Role Based Task Tracking System (RBTTS)

Tracking Made Simple, Results Made Powerful.

NANDINI KAMEPALLI

Associate Software Engineer

Date: 08/09/2025

Summary of Learning & Training Highlights

Overview of Training

During this training program, I gained hands-on experience in backend development and modern software engineering practices. The focus was on designing robust, secure, and efficient RESTful APIs, implementing authentication and authorization mechanisms, and integrating advanced AI-driven search functionalities. This training allowed me to bridge theoretical knowledge with practical implementation through real-world use cases. The program also emphasized problem-solving, debugging, and understanding the end-to-end workflow of software development projects.

Key Takeaways

- Emphasis on **clean and modular code**, organizing helper functions, services, and routes for maintainability.
- Understanding **security best practices** for backend systems, including encryption, JWT, and account activity monitoring.
- Importance of **reusable components** to reduce redundancy and simplify future development.
- Exposure to **vector databases and AI integrations**, expanding capabilities beyond traditional backend systems.
- Real-world project experience reinforced **problem-solving, debugging, and performance optimization**, preparing me for professional backend development challenges.

Tech Stack

The **Role-Based Task Tracking System (RBTTS)** was developed using a modern, scalable, and secure technology stack to ensure efficient backend operations, robust data management, and seamless future integrations

1. Programming Language:

- **Python**: Chosen for its simplicity, readability, and strong ecosystem of libraries for web development and AI integration.

2. Web Framework:

- **FastAPI:** Used to build asynchronous RESTful APIs with automatic validation, clear routing, and high performance

3.Databases:

- **MongoDB:** Document-based database for flexible storage of users, projects, tasks, and logs.

4. Authentication & Security:

- **JWT (JSON Web Tokens):** For secure authentication and session management.
- **Password Hashing:** Ensures that user credentials are stored securely.

6. Utilities & Development Tools:

- **Git/GitHub:** Version control and collaborative development.
- **Pydantic:** For request/response data validation.
- **VS Code / IDE:** Development environment.

Project Overview

Problem Statement

Managing projects and tasks efficiently is a critical requirement for organizations of all sizes. Many teams still rely on spreadsheets, emails, or basic tools, leading to unclear task assignments, missed deadlines, and lack of transparency in project progress.

Organizations also face challenges in role management, as different team members have varying responsibilities and authority levels. Many systems fail to implement granular control over who can view, edit, or delete projects and tasks, causing inefficiencies, miscommunication, and accountability issues.

Another significant gap is the absence of **centralized monitoring and auditing**. Managers often struggle to track overall project performance, team productivity, and the history of task completion. Furthermore, with increasing volumes of tasks and projects, searching for specific information becomes time-consuming.

The consequences of these gaps include delayed project completion, misallocation of resources, decreased team efficiency, and reduced overall organizational performance. Addressing these issues is crucial for teams that aim to work systematically, transparently, and securely.

Proposed Solution

To address these challenges, the **Role-Based Task Tracking System (RBTTS)** was designed as a centralized, secure, and user-friendly platform that streamlines project and task management. RBTTS allows organizations to manage their projects efficiently while ensuring proper access control and accountability.

The system is built on a **modular backend architecture**, enabling scalability, maintainability, and future integration with advanced tools. Users can register, log in securely, and have their actions tracked based on their assigned roles, ensuring that sensitive data is protected. Role-based access ensures that team members only have permissions relevant to their responsibilities, reducing the risk of errors and unauthorized modifications.

Key Features and Functionalities:

- User Management.
- Project Management
- Task Management
- Security & Logging
- Scalability

By implementing RBTTS, organizations can improve collaboration, reduce miscommunication, maintain security, and ensure that projects are completed efficiently and on time.

. Features

1. User Management

- **Scenario:** A new team member joins the organization. They can register, choose a secure password, and get assigned a role (Admin, Manager, or Team Member). The system ensures that they can access only the functionalities allowed for their role.
- **Benefit:** Enhances security, prevents unauthorized access, and simplifies onboarding.

2. Project Management

- **Scenario:** A new team member joins the organization. They can register, choose a secure password, and get assigned a role (Admin, Manager, or Team Member). The system ensures that they can access only the functionalities allowed for their role.
- **Benefit:** Enhances security, prevents unauthorized access, and simplifies onboarding.

3. Task Management

- **Scenario:** A manager assigns a task to a developer with a specific deadline. The developer updates task status as “In Progress” and later “Completed,” which gets automatically logged.

- **Benefit:** Keeps the workflow transparent, ensures timely completion, and helps managers monitor team performance.

4. Security & Logging

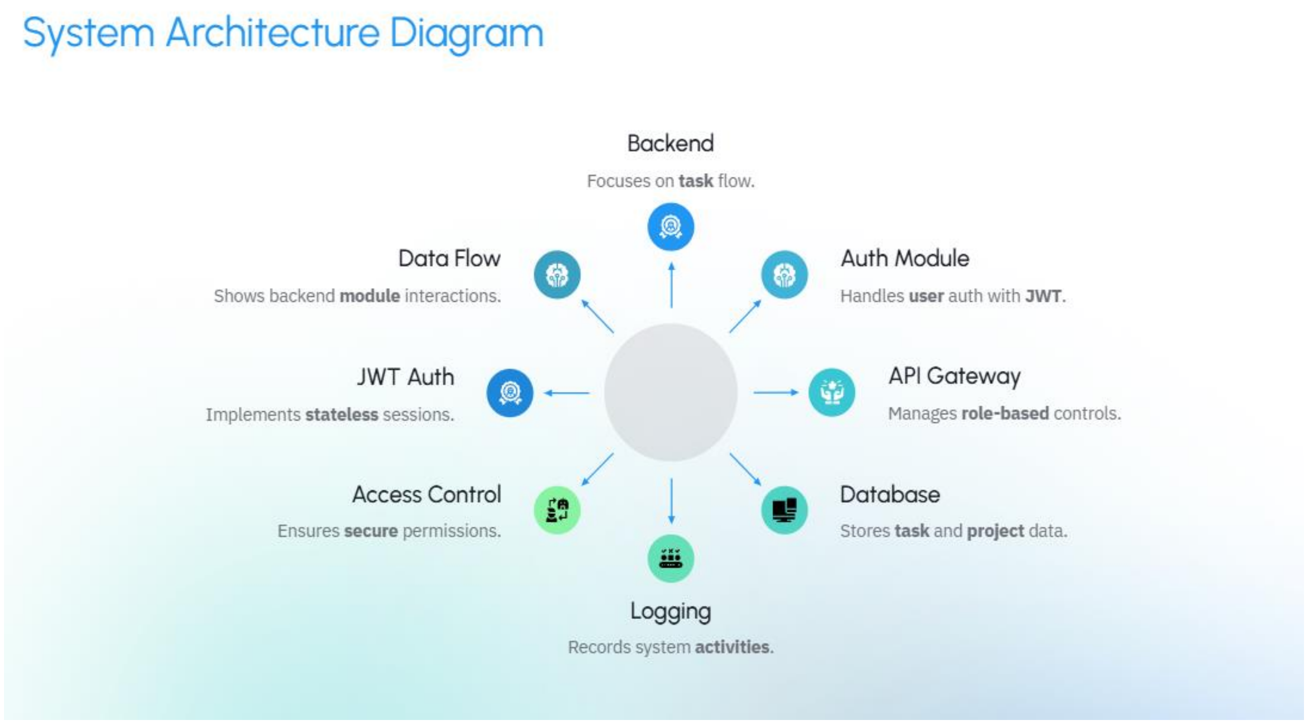
- **Scenario:** The system records all login attempts, project edits, and task updates. If there is suspicious activity, admins can review the logs to identify issues.
- **Benefit:** Maintains an audit trail, increases system security, and supports accountability.

5. Scalability & Modularity

- **Scenario:** As the organization grows, new roles, projects, and AI-powered analytics can be added without redesigning the system.
- **Benefit:** Future-proofs the system, allowing easy expansion and integration.

System Architecture Design

The Role-Based Task Tracking System (RBTTS) is designed with a modular backend architecture to ensure scalability, maintainability, and security. The system separates responsibilities across different layers, allowing efficient development and future expansion.



System Architecture Diagram
Backend
Focuses on task flow.
Data Flow
Shows backend module interactions.
Auth Module
Handles user auth with JWT.
JWT Auth
Implements stateless sessions.
API Gateway
Manages role-based controls.
Access Control
Ensures secure permissions.
Database
Stores task and project data.
Logging
Records system activities.

Backend Layers:

- **Routes / Endpoints:** Handle HTTP requests and map them to service functions.
- **Service Layer:** Contains business logic for user management, project/task operations, and role-based access control.

- **Database Layer:** Interfaces with MongoDB/MySQL for storing users, projects, tasks, and logs

MongoDB Collections:

- **users** – Stores user information, including username, hashed password, role, and creation date.
- **projects** – Contains project details, including name, description, status, owner, and assigned users.
- **tasks** – Tracks individual tasks, including task name, description, assigned user, deadline, status, and project ID.
- **login_logs** – Logs user login attempts for security monitoring and account locking.

Service Layers and Route Handling :

- **User Services:** Registration, login, password reset/change, logout, and role assignment. Handles JWT token generation and verification for session management.
- **Project Services:** Create, read, update, delete (CRUD) operations for projects. Manages project status (active, completed, archived) and user assignments.
- **Route Handling:**
    - Organized by modules (/users, /projects, /tasks)
    - Each route calls its corresponding service function.
    - Input validation is handled using **Pydantic models**, ensuring data integrity and reducing errors.

User Roles and Permissions

1. Roles Overview:

Admin:

- Full access to all system modules.
- Can create, update, and delete users, projects, and tasks.
- Manages roles and permissions for all users.

Manager:

- Can create and manage projects and tasks.
- Assigns tasks to team members and monitors progress.
- Can view user activity and project status.

Team Member:

- Can view assigned projects and tasks.
- Updates task status and adds relevant comments or notes.
- Cannot modify projects or assign tasks.

2. Benefits of Role-Based Access:

- **Security**: Sensitive information and critical operations are restricted to authorized roles only.
- **Accountability**: Every action is logged and associated with a specific user.
- **Efficiency**: Users only see features relevant to their role, reducing confusion and errors.

Project and Task Management

The **Role-Based Task Tracking System (RBTTS)** provides a structured and efficient approach to managing projects and tasks, ensuring that teams can collaborate effectively while maintaining accountability and clarity.

1.Project Management:

- **Creation and Assignment:** Managers or Admins can create new projects, define objectives, assign team members, and set deadlines.
- **Status Tracking:** Projects can be marked as **Active**, **Completed**, or **Archived**, providing a clear view of progress.
- **Monitoring:** Managers can monitor project progress through task completion status, ensuring timely delivery.
- **Updates and Modifications:** Project details can be updated as needed, and all changes are logged for transparency.

2. Task Management:

- **Task Assignment:** Tasks can be assigned to individual team members with specific deadlines and priorities.
- **Progress Tracking:** Users can update task status (e.g., Pending, In Progress, Completed), allowing managers to track performance.
- **Notifications & Alerts:** Optional notifications remind team members of deadlines or task updates.
- **Audit Trail:** Every task creation, update, or deletion is logged to maintain accountability.

3. Workflow Overview:

- Projects are created and structured into tasks.
- Tasks are assigned to users based on roles and availability.
- Users update task status and provide progress updates.
- Managers review progress, provide feedback, and adjust priorities.
- Completed projects are archived for future reference, ensuring historical data is preserved.

4. Benefits:

- Ensures **clear task ownership** and accountability.
- Reduces **miscommunication** and missed deadlines.
- Provides a **centralized view** of all projects and tasks, improving overall productivity.
- Supports **scalability**, allowing multiple projects and large teams to operate efficiently.

By implementing structured **Project and Task Management**, RBTTS ensures that teams operate systematically, tasks are completed on time, and managers have full visibility over all ongoing activities.

Development Workflow

The development of the **Role-Based Task Tracking System (RBTTS)** followed a structured workflow to ensure modularity, maintainability, and adherence to best practices. The workflow emphasized clear separation of concerns, efficient coding, and collaboration.

1. Modular Code Structure:

- **Routes / Endpoints:** Each module (Users, Projects, Tasks) has its own route definitions, making the API organized and easy to maintain.
- **Service Layer:** Contains core business logic for handling user authentication, project/task operations, and role-based access control.
- **Utility / Helper Functions:** Shared helpers for password hashing, JWT token management, and logging, promoting code reusability.
- **Database Layer:** Handles interactions with MongoDB/MySQL, ensuring efficient data storage and retrieval.

2. Version Control and Collaboration:

- **Git & GitHub:** All development was tracked using Git, with feature branching, commits, and pull requests to maintain a clean codebase.
- **Code Reviews:** Periodic reviews ensured code quality, adherence to standards, and identification of potential issues early.
- **Collaboration Practices:** Team members communicated regularly, documented features, and resolved conflicts efficiently through Git workflows.

3. Development Practices:

- **Asynchronous Programming:** FastAPI's async capabilities were leveraged to improve API performance and handle multiple requests efficiently.
- **Data Validation:** Pydantic models were used for request and response validation, ensuring data integrity.
- **Logging and Error Handling:** Comprehensive logging and structured error responses were implemented to facilitate debugging and monitoring.
- **Security Measures:** Security was integrated into development, including input validation, authentication checks, and secure storage of sensitive data.

4. Workflow Summary:

- Feature development followed a modular approach, allowing independent testing and deployment.
- New functionalities were integrated progressively, with proper validation and testing at each stage.
- Security and performance considerations were incorporated throughout the development lifecycle.
- Continuous Improvement: Feedback from testing and team reviews was incorporated iteratively to refine the system and enhance reliability.

This structured **development workflow** ensured that RBTTS is **scalable, secure, and maintainable**, while enabling smooth collaboration and consistent quality in the backend codebase.

API Endpoints Overview

The Role-Based Task Tracking System (RBTTS) provides a set of RESTful API endpoints organized by modules for Users, Projects, and Tasks. The total number of endpoints is 23, covering all functionalities from authentication to project and task management.

1. Default / Health Check

Method	Endpoint	Description
GET	/health-check	Health Check

Count :1

2. Users Module

Method	Endpoint	Description
POST	/users/register	Register a new user
POST	/users/login	User login
POST	/users/request-password-reset	Request password reset
POST	/users/change-password	Change user password
POST	/users/logout	Logout

Count :5

3. Projects Module

Method	Endpoint	Description
POST	/projects/	Create Project
GET	/projects/	List Projects
PUT	/projects/{project_id}	Update Project
GET	/projects/{project_id}	Get Project Details
DELETE	/projects/{project_id}	Delete Project

Count: 5

4. Tasks Module

Method	Endpoint	Description
POST	/tasks/	Create New Task
GET	/tasks/	List All Tasks
GET	/tasks/{task_id}	Get Task Details
PUT	/tasks/{task_id}	Update Existing Task (Admin)
DELETE	/tasks/{task_id}	Delete Existing Task
PUT	/tasks/{task_id}/assign	Assign Task
GET	/tasks/my	Get My Tasks
PUT	/tasks/{task_id}/status	Update Developer Task Status
PUT	/tasks/{task_id}/remarks	Append Developer Remarks

Method	Endpoint	Description
GET	/tasks/my-testing	Get My Testing Tasks
PUT	/tasks/{task_id}/test-status	Update Testing Status
PUT	/tasks/{task_id}/test-remarks	Append Testing Remarks

Count: 12

Total Endpoints:

Module	Number of Endpoints
Default / Health Check	1
Users	5
Projects	5
Tasks	12
Total	**23**

Testing & Validation

To ensure the Role-Based Task Tracking System (RBTTS) functions as expected, comprehensive testing was conducted covering both unit testing and API testing using Postman. This helped identify and resolve issues early, ensuring reliability, data integrity, and proper security.

1. Unit Testing:

- Individual functions and service modules were tested independently to verify their correctness.
- Examples include:
    - Password hashing and validation functions.
    - JWT token generation and verification.
    - Role-based access control checks.
- Unit tests ensured that business logic executed correctly before integrating with the API routes.
- Automated tests were run during development to quickly catch regressions after changes.

2. API Testing with Postman:

- Postman was used to test all RESTful endpoints for **Users, Projects, and Tasks** modules.
- Each endpoint was validated for:
    - Correct HTTP method usage (GET, POST, PUT, DELETE).
    - Proper request body and parameter validation.
    - Expected responses, including success and error scenarios.

- Example workflows tested using Postman:
    - User registration and login with JWT token verification.
    - Creating, updating, and retrieving projects and tasks.
    - Role-based access enforcement for Admin, Manager, and Team Member.

3. Key Observations:

- API responses were consistent with the expected schema.
- Invalid requests were handled gracefully with clear error messages.
- Security measures such as authentication, authorization, and input validation worked effectively.
- Testing ensured that tasks and projects could be managed without data inconsistencies or system crashes.

By combining **unit testing** and **Postman API testing**, the RBTTS system was validated for **functionality, reliability, and security**, ensuring a robust backend for project and task management.

Challenges & Solutions

During the development of the **Role-Based Task Tracking System (RBTTS)**, several technical and operational challenges were encountered. Each challenge provided an opportunity to improve the system and ensure robustness, security, and maintainability.

1. Role-Based Access Control Complexity

- **Challenge:** Implementing fine-grained role-based permissions for Admin, Manager, and Team Member while ensuring security at each endpoint.
- **Solution:** Developed a centralized authorization middleware using JWT tokens. Each route checks the user's role before allowing access, preventing unauthorized actions.

2. Handling Asynchronous Operations

- **Challenge:** FastAPI endpoints needed to handle multiple concurrent requests efficiently without blocking.
- **Solution:** Leveraged Python's async/await features and FastAPI's asynchronous capabilities for database queries and service functions, improving performance and responsiveness.

3. Data Validation and Integrity

- **Challenge:** Ensuring that all user input, project data, and task updates followed strict validation rules to prevent errors and inconsistencies.
- **Solution:** Used **Pydantic models** for request and response validation, enforcing data integrity and reducing runtime errors.

4. Secure Authentication and Session Management

- **Challenge:** Implementing secure login, password storage, and session management while protecting against unauthorized access.
- **Solution:** Implemented **password hashing**, JWT-based authentication, and session expiry mechanisms, along with logging of login attempts for monitoring.

5. Testing and Debugging APIs

- **Challenge:** Ensuring all endpoints worked correctly under various scenarios and roles, and identifying bugs across modules.
- **Solution:** Conducted comprehensive **unit testing** and **Postman API testing**, covering normal, edge, and error cases. This helped identify and fix issues early in the development cycle.

6. Managing Project and Task Relationships

- **Challenge:** Maintaining consistent links between projects and their tasks, especially during updates or deletions.
- **Solution:** Designed a relational mapping (MongoDB references or MySQL foreign keys) and implemented cascading updates/deletes to ensure data consistency.

By addressing these challenges systematically, the RBTTS backend became **robust, secure, and maintainable**, capable of supporting multiple users, projects, and tasks efficiently.

Future Enhancements & Scope

The **Role-Based Task Tracking System (RBTTS)** is designed to be modular and scalable, allowing for continuous improvement and feature expansion. The following enhancements are planned or recommended for future versions:

1. Frontend Integration:

- Develop a user-friendly web or mobile interface to improve accessibility and provide a seamless experience for users interacting with the system.
- Incorporate dashboards for real-time project and task visualization.

2. Advanced AI Features:

- Integrate **AI-driven task suggestions** to automatically prioritize tasks based on deadlines and workload.
- Implement **semantic search** using FAISS or other AI frameworks to quickly retrieve tasks or projects based on natural language queries.

3. Notifications and Alerts:

- Implement email or in-app notifications to alert users about task deadlines, project updates, or status changes.
- Automated reminders for pending tasks to improve workflow efficiency.

4. Analytics and Reporting:

- Provide detailed reports on project progress, team performance, and task completion trends.
- Visualize data using charts or graphs for better management insights.

5. Enhanced Security Features:

- Implement multi-factor authentication (MFA) for added security.
- Advanced logging and anomaly detection to identify suspicious activity.

6. Collaboration Tools:

- Add features for team collaboration, such as comments, file attachments, and discussion threads within projects and tasks.

By implementing these enhancements, RBTTS can evolve into a **comprehensive project and task management platform** that not only tracks work but also provides actionable insights, improves collaboration, and boosts overall productivity.

Project Outcomes & Conclusion

The **Role-Based Task Tracking System (RBTTS)** successfully provides a secure, structured, and efficient backend for managing projects and tasks across teams. By implementing role-based access control, the system ensures that Admins, Managers, and Team Members have appropriate permissions, enhancing security and accountability. Centralized project and task management allows for seamless tracking, timely updates, and efficient collaboration, reducing miscommunication and improving overall productivity. The modular architecture, asynchronous processing, and well-structured services ensure scalability, maintainability, and readiness for future enhancements.

Through comprehensive testing, including unit tests and Postman API validation, RBTTS has been verified for reliability, data integrity, and performance. The system addresses the initial problem statement effectively and provides a robust foundation for advanced features, such as AI-driven search, analytics, and notifications. Overall, RBTTS demonstrates that a thoughtfully designed backend can significantly improve workflow efficiency while providing a secure, extensible, and future-ready project and task management platform.

Impact:

- Enhanced workflow efficiency through structured task assignment and progress monitoring.

- Reduced errors and mismanagement by enforcing role-based access and centralized logging.
- Created a solid foundation for future enhancements such as AI-driven search, analytics, and notifications.

Conclusion:

The RBTTS backend demonstrates that a **well-designed, secure, and modular system** can significantly improve project and task management. The system is future-ready, scalable, and provides a robust framework for managing projects efficiently while supporting further enhancements in AI integration, reporting, and collaboration.

References / Tools Used

The **Role-Based Task Tracking System (RBTTS)** was developed using a combination of modern programming languages, frameworks, databases, and tools. Below are the official references and resources for each tool used:

Python – Backend programming language.

- Download: https://www.python.org/downloads/
- Docs: https://docs.python.org/3/

FastAPI – Framework for building RESTful APIs.

- Docs: https://fastapi.tiangolo.com/

MongoDB – NoSQL database for storing users, projects, tasks, and logs.

- Download: https://www.mongodb.com/try/download/community

Postman – API testing and workflow validation.

- Download & Docs: https://www.postman.com/downloads/

Git & GitHub – Version control and collaborative development.

- Git Docs: https://git-scm.com/doc
- GitHub Docs: https://docs.github.com/

VS Code – Development environment.

- Download & Docs: https://code.visualstudio.com/download

This list includes only the **core tools and frameworks used for RBTTS development**, providing direct references for setup, documentation, and further learning.