AI4Bharat Backend Hiring Challenge

Overview

This assignment evaluates your ability to design and implement a **production-ready backend system** with strong foundations in:

- RESTful API design
- Backend architecture
- Security best practices
- DevOps & containerization
- CI/CD automation
- Infrastructure readiness

The goal is to build a **Bug Reporting System API** suitable for real production use.

Tech Stack Requirements

Category	Requirement
Backend	Python (framework of your choice)
Containerization	Docker, Docker Compose
Orchestration	Kubernetes **or** Docker Swarm
CI/CD	GitHub Actions / GitLab CI

Objective

Design and implement a **production-ready backend API** for a bug reporting system with emphasis on:

- Clean, well-structured RESTful API design
- Containerization with Docker and orchestration readiness
- Security best practices and vulnerability mitigation
- CI/CD pipelines with automated testing
- Production-grade infrastructure configuration

This assignment evaluates backend architecture, API design, DevOps practices, security implementation, and infrastructure skills.

Context

You're building an internal bug tracker API for a team of ~50 developers across 10 projects. The system should be designed to handle production traffic, be easily deployable across different environments, and follow security best practices.

Document your framework choices, architectural decisions, and security considerations in the README.

Part 1 — API Design & Implementation

Choose any Python web framework (Django + DRF, FastAPI, Flask, Litestar, etc.). Design your schema with the following entities and constraints.

1. User Model

Field	Type	Constraints

id	UUID/Int	Primary key
username	String	Unique, max 50 chars
email	String	Unique, valid email format
password	String	Hashed (bcrypt/argon2), min 8 chars
role	Enum	developer, manager, admin
is_active	Boolean	Default: true
created_at	DateTime	Auto-set
last_login	DateTime	Nullable, updated on login

---

2. Project Model

Field	Type	Constraints
id	UUID/Int	Primary key
name	String	Unique, max 100 chars, required

description	Text	Max 1000 chars, optional
created_by	FK → User	Required, protect on delete
created_at	DateTime	Auto-set
updated_at	DateTime	Auto-update
is_archived	Boolean	Default: false

Business Rules: Soft delete via `is_archived`. Only creator or admin can archive.

3. Issue Model

Field	Type	Constraints
id	UUID/Int	Primary key
title	String	Max 200 chars, required
description	Text	Max 5000 chars, markdown supported
status	Enum	open, in_progress, resolved, closed, reopened

priority	Enum	low, medium, high, critical
project	FK → Project	Required, cascade on delete
reporter	FK → User	Required, protect on delete
assignee	FK → User	Nullable, set null on delete
due_date	Date	Optional
created_at	DateTime	Auto-set
updated_at	DateTime	Auto-update

Status Transition State Machine

open → in_progress → resolved → closed | reopened ← (from resolved or closed)

Business Rules: Valid status transitions must follow state machine. Critical issues cannot be closed without at least one comment.

4. Comment Model

Field	Type	Constraints
id	UUID/Int	Primary key

content	Text	Max 2000 chars, required, non-empty
issue	FK → Issue	Required, cascade on delete
author	FK → User	Required, protect on delete
created_at	DateTime	Auto-set
updated_at	DateTime	Auto-update

Business Rules: Comments cannot be deleted (audit trail).

API Endpoints

Authentication

Method	Endpoint	Description
POST	/api/auth/register	Register new user
POST	/api/auth/login	Obtain tokens (access + refresh)
POST	/api/auth/refresh	Refresh access token
POST	/api/auth/logout	Invalidate refresh token

GET	/api/auth/me	Get current user profile

Requirements: JWT-based authentication. Access token expiry: 15 minutes. Refresh token expiry: 7 days. Implement token blacklisting for logout.

Projects

Method	Endpoint	Description	Auth
GET	/api/projects	List projects	Required
POST	/api/projects	Create project	Required
GET	/api/projects/{id}	Get project details	Required
PATCH	/api/projects/{id}	Update project	Owner/Admin
DELETE	/api/projects/{id}	Archive project	Owner/Admin

Query Params: `?search=&is_archived=false&page=&limit=&sort=`

Issues

Method	Endpoint	Description	Auth

GET	/api/projects/{id}/issues	List project issues	Required
POST	/api/projects/{id}/issues	Create issue	Required
GET	/api/issues/{id}	Get issue details	Required
PATCH	/api/issues/{id}	Update issue	Reporter/Assignee/Owner

Query Params: `?status=&priority=&assignee=&search=&sort=&page=&limit=`

Comments

Method	Endpoint	Description	Auth
GET	/api/issues/{id}/comments	List comments	Required
POST	/api/issues/{id}/comments	Add comment	Required
PATCH	/api/comments/{id}	Edit comment	Author only

Permission Matrix

Action	Anon	Developer	Reporter	Assignee	Manager	Admin
View projects	No	Yes	Yes	Yes	Yes	Yes
Create project	No	No	No	No	Yes	Yes
Edit/Archive project	No	No	No	No	Yes	Yes
View issues	No	Yes	Yes	Yes	Yes	Yes
Create issue	No	Yes	Yes	Yes	Yes	Yes
Edit issue	No	No	Yes	Yes	Yes	Yes
Change assignee	No	No	Yes	No	Yes	Yes
Add comment	No	Yes	Yes	Yes	Yes	Yes

Implement permissions using middleware/decorators — not inline view logic.

Part 2 — Security Requirements

Implement comprehensive security measures to protect the API against common vulnerabilities.

Authentication & Session Security

Password Security: Use bcrypt or Argon2 for hashing. Enforce minimum 8 characters with complexity rules (uppercase, lowercase, number, special character).
JWT Security: Use RS256 or ES256 algorithm (asymmetric). Short-lived access tokens (15 min). Secure refresh token rotation. Implement token blacklisting for logout.
Brute Force Protection: Implement rate limiting on login endpoint (e.g., 5 attempts per minute). Account lockout after repeated failures.
Session Management: Invalidate all sessions on password change. Track active sessions per user. Implement 'logout all devices' functionality.

Input Validation & Injection Prevention

SQL Injection: Use parameterized queries or ORM. Never concatenate user input into queries. Validate and sanitize all inputs.
XSS Prevention: Sanitize HTML in markdown fields. Escape output in API responses. Implement Content-Security-Policy headers.
Request Validation: Validate Content-Type headers. Limit request body size (e.g., 1MB). Validate and whitelist query parameters.
Path Traversal: Validate file paths if any file operations. Whitelist allowed characters in URL parameters.

API Security Headers & CORS

Security Headers: X-Content-Type-Options: nosniff, X-Frame-Options: DENY, X-XSS-Protection: 1; mode=block, Strict-Transport-Security (HSTS)
CORS Configuration: Whitelist specific origins (no wildcards in production). Limit allowed methods and headers. Set appropriate max-age for preflight caching.
Rate Limiting: Global rate limit (e.g., 100 requests/minute per IP). Endpoint-specific limits for sensitive operations. Return Retry-After header on 429 responses.

Data Protection

Sensitive Data: Never log passwords or tokens. Mask sensitive fields in responses. Implement field-level encryption for PII if required.
Error Handling: Generic error messages to clients (no stack traces). Detailed logging server-side. Different error detail levels for dev/prod.
Audit Logging: Log all authentication events. Log permission-sensitive operations. Include timestamp, user ID, IP, action, resource.

Dependency Security

Use tools like safety, pip-audit, or Snyk to scan for vulnerable dependencies.
Pin dependency versions in requirements.txt or pyproject.toml.
Implement automated dependency updates with security scanning in CI/CD.

Part 3 — Containerization & Infrastructure

Docker Configuration

Create production-ready Docker configuration:

Dockerfile: Multi-stage build for smaller image size. Non-root user for security. Proper layer caching for dependencies. Health check instruction.
docker-compose.yml: Service definitions for API, database, Redis (for caching/sessions). Named volumes for data persistence. Environment variable configuration. Network isolation between services.
docker-compose.override.yml: Development-specific overrides. Volume mounts for hot reloading. Debug ports exposed.

Nginx Configuration

Implement Nginx as reverse proxy with:

Load Balancing, SSL/TLS, Security Hardening, Performance optimizations (Gzip, caching, buffer tuning).

Container Orchestration

Provide configuration for **ONE**:

Option A: Kubernetes
Deployment, Service, ConfigMap & Secrets, Ingress, HPA.

Option B: Docker Swarm
Stack file, networking, secrets, health checks, update config.

Part 4 — CI/CD Pipeline

Implement a comprehensive CI/CD pipeline using GitHub Actions or GitLab CI.

Pipeline Stages

1. Code Quality Stage
2. Testing Stage
3. Build Stage
4. Deploy Stage (Bonus)

Include linting, formatting, type checking, security scanning, unit tests (≥70% coverage), integration tests, Docker image scanning, registry push, and optional deployment.

Part 5 — Additional Requirements

API Documentation (Swagger/OpenAPI), Logging & Monitoring, Database Migrations, Load Testing (Bonus), Error Response Format, Seed Data Script.

Submission Guidelines

Follow the required repository structure, include README with architecture and security documentation, and ensure CI passes on main branch. Tag final submission version.

Clarifications & FAQ

Use any Python framework. Choose **one** orchestration platform. PostgreSQL recommended. Focus on listed security requirements. Prioritize quality over quantity.