A Time Monitoring Tool

Project 3 - Object Oriented Design

Group 5

Team Members

Ratnesh Pawar [16.7] Rohan Sonawane [16.7] Abhishek Dhale [16.7]

Mayank Sharma [16.7] FNU Sumit Kumar [16.7] Neil Duraiswami [16.7]

FIGURES LIST

Introduction:

- We are focusing on the object-oriented design of our Time Monitoring Tool (TMT), a system designed to make time monitoring for software teams easier, in the project's third iteration.
- We address TMT's class design, architectural framework, and real-world configuration in this study. Creating a package diagram that summarizes TMT's structure, outlining how each class will collaborate inside the system, and layering classes for clarity are among our responsibilities.
- In addition, we'll go over the practical aspects of putting TMT into practice, including everything from class management to system deployment. In simple terms, this study discusses setting up an object-oriented software design for TMT so that it is ready for future growth as well as operation.

Packages and Package Diagrams

The package diagram appears to divide the TMT system into four primary packages: **TimeTracking, UserManagement, DataHandling, and Utilities.**

- **TimeTracking Package:**
  This package contains classes such as TimeSheet, Activity, and Project, which are central to the core functionality of the TMT system. These classes handle the recording of time, management of individual activities or tasks, and project-related information.

- **UserManagement Package:**
  It includes the Authentication, Developer, and Manager classes, alongside a User class that Developer and Manager inherit from. This package is responsible for handling user authentication and the different roles users may have within the system.

- **DataHandling Package:**
  - This package is made up of DataExporter and DatabaseManager classes. It is focused on the management of data persistence, including database connections and the exporting of data for reporting or external use.

- **Utilities Package:**
  - Comprising ErrorHandling and Logger classes, this package provides support functions across the system, such as logging events and handling errors.

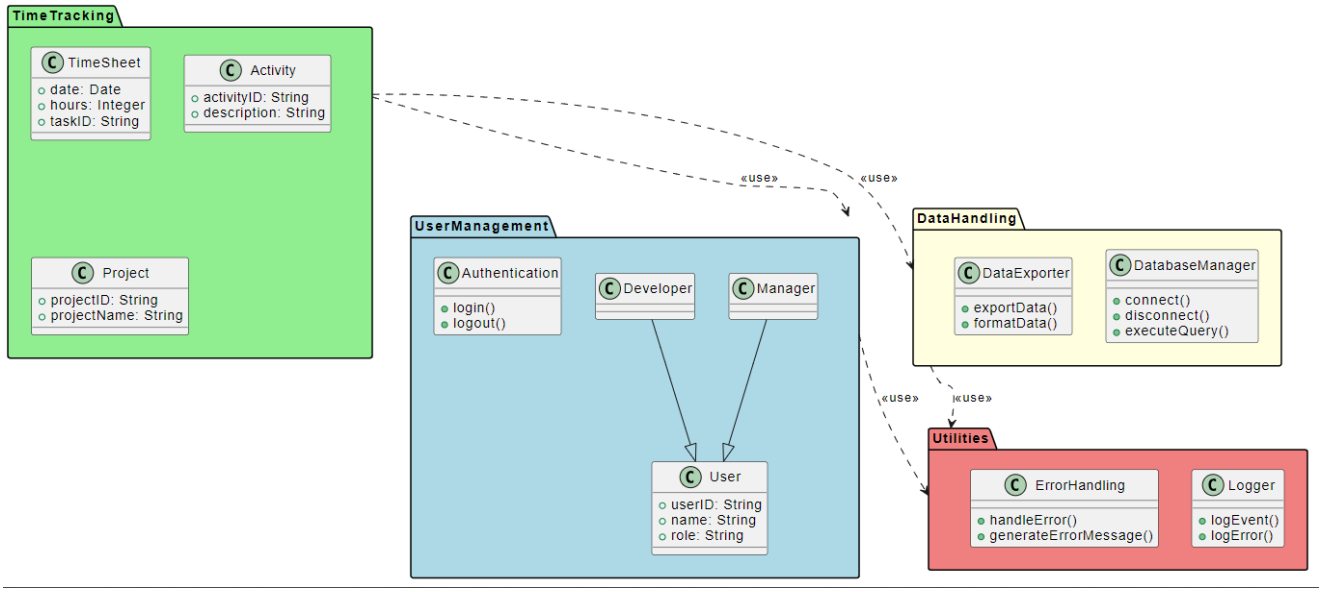

Fig 1: Package Diagrams

Class Specifications

Manager

- Class Constraints (OCL):
  - `context Manager inv: self.projects->forAll(p | p.manager = self)`
  - This ensures that all projects managed by a `Manager` instance are indeed associated with that manager.
- Class Contract: Responsible for overseeing projects and generating reports.

Developer

- Class Constraints (OCL):
  - `context Developer inv: self.loggedTime->notEmpty() implies self.loggedTime.project->includes(self.assignedProject)`
  - Ensures that a `Developer` can only log time for projects they are assigned to.
- Class Contract: Engages in development activities and logs time.

Authentication

- Class Constraints (OCL):
  - `context Authentication inv: User.allInstances()->forAll ( u | u.username->isUnique and u.password->notEmpty() )`
  - Ensures that all users have a unique username and a non-empty password.
- Class Contract: Manages user authentication and access control.

TimeSheet

- Class Constraints (OCL):
  - `context TimeSheet inv: self.entries->forAll(e | e.duration > 0)`
  - Ensures that all timesheet entries have a positive duration.
- Class Contract: Records and maintains time entries.

Report

- Class Constraints (OCL):
  - `context Report inv: self.generatedReports->notEmpty() implies self.dataSource->notEmpty()`
  - Ensures that a report is generated only if there is data available.
- Class Contract: Generates various analytical reports.

Project

- Class Constraints (OCL):
  - `context Project inv: self.activities->notEmpty() implies self.resources->notEmpty()`
  - Ensures that a project with activities must have resources assigned.
- Class Contract: Manages project details and resources.

Activity

- Class Constraints (OCL):
  - `context Activity inv: self.duration <= self.allocatedTime`
  - Ensures that the duration of an activity does not exceed the allocated time.
- Class Contract: Tracks tasks and associated time.

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		

Methods

Method specifications for each class in the Time Monitoring Tool (TMT) system:

Project Manager Class Methods

Method Name: createProject	**Class Name:** Manager	**ID:** 101
Contact ID: 2023-MGR	**Programmer:** Jane Smith	**Date Due** 2023-12-31
Programming Language: ☑ Java		
Triggers/Events: Customer request for new project via interface		
Arguments Received: **Data Type:** string	**Notes:** Contains all necessary information to create a project. Must be non-null and valid.	
Messages Sent & Arguments Passed: **ClassName:** ProjectDetails**, MethodName:** NotificationService.sendProjectCreationNotice()	**Data Type:** String	**Notes:** The newly created project instance.
Arguments Returned: Project **Data Type:** String	**Notes:** Sends a notification that a new project has been created.	
Algorithm Specification: See Project Management Process Doc v2.3		
Misc Notes: Ensure compatibility with existing project management tools.		

Method Name: generateReport	**Class Name:** Manager	**ID:** 102
Contact ID: MGR-2023-RPT	**Programmer:** A. Smith	**Date Due** 2023-06-30
Programming Language: ☑ Java		
Triggers/Events: Request for report through Reporting Interface		
Arguments Received: **Data Type:** ReportType	**Notes:** Enum detailing the type of report required.	
Messages Sent & Arguments Passed: ClassName: **MethodName:** Database.query	**Data Type:** SQLQuery	**Notes:** Retrieves project data based on report type
Arguments Returned: Report **Data Type:** String	**Notes:** Generated report object with project data.	
Algorithm Specification: Algorithms to aggregate project data based on report type, format data, and generate report. Detailed steps in Algorithm Docs.		
Misc Notes: Ensure data privacy and adhere to GDPR standards.		

Developer Class Methods

Method Name: updateSettings	**Class Name:** Developer	**ID:** 201
Contact ID: DEV2023Settings	**Programmer:** Jane Developer	**Date Due** 2023-07-15
Programming Language: ☑ Java		
Triggers/Events: The developer submits a change via Settings Interface		
Arguments Received: **Data Type:** Settings	**Notes:** Object containing configuration settings	
Messages Sent & Arguments Passed: ClassName:, **MethodName:** SettingsManager.applyChanges	**Data Type:** Settings	**Notes:** Applies new settings and validates changes
Arguments Returned: void **Data Type:** String	**Notes:** No return value, confirmation via interface.	
Algorithm Specification: Steps to validate and apply settings, refer to the system documentation for a detailed workflow.		
Misc Notes: Ensure to handle exceptions and log updates.		

Method Name: logTime	**Class Name:** Developer	**ID:** 202
Contact ID: DEV2023TimeLog	**Programmer:** Dev A. Logger	**Date Due** 2023-08-01
Programming Language: ☑ Java		
Triggers/Events: The developer submits time log entry via the Time Tracking Interface		
Arguments Received: **Data Type:** Timestamp, String	**Notes:** Timestamp for the log entry, taskID for the task to log time against	
Messages Sent & Arguments Passed: ClassName:, **MethodName:** SettingsManager.applyChanges	**Data Type:** Settings	**Notes:** Applies new settings and validates changes
Arguments Returned: void **Data Type:** Void	**Notes:** No return value, time log entry is confirmed via UI.	
Algorithm Specification: Verify the validity of timestamp and taskID, then log the time to the corresponding task.		
Misc Notes: If the taskID does not exist, throw an error. Ensure synchronization for concurrent log entries.		

Authentication Class Methods

Method Name: login	**Class Name:** Authentication	**ID:** 301
Contact ID: AUTH2023Login	**Programmer:** Sam Securicode	**Date Due** 2023-09-01
Programming Language: ☑ Java		
Triggers/Events: User submits login credentials via Login Interface		
Arguments Received: **Data Type:** String, String	**Notes:** username and password for authentication	
Messages Sent & Arguments Passed: ClassName:, **MethodName:** SessionManager.startSession	**Data Type:** UserSession	**Notes:** Initiates a new session with user privileges
Arguments Returned: **Data Type:** UserSession	**Notes:** Represents the authenticated user's session	
Algorithm Specification: Validate credentials against user database, create session token, and return UserSession object.		
Misc Notes: Ensure encryption for the password during transit. Log all login attempts for security auditing.		

Method Name: logout	**Class Name:** Authentication	**ID:** 302
Contact ID: AUTH2023Logout	**Programmer:** Alex Securecode	**Date Due** 2023-10-01
Programming Language: ☑ Java		
Triggers/Events: User initiates logout via Logout Interface		
Arguments Received: **Data Type:** String, String	**Notes:** sessionID to identify the user session	
Messages Sent & Arguments Passed: MethodName: SessionManager.endSession	**Data Type:** String	**Notes:** Terminates the user session based on sessionID
Arguments Returned: **Data Type:** Void	**Notes:** No return value, confirmation via UI	
Algorithm Specification: Locate the session by sessionID, verify it is active, and then safely terminate the session.		
Misc Notes: Log the session termination for audit purposes. Handle any exceptions and cleanup resources.		

Physical Architecture

For the Time Monitoring Tool (TMT), selecting a physical architecture that aligns with the criteria of cost, development ease, interface capabilities, control, security, and scalability, a cloud-based layered architecture is recommended. When selecting a physical architecture for the Time Monitoring Tool (TMT), several considerations come into play, each with its own set of pros and cons. **Let's explore these in the context of a cloud-based layered architecture:**

Presentation Layer

- **Location**: Cloud platforms (AWS, Azure, Google Cloud).
- **Components**: Web interfaces, mobile apps.
- **Function**: Interface for developers, managers with TMT.
- **Cost & Development Ease**: Lower upfront costs, easy development with cloud tools.
- **Interface Capabilities**: Supports various devices, browsers; integrates with cloud services.

Business Logic Layer

- **Location**: Serverless functions, cloud servers.
- **Components**: Business logic, authentication, authorization.
- **Function**: Manages user requests, business rules, authentication.
- **Cost & Development Ease**: Lower operational costs, rapid deployment.
- **Control and Security**: Managed cloud security, custom protocols.

Data Access Layer

- **Location**: Cloud databases (Amazon RDS, Azure SQL).
- **Components**: Databases, storage services.
- **Function**: Manages data storage, retrieval.
- **Scalability**: Scalable, supports backups, redundancy.
- **Control and Security**: Strong encryption, cloud security.

Integration Layer

- **Location**: Cloud integration services.
- **Components**: API gateways, message queues (AWS SQS, Azure Service Bus).
- **Function**: Communication between services, external integrations.
- **Interface Capabilities**: Supports various protocols, data formats.

Network Infrastructure

- **Location**: Managed in cloud.
- **Components**: Load balancers, firewalls, CDN.
- **Function**: Manages traffic, security, content delivery.
- **Scalability & Security**: Scalable infrastructure, enhanced cloud security.

Monitoring and Maintenance

- **Location**: Cloud monitoring tools (AWS CloudWatch, Azure Monitor).
- **Components**: Monitoring agents, log services.
- **Function**: System performance monitoring, error logging.
- **Ease of Development**: Simplified setup, proactive maintenance.

Cloud-based layered architecture for TMT optimizes cost efficiency, scalability, and ease of development, while ensuring robust security and control. It leverages the strengths of cloud computing to provide a flexible, scalable, and secure environment, suitable for the dynamic needs of the TMT system.

Design Patterns

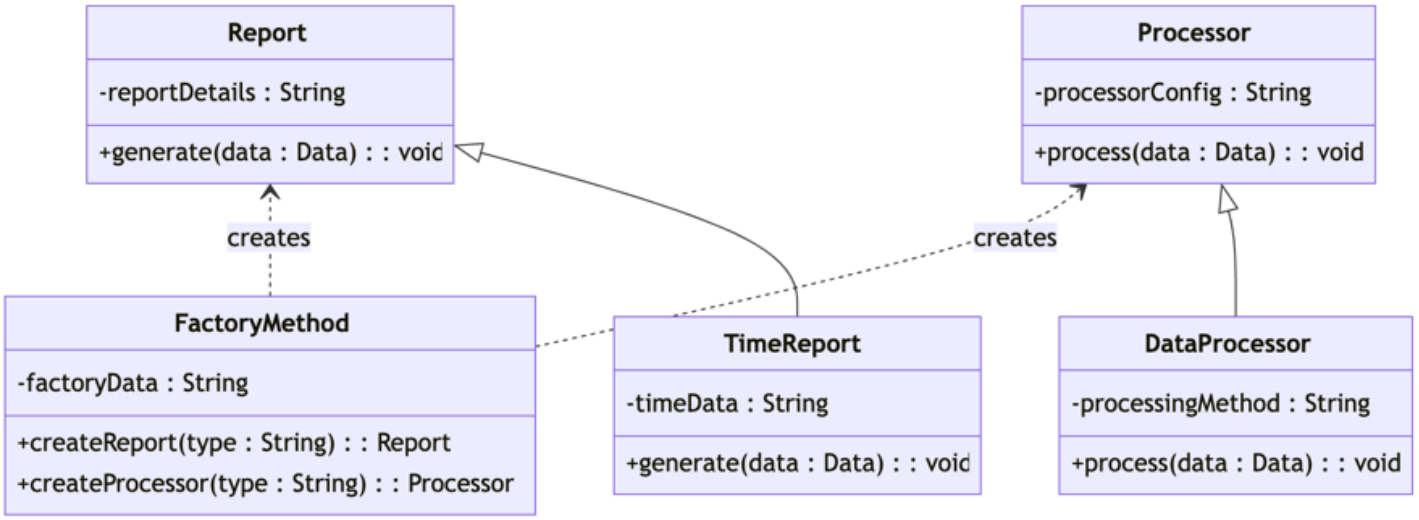

Fig 2.1: Creational Design Pattern

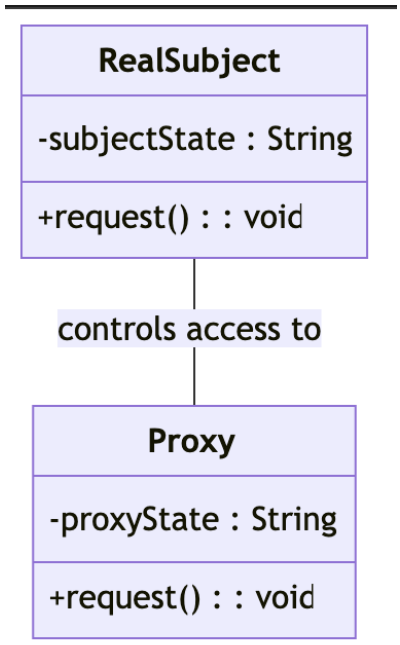

Fig 2.2: Structural Design Pattern

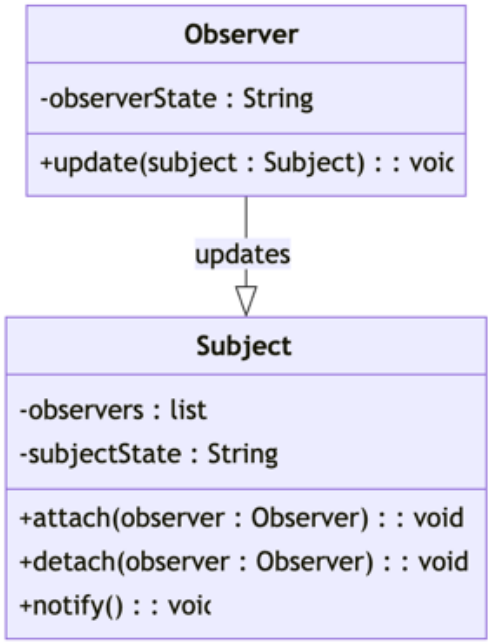

Fig 2.3: Behavioral Design Pattern

The Time Monitoring Tool (TMT) has been thoroughly studied and evaluated to determine the best design patterns to use. Three design patterns were scrutinized - **Creational, Structural, and Behavioral** - to identify the most efficient and effective solution. The Factory Method design pattern, which falls under Creational, enables flexible report and processor creation. The Proxy design pattern, which falls under Structural, ensures controlled access to system components, enhances security, and effectively manages resource load. Lastly, the Observer design pattern, which falls under Behavioral, supports a responsive system where changes are propagated to interested parties. This allows for updates in time entries to reflect across reports, providing a seamless experience.

The optimal design pattern for TMT depends on specific project needs, but a blend of patterns often yields the most robust solution. Each pattern addresses different aspects of software design, ensuring that TMT is flexible, secure, and responsive to changes within its environment.

For the deployment diagram of TMT, **the Structural design pattern - specifically, the Proxy pattern** - was selected. This pattern is the most suitable for controlling access and managing the complexity of interactions between client devices and the server. It allows for the secure and efficient handling of requests, which is crucial in cloud-based environments depicted in the deployment diagram. This choice aligns with the need for security and scalability within TMT's cloud-based infrastructure, which should not be compromised.

Deployment Diagram

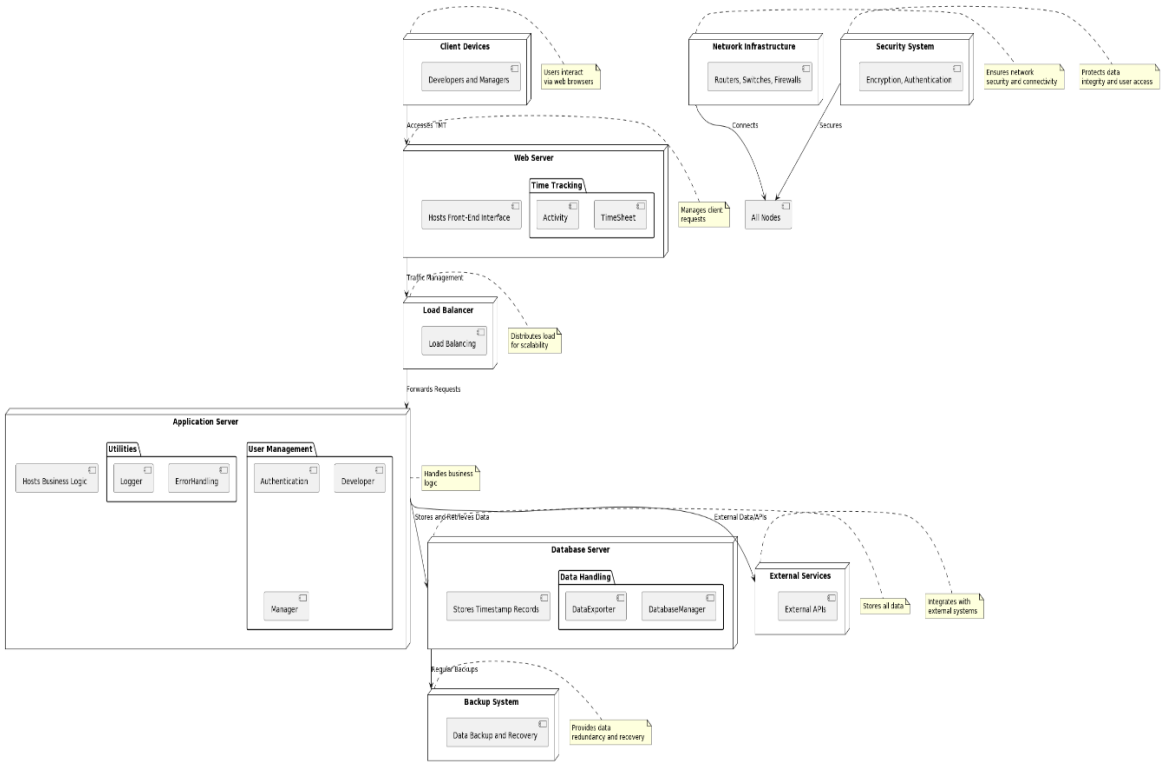

Fig 3: Deployment Diagram

The deployment diagram of the Time Monitoring Tool clearly presents its architecture overview. The system is accessed by users through web browsers. The web server seamlessly hosts the user interface components 'TimeSheet' and 'Activity'. Business logic is processed on an application server, which handles user management and utility services, including error logging and authentication processes. The database server efficiently manages data storage and retrieval, interfaced with backup systems for data integrity. The network infrastructure ensures secure connectivity, and a load balancer efficiently distributes web traffic. Additionally, external services are incorporated for further API functionalities.

Verification and Validation of the Non-Functional Requirements:

- **Operational Requirements: Technical Environment**
  - **Verification:** Review the system's architecture and deployment diagram to ensure it is designed for a web environment. Check for compatibility with various web browsers and internet connectivity requirements.
  - **Validation:** Test the system in a web environment to confirm it operates as intended. This includes testing on different browsers and network conditions.

- **Performance Requirements: Speed**
  - **Verification:** Analyze the system design and server configurations to ensure they support the required response time of less than 1 second.
  - **Validation:** Conduct performance testing using tools like LoadRunner or Apache JMeter. Measure the response times for various actions to ensure they meet the specified requirement.

- **Security Requirements: Encryption and Authentication**
  - **Verification:** Review the security architecture to confirm that encryption and authentication mechanisms are in place. This includes checking the implementation of SSL/TLS for data transmission and secure storage of user information.
  - **Validation:** Perform security testing, including encryption protocol analysis and authentication process testing, to ensure that user data is securely encrypted, and authentication mechanisms are robust.

- **Cultural/Political Requirements: Legal Compliance**
  - **Verification:** Review legal requirements relevant to the system's operation, such as data protection laws (like GDPR or CCPA). Ensure the system's design complies with these laws.
  - **Validation:** Consult with legal experts to confirm compliance. Conduct a compliance audit to ensure all legal requirements are met, including data handling, user privacy, and information storage.

Through this structured approach to verification and validation, the TMT system's non-functional requirements are thoroughly assessed to ensure they align with operational, performance, security, and legal standards. This process ensures the system is robust, efficient, secure, and compliant with necessary regulations and standards.

Conclusion Remarks

The Time Monitoring Tool (TMT) project report encapsulates a cutting-edge design approach that uses object-oriented principles to establish a highly efficient, secure, and maintainable system. By integrating design patterns with system architecture, the TMT is not only theoretically sound but also practically equipped to ensure effective implementation. Aligning class responsibilities with system operations and modeling interactions within a cloud-based environment, the TMT is well-prepared to meet its functional and non-functional requirements. The report lays a solid foundation for the next phase of development and sets a clear path forward for testing and deployment. With the TMT, you can enjoy an efficient, user-friendly, and reliable platform for time tracking and management.