A Time Monitoring Tool

Project 2 - Structural and Behavioral Modeling

Group 5

Team Members

Ratnesh Pawar	Rohan Sonawane	Abhishek Dhale
Mayank Sharma	FNU Sumit Kumar	Neil Duraiswami

1. **Structural Modeling** **2**
   i. Introduction 2
   ii. Class Identification 2
   iii. CRC Cards 2
   iv. Class Diagram 5
   v. Object Diagram 6
   vi. Verification and Validation 6

2. **Behavioral Modeling** **8**
   i. Introduction 8
   ii. Sequence Diagram 8
   iii. CRUDE Analysis 11
   iv. Complex Objects 11
   v. Behavioral State Machine 12
   vi. Verification & Validation 13

FIGURES LIST

1. ***Class Diagram for TMT System***
2. ***Object Diagram for TMT System***
3. **Sequence Diagrams**
   3.1 Sequence Diagram for developer
   3.2 Sequence Diagram for Manager
   3.3 Sequence Diagram for Task Prioritization
   3.4 Sequence Diagram for Resource Tracking
   3.5 Sequence Diagram for Activity Categorization
   3.6 Sequence Diagram for Data Analysis
   3.7 Sequence Diagram for Time Budgeting
   3.8 Sequence Diagram for TimeSheet
   3.9 Sequence Diagram for User Authentication
   3.10 Sequence Diagram for Recording Timestamp
4. **Behavioral State Machine**
   4.1 Sequence Diagram for Project
   4.2 Sequence Diagram for TimeSheet
   4.3 Sequence Diagram for Manager

Introduction:

We will identify and list the Classes utilized in the TMT system, produce CRC Cards, construct a Class and Object diagram, and verify and validate the Functional and Structural models necessary to develop an efficient TMT system.

Class Identification for TMT:

- **Manager:** Manages projects, writes reports, and supervises execution.
- **Developer:** Update the settings using your Assignment timestamp.
- **Authentication:** Controls user login, authentication, and access.
- **TimeSheet:** Maintains weekly activity information.
- **Report:** Handles the creation of different analytical reports and charts based on activities, projects, and resources.
- **Project:** Encompasses details about a specific project, its tasks, and the resources assigned to it.
- **Activity:** Holds details about tasks or actions performed, associated with timestamps, categories, and related notifications if any time budget violations occur.

CRC CARDS:

Developer:

Front:		
Class Name: Developer	ID: CRD001	Type: Entity
Description: Represents a software developer in the system		Associated Use Cases: Resource Tracking, User Authentication
Responsibilities: Manage personal profile information. - Record timestamps for activities. - Update skills and competencies.	Collaborators: Manager, Authentication, Activity, TimeSheet	
Back:		
Attributes: UserID, Username, Password, Email, Skills		
Relationships: Reports to Manager - Uses Authentication for login - Records activities in TimeSheet		

Manager:

Front:		
Class Name: Manager	ID: CRM002	Type: Entity
Description: Represents a team manager in the system.		Associated Use Cases: Resource Tracking, User Authentication, Task Prioritization

Responsibilities: • Manage team information. • Oversee project assignments. • Track the team's activities and timesheets.	Collaborators: Developer, Project, TimeSheet, Report
Back:	
Attributes: UserID, Username, Password, Email, TeamSize	
Relationships: Oversees Developers - Manages Projects - Analyzes Reports	

Authentication:

Front:		
Class Name: Authentication	ID: CRA003	Type: Service
Description: Manages user authentication in the system.		Associated Use Cases: User Authentication
Responsibilities: • Authenticate users. • Maintain security protocols.	Collaborators: Developer, Manager	
Back:		
Attributes: AuthToken, SecurityProtocol		
Relationships: Used by Developers and Managers for secure access		

Activity:

Front:		
Class Name: Activity	ID: CRA004	Type: Entity
Description: Represents a work activity or task		Associated Use Cases: Activity Categorization, Weekly Time Sheets
Responsibilities: • Store activity details. • Categorize activities.	Collaborators: Developer, TimeSheet	
Back:		
Attributes: ID, Description, Category		
Relationships: Recorded in TimeSheet by Developer		

TimeSheet:

Front:		
Class Name: TimeSheet	ID: CRT005	Type: Entity
Description: Represents weekly working hours and activities of a Developer.		Associated Use Cases: Weekly Time Sheets, Time Budgeting

Responsibilities: • Track weekly activities and hours worked.	Collaborators: Developer, Activity
Back:	
Attributes: WeeklyActivities, TotalHoursWorked	
Relationships: Contains Activities Associated with Developer	

Project:

Front:		
Class Name: Project	ID: CRP006	Type: Entity
Description: Represents a project with budget, status, and deadline.		Associated Use Cases: Task Prioritization, Resource Tracking
Responsibilities: • Track project status and budget. • Manage project details.	Collaborators: Manager, Developer	
Back:		
Attributes: ProjectID, Name, Budget, Status, Deadline		
Relationships: Managed by Manager Developers work on projects.		

Report:

Front:		
Class Name: Report	ID: CRR007	Type: Entity
Description: Used for generating different types of reports.		Associated Use Cases: Data Analysis
Responsibilities: • Generate various reports.	Collaborators: Manager	
Back:		
Attributes: Type, GenerationDate		
Relationships: Used by Managers for analysis		

Class Diagram for TMT:

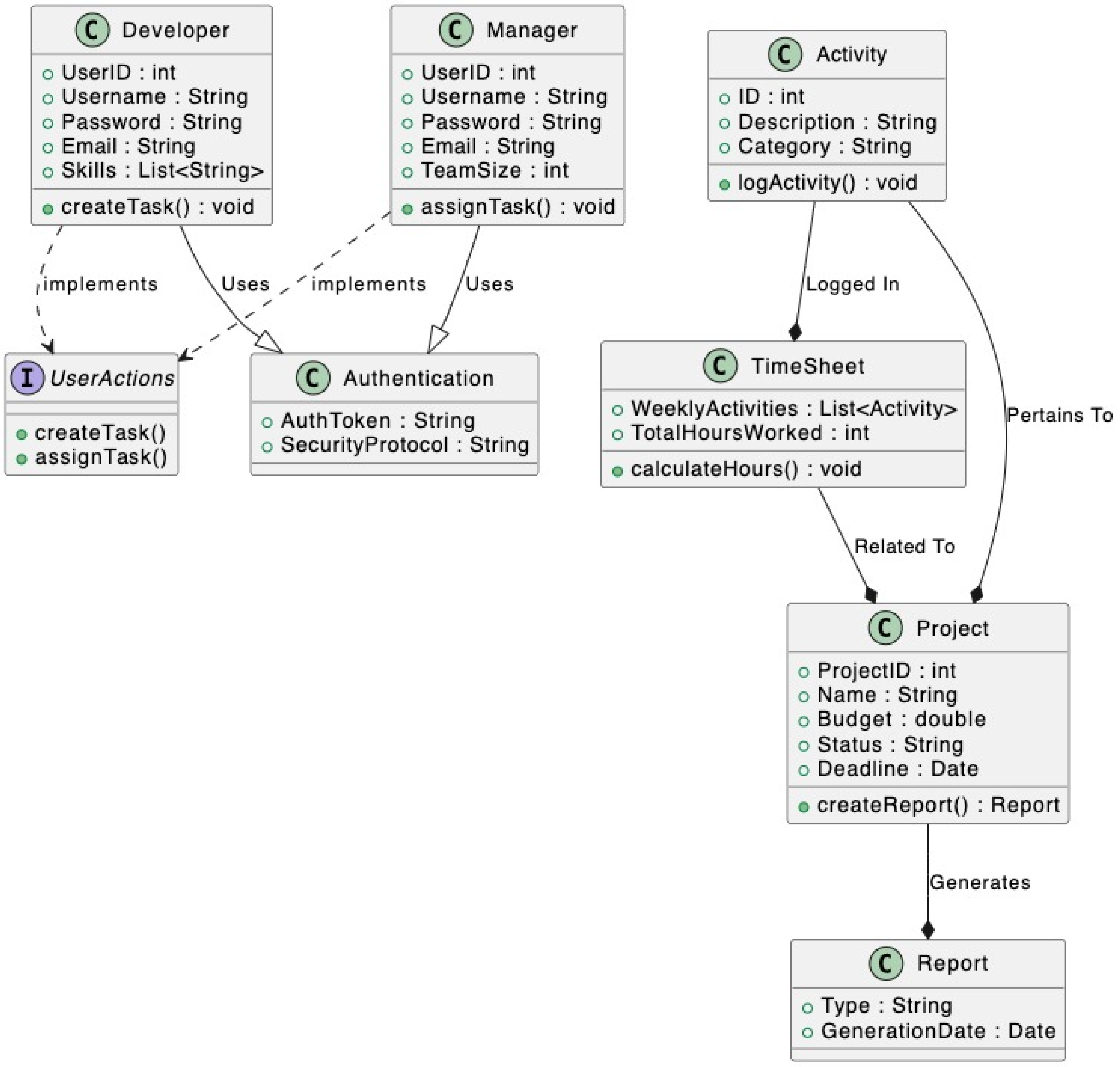

Fig 1: *Class Diagram for TMT System*

Object Diagram for TMT:

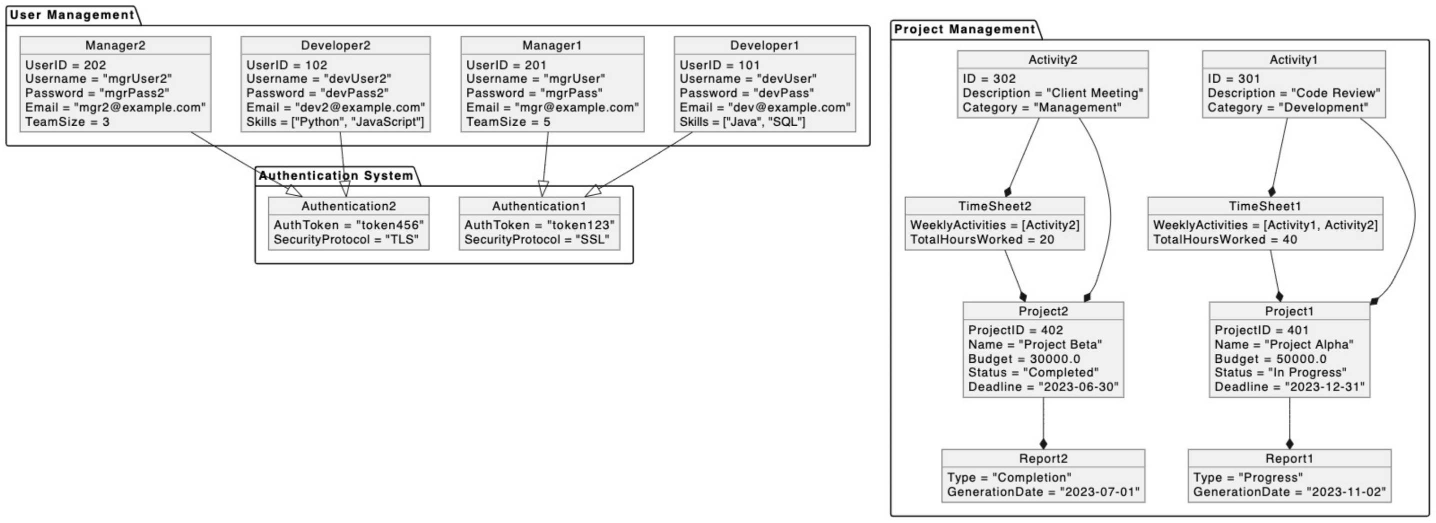

Fig 2: *Object Diagram for TMT System*

Verification and Validation for Functional Models

Alignment with Requirements:

- **Verified:** Reviewed project documentation and confirmed the inclusion of all key functionalities: task management, user authentication, time tracking, and report generation.

- **Validated:** Cross-referenced with user requirements and confirmed support for different user roles (Developer, Manager), aligning with project goals.

Process Flow:

- **Verified:** Examined process diagrams and ensured logical, complete workflows for task assignment, progress tracking, and report generation.

- **Validated:** Conducted user scenario testing, confirming no gaps in activity logging and timesheet reporting.

User Interaction:

- **Verified:** Reviewed UI/UX designs, ensuring clarity and ease of use in task management and report viewing.

- **Validated:** Performed usability testing with users, confirming effective and intuitive user interfaces.

Error Handling:

- **Verified:** Checked system design for error handling processes, especially for authentication failures and data entry errors.

- **Validated:** Tested error scenarios, confirming the system provides appropriate user feedback.

Performance and Scalability:

- **Verified:** Assessed system architecture for performance optimization and scalability.
- **Validated:** Conducted load testing, confirming system meets performance benchmarks and can scale under increased load.

Completed Verification and Validation of Structural Models

Consistency and Correctness:

- **Verified:** Reviewed the class diagram, confirming logical relationships (e.g., Developers linked to Authentication) and correct attribute types (e.g., Budget as a numerical value).
- **Validated:** Cross-checked these elements against system requirements, ensuring accuracy and relevance.

Completeness:

- **Verified:** Confirmed the inclusion of all necessary classes (Developer, Manager, Project, etc.) and attributes in the class diagram.
- **Validated:** Cross-referenced with the functional model and project requirements, ensuring no critical elements are missing.

Redundancy and Clarity:

- **Verified:** Identified and removed any redundant classes or attributes, ensuring the model's clarity.
- **Validated:** Received feedback from developers and architects, confirming the model's clarity and lack of redundancy.

Alignment with Requirements:

- **Verified:** Ensured the structural model aligns with the system's requirements, particularly in user management and project tracking.
- **Validated:** Reviewed with stakeholders, confirming the model meets the intended system structure.

Real-World Mapping:

- **Verified:** Checked that classes accurately reflect real-world entities like users, projects, and reports.
- **Validated:** Consulted with domain experts, ensuring accurate real-world applicability.

Flexibility and Scalability:

- **Verified:** Assessed the model for its ability to accommodate future changes, such as adding new project types or user roles.

- **Validated:** Discussed with system architects, confirming adaptability to future requirements.

Part II – Behavioral Modeling

Introduction:

We map out the dynamic interactions within our system. We use sequence diagrams to depict how entities communicate, and the CRUDE matrix to chart their operations. By identifying complex objects, we further detail their behavior with state machines.

Sequence Diagrams for the TMT System Use Cases

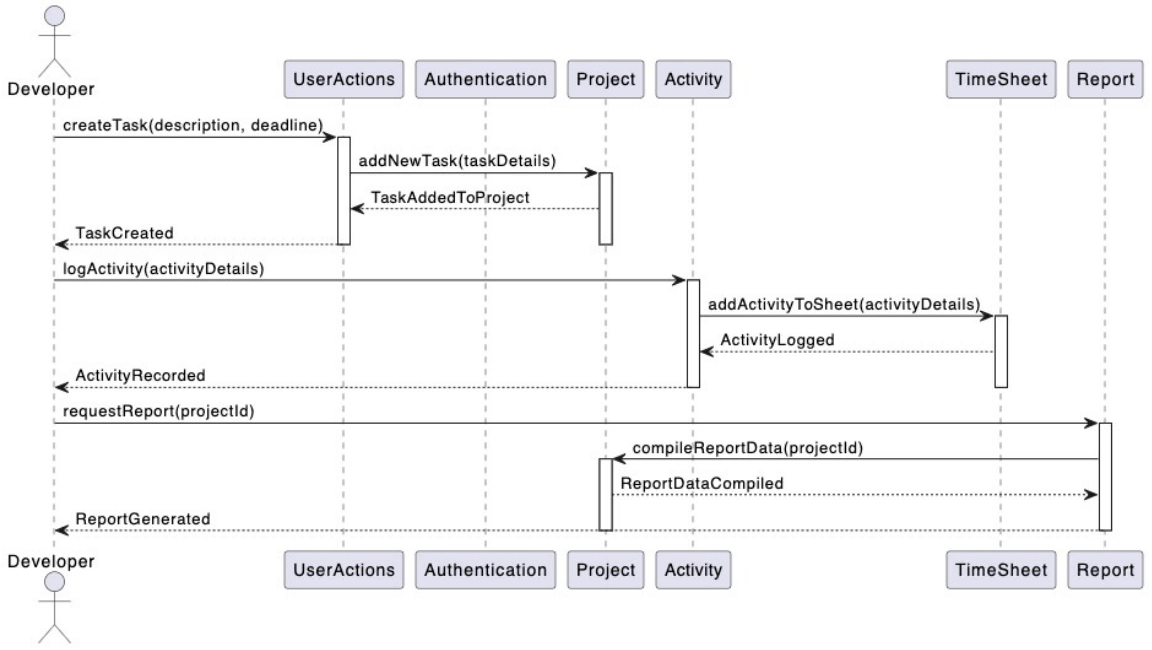

Fig 3.1: *Sequence Diagram for developer*

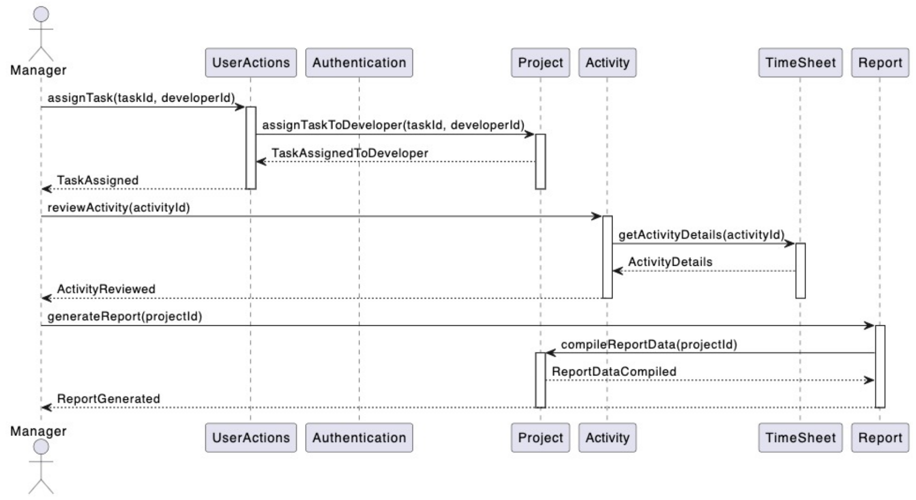
Manager
UserActions
Authentication
Project
Activity
TimeSheet
Report
assignTask(taskId, developerId)
assignTaskToDeveloper(taskId, developerId)
TaskAssignedToDeveloper
TaskAssigned
reviewActivity(activityId)
getActivityDetails(activityId)
ActivityDetails
ActivityReviewed
generateReport(projectId)
compileReportData(projectId)
ReportDataCompiled
ReportGenerated

Fig 3.2: *Sequence Diagram for Manager*

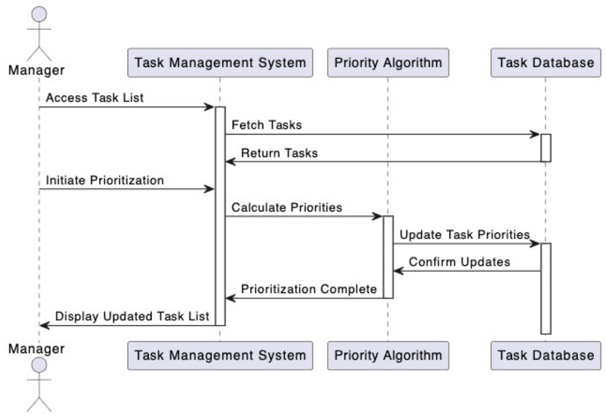

Fig 3.3: *Sequence Diagram for Task Prioritization*

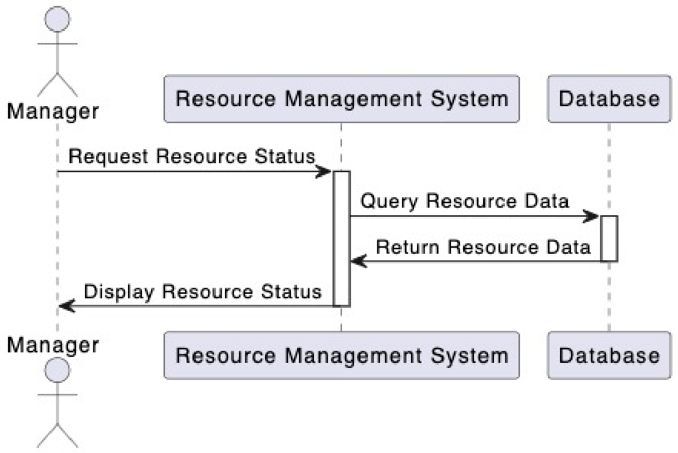

Fig 3.4: *Sequence Diagram for Resource Tracking*

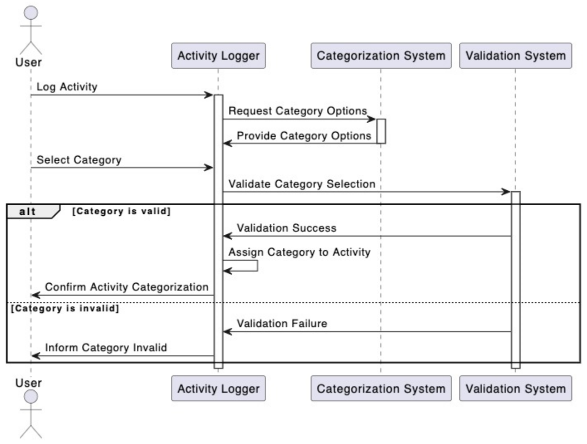

Fig 3.5: *Sequence Diagram for Activity Categorization*

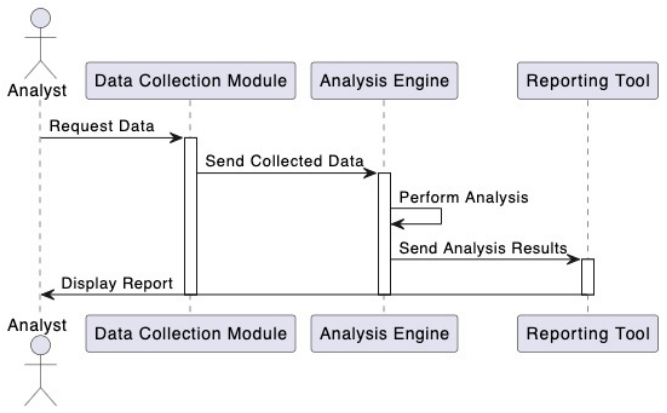

Fig 3.6: *Sequence Diagram for Data Analysis*

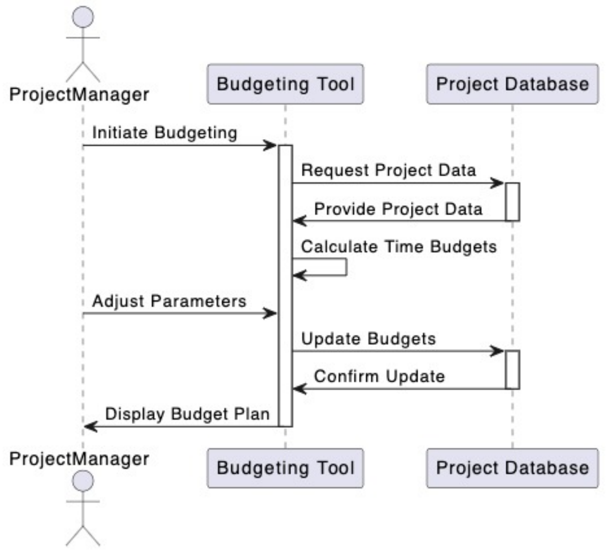

Fig 3.7: *Sequence Diagram for Time Budgeting*

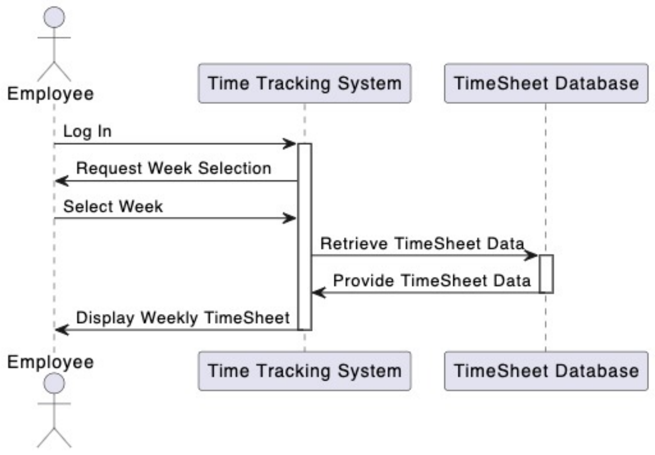

Fig 3.8: *Sequence Diagram for TimeSheet*

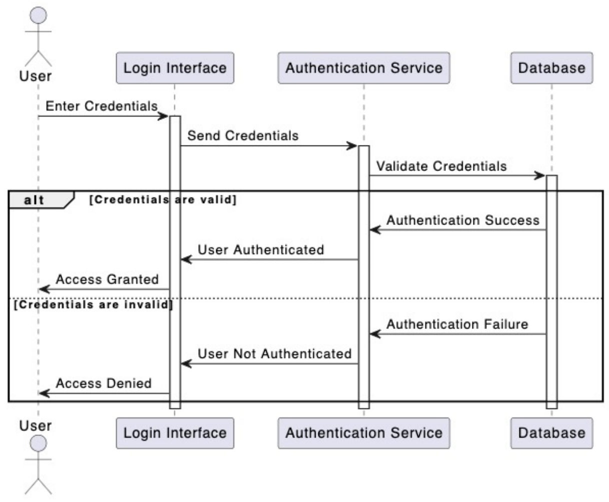

Fig 3.9: *Sequence Diagram for User Authentication*

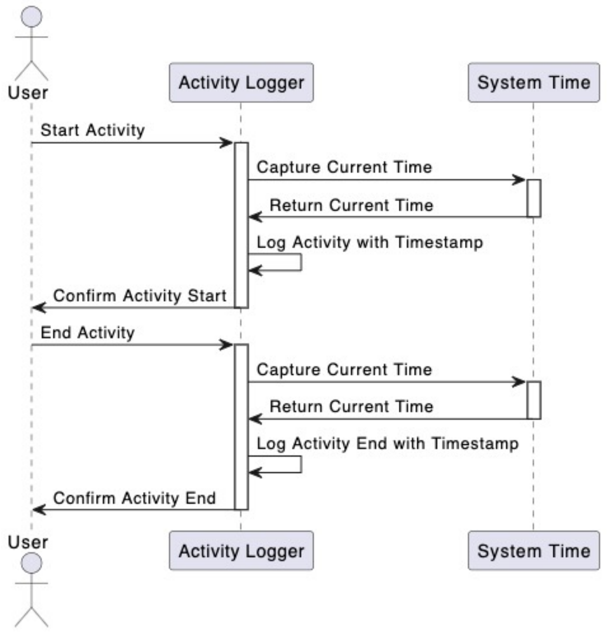

Fig 3.10: *Sequence Diagram for Recording Timestamp*

CRUDE Analysis for TMT System:

The matrix is built based on the relationships and interactions that the objects in the system have with one another, as concluded from the provided object and class diagrams.

	Report	Project	Activity	Manager	Developer	Timesheet	Auth
Report	RUD	R	-	-	-	-	-
Project	R	RUD	RC	RU	R	-	-
Activity	-	R	RUD	RC	RUD	RC	-
Manager	R	RU	RU	RUD	CRUD	RU	RE
Developer	R	RU	RU	-	R	RU	RE
Timesheet	-	-	R	R	RU	RUD	-
Auth	-	-	-	RE	RE	-	RUD

Complex Objects:

Project Objects (Project1, Project2):

- **Attributes:** ProjectID, Name, Budget, Status, Deadline.
- **Relationships:** They are central to the system, linking with TimeSheet objects, generating Report objects, and associated with Activity objects. The complexity arises from their multifaceted role in managing various aspects of a project, including budget, status, and deadlines, as well as their interactions with multiple other objects.

TimeSheet Objects (TimeSheet1, TimeSheet2):

- **Attributes:** WeeklyActivities, TotalHoursWorked.
- **Relationships:** These objects contain Activity objects and are related to Project objects. The complexity is due to their role in tracking activities and hours worked, which is essential for project management and reporting.

Manager Objects (Manager1, Manager2):

- **Attributes:** UserID, Username, Password, Email, TeamSize.
- **Relationships:** Each manager is associated with an authentication object and likely plays a key role in overseeing projects, assigning tasks, and managing teams. The complexity comes from their administrative and supervisory responsibilities in the system.

Behavioral State Machine for Complex Objects:

Project:

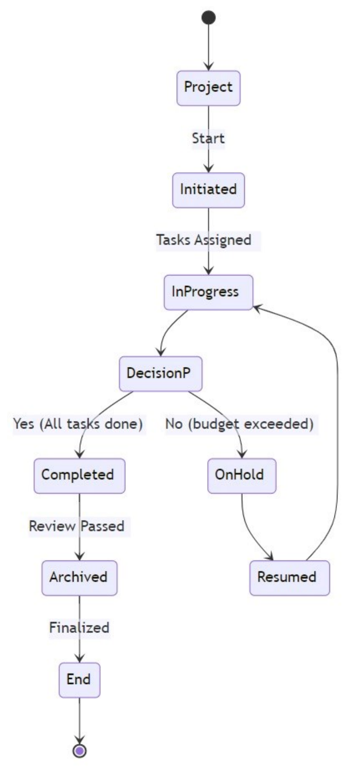

Fig 4.1: *Behavioral State Machine for Project*

Timesheet:

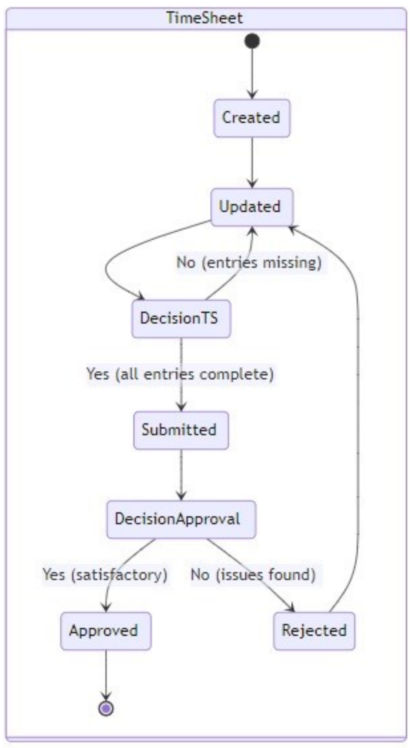

Fig 4.2: *Behavioral State Machine for Timesheet*

Manager:

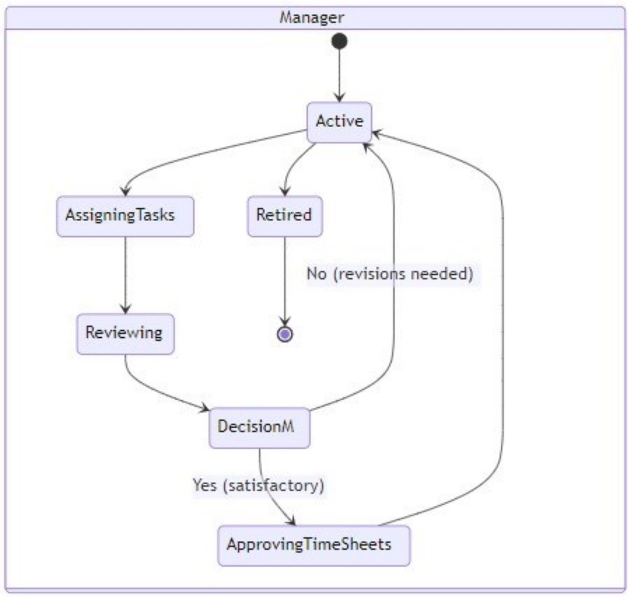

Fig 4.3: *Behavioral State Machine for Manager*

Validation and Verification of functional and behavioral models

We analyzed the behavioral state machines for Manager, TimeSheet, and Project and compared them with the functional description of features like Recording Timestamps, Activity Categorization, and Weekly Time Sheets. This comprehensive validation and verification approach ensured the robustness of the project.

- **User Scenario Testing**: Based on the state machines, we identified various user scenarios. From the Manager's state machine, a scenario emerged where a manager reviewed a task and found it unsatisfactory, leading to the "No (revisions needed)" state. We conducted User Acceptance Testing (UAT) and confirmed the system's behavior matched end-user expectations.

- **Comparison with Functional Requirements:** We aligned states and transitions in the state machines with the functional descriptions. For example, the TimeSheet's transition from "Updated" to "Submitted" directly corresponded with the "Recording Timestamps" function.

- **Formal Analysis:** Using specialized tools, we checked the state machine diagrams for consistency and completeness. This exercise confirmed the absence of dead-ends, unreachable states, or circular dependencies.

- **Peer Review:** Team members meticulously reviewed the state machine diagrams. They confirmed the correctness of every decision point, such as the "DecisionTS" in the TimeSheet state machine having both 'yes' and 'no' outcomes.

- **Traceability:** Every state and transition in the behavioral models is traced back to a requirement or user story, ensuring the models' completeness.

- **Documentation Review:** We cross-referenced the detailed functional descriptions with the project's documented requirements. The "Task Prioritization" function, for instance, aligned perfectly with detailed requirements.

Validation and verification of structural and behavioral models

- **Analysis:** We carefully examined the behavioral state machines for Manager, TimeSheet, and Project to ensure that both models perfectly aligned with the project's defined scope. We cross-referenced them with functional descriptions of tasks such as Recording Timestamps, Activity Categorization, and Weekly Time Sheets to confirm their robustness and reliability. This comprehensive approach to validation and verification confirmed the structural and behavioral models of our project, making it more reliable and efficient.

- **User Scenario Testing:** After inspecting the state machines, we were able to identify potential user scenarios. One notable scenario we extracted from the Manager's state machine was when a manager reviewed a task and determined it to be unsatisfactory, causing the flow to transition to the "No (revisions needed)" state. To ensure that the system's behavior was perfectly aligned with end-user expectations, we implemented User Acceptance Testing (UAT) sessions. These sessions allowed us to test the system and make sure it met the needs of our users.

- **Comparison with Functional Requirements:** Our team carefully aligned the state machines with functional requirements. For example, the TimeSheet transition from "Updated" to "Submitted" was synchronized with "Recording Timestamps".

- **Formal Analysis:** We utilized advanced tools and software to conduct a comprehensive review of our state machine diagrams. The examination verified that our diagrams were coherent and complete, and enabled us to identify and eliminate any potential issues such as dead ends, unreachable states, and circular dependencies.

- **Peer Review:** The peer review sessions played a crucial role in validating our work. Our team members meticulously examined the state machine diagrams, utilizing their diverse expertise. This collaborative effort ensured that every decision point was infallible. Notably, the "DecisionTS" node of the TimeSheet state machine was confirmed to have clear 'yes' and 'no' outcomes, leaving no room for ambiguity.

- **Traceability:** As a part of our methodology, we introduced a traceability matrix to ensure that our models are complete and aligned. This matrix enabled us to track every state and transition in our behavioral models back to a specific requirement or user story. By following this systematic approach, we were able to establish the accuracy and comprehensiveness of our models.

- **Documentation Review:** In a collaborative effort, our team cross-referenced the detailed functional descriptions with the project's overarching documented requirements. This alignment was exemplified in the "Task Prioritization" function, which perfectly mapped to the requirements.