TASK MANAGEMENT SYSTEM API

Agile Sprint Planning & Execution Documentation

1. PRODUCT VISION

To empower developers to build productivity tools by providing a secure, scalable, and intuitive RESTful API that simplifies task creation, tracking, and management for any client application.

2. PRODUCT BACKLOG (USER STORIES)

Format:
As a [User Role], I want [Feature], So that [Benefit].

1. **User Authentication**
   As a developer integrating the API, I want to authenticate users via JWT, so that only authorized users can access their specific task data.
2. **Create Task**
   As an authenticated user, I want to create a new task with a title and description, so that I can record work that needs to be done.
3. **Retrieve Tasks**
   As an authenticated user, I want to retrieve a list of my tasks, so that I can view my current workload.
4. **Update Task**
   As an authenticated user, I want to update the status or details of a task, so that I can reflect progress or changes in requirements.
5. **Delete Task**
   As an authenticated user, I want to delete a task, so that I can remove completed or irrelevant items from my list.
6. **Logging, Monitoring & Health Check**
   As a system administrator or DevOps engineer, I want basic logging, monitoring, and a health endpoint, so that I can observe application behavior and quickly detect system issues.

3. Product BACKLOG

Story 1: User Authentication

- **Priority:** High (Must Have)
- **Estimation:** 5 Points
- **Acceptance Criteria:**
    - `POST /api/auth/register` accepts email/password and returns `201 Created`.
    - `POST /api/auth/login` validates credentials and returns a valid JWT (`200 OK`).
    - Invalid credentials return `401 Unauthorized`.
    - Passwords are securely hashed before storage.
    - Middleware protects all private routes.

Story 2: Create Task

- **Priority:** High (Must Have)
- **Estimation:** 3 Points
- **Acceptance Criteria:**
    - `POST /api/tasks` accepts title (required) and description (optional).
    - Requires valid Bearer token.
    - Successful creation returns `201 Created` with task object.
    - Task is associated with the authenticated user.
    - Empty title returns `400 Bad Request`.

Story 3: Retrieve Tasks

- **Priority:** High (Must Have)
- **Estimation:** 3 Points
- **Acceptance Criteria:**
    - `GET /api/tasks` returns a list of tasks.
    - Requires valid Bearer token.
    - Users only see their own tasks.
    - Returns `200 OK` with empty array if no tasks exist.

Story 4: Update Task

- **Priority:** Medium (Should Have)
- **Estimation:** 3 Points
- **Acceptance Criteria:**
    - `PATCH /api/tasks/:id` supports partial updates.
    - Requires valid Bearer token.
    - Users cannot update tasks they do not own (`403 Forbidden`).
    - Non-existent task returns `404 Not Found`.
    - Successful update returns `200 OK`.

Story 5: Delete Task

- **Priority:** Medium (Should Have)
- **Estimation:** 2 Points
- **Acceptance Criteria:**
    - `DELETE /api/tasks/:id` removes a task.
    - Requires valid Bearer token.
    - Cross-user deletion is forbidden.
    - Successful deletion returns `204 No Content`.
    - Non-existent task returns `404 Not Found`.

Story 6: Logging, Monitoring & Health Check

- **Priority:** Medium (Should Have)
- **Estimation:** 3 Points
- **Acceptance Criteria:**
    - Application logs incoming requests and errors.
    - Log levels (INFO, WARN, ERROR) are configurable.
    - Sensitive data is excluded from logs.
    - `GET /api/health` endpoint is publicly accessible.
    - Health endpoint returns `200 OK` with JSON service status.

4. DEFINITION OF DONE (DOD)

A story is considered “Done” when:

1. Code follows style guidelines and is merged into the develop branch.
2. Unit and integration tests pass with minimum 80% coverage.
3. API documentation is updated.
4. Input validation and security controls are implemented.
5. Code has been peer-reviewed.
6. Changes are deployed to the Staging environment.

5. SPRINT 1 PLAN

Sprint Goal:
Establish a secure foundation and core task management functionality.

Sprint Capacity: 11 Story Points

Selected Stories:

1. Story 1: User Authentication (5 Points)
2. Story 2: Create Task (3 Points)
3. Story 3: Retrieve Tasks (3 Points)

Risks & Mitigation:

- **Risk:** JWT implementation complexity
- **Mitigation:** Use well-established authentication libraries.

6. SPRINT 1 REVIEW

Outcome:
Sprint 1 successfully delivered all planned functionality and met the sprint goal.

Completed Work:

- JWT authentication implemented with secure password handling.
- Task creation and retrieval endpoints completed.
- Authorization and data isolation enforced.
- Unit and integration tests added.
- API documentation updated.

Sprint Metrics:

- Planned Points: 11
- Completed Points: 11
- Sprint Goal Achievement: **100%**

7. SPRINT 1 RETROSPECTIVE

What Went Well:

- Clear acceptance criteria reduced ambiguity.
- Early security focus simplified later development.
- Good collaboration during code reviews.

What Didn’t Go Well:

- CI pipeline setup took longer than expected.
- Infrastructure effort was underestimated.

Improvements:

- Treat CI/CD and infrastructure as backlog items.
- Improve sprint capacity planning.

8. SPRINT 2 PLAN

Sprint Goal:
Improve system observability and operational readiness.

Sprint Capacity: 3 Story Points

Selected Stories:

1. Story 6: Logging, Monitoring & Health Check (3 Points)

Risks & Mitigation:

- **Risk:** Logging exposes sensitive data
- **Mitigation:** Explicitly filter confidential fields.

9. SPRINT 2 REVIEW

Outcome:
Sprint 2 successfully enhanced the operational maturity of the system.

Completed Work:

- Structured logging implemented.
- Configurable log levels introduced.
- Health check endpoint added and documented.
- Verified public accessibility of `/api/health`.

Sprint Metrics:

- Planned Points: 3
- Completed Points: 3
- Sprint Goal Achievement: **100%**

10. FINAL RETROSPECTIVE

What Went Well:

- Strong authentication foundation enabled rapid feature delivery.
- Logging and health checks improved debugging and deployment confidence.
- Team adapted effectively to sprint re-scoping.

What Didn’t Go Well:

- Initial sprint over-commitment increased pressure.
- Infrastructure setup delayed early momentum.

Key Learnings:

- Observability is essential, even for small APIs.
- Infrastructure should be planned explicitly.
- Smaller, focused sprints improve predictability.

Overall Outcome:
The two-sprint execution delivered a secure, fully functional, and operationally observable Task Management System API. The project meets functional, security, and maintainability objectives and is ready for extension or production deployment.