiness indicators:

- **Technical Alignment**: The deployment design is fully mapped to a **Cloud Server environment** compatible with **Python** and **Django.**
- **Logical Robustness**: Detailed **Sequence Diagrams** and **Pseudocode** for every major use case provide a clear roadmap for backend developers, specifically regarding document versioning and financial validation logic.
- **Data Integrity**: A comprehensive **Data Dictionary** and defined **Data Structures** (e.g., User, Proposal, Grant, Budget) ensure that the relational schema is optimized for both development (SQLite) and production (PostgreSQL).

References

State Machine Diagrams

[1] SmartDraw, "State Diagram - What is a State Machine Diagram?," SmartDraw.com. [Online]. Available: https://www.smartdraw.com/state-diagram/

[2] Lucidchart, "UML State Machine Diagram Tutorial," Lucidchart.com. [Online]. Available: https://www.lucidchart.com/pages/uml-state-machine-diagram

[3] S. Stotts, "State Transition Diagrams," UNC Computer Science. [Online]. Available: http://www.cs.unc.edu/~stotts/145/CRC/state.html

[4] "State Transition Diagrams," StickyMinds. [Online]. Available: https://www.stickyminds.com/article/state-transition-diagrams

[5] Lucidchart, "State Machine Diagram (Video)," YouTube. [Online]. Available: https://www.youtube.com/watch?v=PF9QcYWIsVE

[6] Visual Paradigm, "UML State Machine Diagram Tutorial (Video)," YouTube. [Online]. Available: https://www.youtube.com/watch?v=OsmWASXE2IM

Sequence Diagrams

[7] Lucidchart, "UML Sequence Diagram Tutorial," Lucidchart.com. [Online]. Available: https://www.lucidchart.com/pages/uml-sequence-diagram

[8] SmartDraw, "Sequence Diagram - Everything You Need to Know," SmartDraw.com. [Online]. Available: https://www.smartdraw.com/sequence-diagram/

[9] Visual Paradigm, "What is Sequence Diagram?," Visual Paradigm Guide. [Online]. Available:
https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-sequence-diagram/

[10] IBM Developer, "UML's Sequence Diagram," IBM Rational Library. [Online]. Available: https://www.ibm.com/developerworks/rational/library/3101.html

[11] Lucidchart, "Sequence Diagram Tutorial (Video)," YouTube. [Online]. Available: https://www.youtube.com/watch?v=XIQKt5Bs7II

[12] Visual Paradigm, "UML Sequence Diagram Tutorial (Video)," YouTube. [Online]. Available: https://www.youtube.com/watch?v=cxG-qWthxt4

[13] Derek Banas, "UML 2.0 Sequence Diagrams (Video)," YouTube. [Online]. Available: https://www.youtube.com/watch?v=18_kVIQMavE