AlphaBits Solutions

Assignment: TODO Application (REST APIs)

Duration

48 Hours

Objective

Design and implement a **secure, production-ready TODO application backend** using **Python OR NodeJS.**.
The application must expose **RESTful APIs**, support **authentication using JWT and session management**, and follow **good security practices**.

There are **no restrictions on frameworks or libraries**, but design decisions should be clearly documented.

Tech Stack

- **Language:** Python 3.x / Nodejs v20+
- **Framework:** Any (FastAPI / Django / Flask / Express / others)
- **Database:**
  - Required: Any database
  - **Good to have:** PostgreSQL or MongoDB
- **Authentication:**

    - JWT-based authentication
    - Session management (refresh tokens / server-side sessions)
- **Version Control:** Git (GitHub / GitLab / Bitbucket)

Core Requirements

1. Authentication & Authorization

Implement a **secure authentication system** with:

- User registration
- User login
- JWT access token generation
- Refresh token or session-based token management
- Protected routes (authorized users only)
- Secure password storage (hashing + salting)

Good to have (Bonus):

- Token expiration handling
- Logout / token invalidation
- Role-based access (e.g., user / admin)

---

2. TODO Management APIs

Each TODO item must belong to a **specific authenticated user**.

Required Endpoints

- `POST /todos` – Create a TODO
- `GET /todos` – List all user TODOs
- `GET /todos/{id}` – Get a single TODO
- `PUT /todos/{id}` – Update a TODO
- `DELETE /todos/{id}` – Delete a TODO

TODO Fields (Minimum)

- `id`
- `title`
- `description` (optional)
- `status` (pending / completed)
- `created_at`
- `updated_at`

---

3. Security Expectations

The application should demonstrate **security awareness**, including:

- Password hashing (bcrypt / argon2 / similar)
- JWT secret/key management via environment variables
- Input validation & sanitization
- Protection against:
    - Unauthorized access
    - IDOR (Insecure Direct Object Reference)
- Proper HTTP status codes
- Avoid hardcoding secrets in codebase

Good to have (Bonus):

- Rate limiting
- CORS configuration
- CSRF protection (for session-based auth)
- Security headers

---

4. Database Design

- Proper schema / collection design
- Relationship between users and TODOs
- Use ORM / ODM if applicable

- Handle database migrations (if supported by framework)

Documentation (README.md)

Your repository **must include a README** with:

1. Project overview
2. Tech stack used & reasoning
3. Setup instructions
    - Environment variables
    - Database configuration
    - How to run the project locally
4. Authentication flow explanation
5. API endpoint list (brief description)
6. Assumptions & design decisions

Deliverables

- Git repository link containing:
    - Complete source code

    - README.md

- Code should be:
    - Well-structured
    - Readable
    - Properly named modules & functions

Bonus Points

- Dockerized setup (Dockerfile / docker-compose)
- Unit or integration tests
- Pagination & filtering for TODOs
- Swagger / OpenAPI documentation
- Clean architecture / layered design
- Meaningful commit history