# Project Design Document: JIRA-like Workforce & Work Management Application

## 1. Introduction

This document describes the design options, architecture, and implementation strategy for a **JIRA-like application** aimed at improving organizational productivity by enabling efficient work tracking, task distribution, and workforce management. The application integrates modern UI/UX, role-based permissions, AI-powered suggestions, and performance/KRA reporting to provide a comprehensive platform for employees, team leads, and administrators.

## 2. Goals & Objectives

* Provide a **centralized platform** for task, story, and resource management.
* Enable **role-based access control** for Users, Team Leads, and Admins.
* Support **story lifecycle management** with features like priority, status, comments, and acceptance criteria.
* Facilitate **team lead controls** (task assignment, notes, distribution summaries).
* Offer **KRA-driven reporting** for employees.
* Integrate an **AI agent** for task/story suggestions, effort estimation, and intelligent prioritization.
* Ensure scalability, security, and extensibility for enterprise adoption.

## 3. User Roles & Permissions

### User (Employee)

* Create/edit personal tasks/stories.
* Log time and track progress.
* Comment and collaborate with team members.
* View personal and team boards.

### Team Lead

* Create and assign stories for team members.
* Track team workload and progress.
* Add private notes and distribution summaries.
* Generate KRA summaries for team.
* Approve acceptance criteria and monitor blockers.

### Admin

* Global configuration: projects, users, teams.
* Manage access (SSO/AD, role provisioning).
* Configure integrations (GitHub, Slack, Calendar).
* Audit logs and compliance.

### Optional Roles

* **Project Manager**: Cross-team coordination, release planning.
* **Observer**: Read-only access.

## 4. Core Features

### Stories & Tasks

* Title, description, acceptance criteria.
* Estimated points & hours.
* Priority (Low, Medium, High, Urgent).
* Status (Backlog → Done lifecycle).
* Comments, attachments, and labels.
* Parent/child story linking.

### Time Tracking

* Start/stop timer on tasks.
* Manual time log entries.
* Auto-calculated effort from logs.

### Team Lead Tools

* Isolated notes & summaries.
* Bulk assignment actions.
* Team-level workload distribution.

### KRA & Reporting

* User summaries: completed stories, velocity, logged hours.
* Exportable KRA packets for performance reviews.
* Team heatmaps for workload and throughput.

### AI Agent

* Suggest story titles & acceptance criteria.
* Predict effort (hours/story points).
* Recommend assignees based on workload & skills.
* Detect anomalies (e.g., unusually long status times).

### Special Task Types

* **Adhoc tasks**: lightweight quick logs.
* **Call stories**: for meetings/incident handling.
* **Helping tasks**: linked to parent story.

### Integrations

* **AD/SSO** (Okta, Azure AD).
* **Slack** (notifications, quick-create tasks).
* **GitHub/GitLab** (auto-close on PR merge).
* **Calendar sync** (meetings, deadlines).

## 5. Recommended Additional Features

1. **Workload Heatmap**: Visualize per-user allocation.
2. **Cycle Time Analytics**: Track bottlenecks.
3. **SLA Management**: Escalations for overdue items.
4. **Cross-project Dashboard**: Unified view for managers.
5. **Exportable KRA Reports**: CSV/PDF summaries.

## 6. System Design Options

### Option A: Monolithic Architecture

* **Stack**: Django/Flask + PostgreSQL + React.
* Pros: Simple, faster MVP.
* Cons: Scaling issues, harder to separate AI microservices.

### Option B: Microservices Architecture (Recommended)

* **Frontend**: React + TailwindCSS.
* **Backend**: FastAPI (Python) for core services.
* **Database**: PostgreSQL for structured data.
* **AI Service**: Dedicated Python microservice with ML models.
* **Message Queue**: RabbitMQ/SQS for async processing.
* **File Storage**: AWS S3 for attachments.
* **Hosting**: AWS ECS/EKS or GCP GKE.
* Pros: Scalable, modular, AI-friendly.
* Cons: Higher initial complexity.

### Option C: Serverless-first

* **Frontend**: React hosted on S3/CloudFront.
* **Backend**: AWS Lambda + API Gateway.
* **DB**: DynamoDB.
* Pros: Pay-per-use, minimal infra management.
* Cons: Harder for complex workflows.

## 7. Data Model (High-Level)

### Users

* id, name, email, role, teams, skills, manager\_id.

### Teams

* id, name, lead\_id, members.

### Projects

* id, name, key, lead\_team\_id.

### Stories

* id, project\_id, title, description, type, status, priority.
* story\_points, estimated\_hours, logged\_hours.
* acceptance\_criteria, labels, assignees.

### Time Logs

* id, story\_id, user\_id, start, end, duration.

### Comments

* id, story\_id, user\_id, body, is\_private.

### AI Insights

* id, story\_id, suggestion\_type, value, confidence.

## 8. API Endpoints (Sample)

* POST /auth/login → JWT token.
* GET /users/{id}/summary → KRA view.
* POST /projects/{id}/stories → Create story.
* GET /stories/{id} → Fetch story detail.
* PATCH /stories/{id} → Update fields.
* POST /stories/{id}/comment → Add comment.
* POST /stories/{id}/timelog → Add time log.
* POST /ai/suggest → AI-powered suggestion.

## 9. Permissions Matrix

| Feature | User | Team Lead | Admin |
| --- | --- | --- | --- |
| Create personal task | ✔️ | ✔️ | ✔️ |
| Assign tasks | ❌ | ✔️ | ✔️ |
| View team workload | ❌ | ✔️ | ✔️ |
| Private notes | ❌ | ✔️ | ✔️ |
| Reports (KRA/team) | ✔️ (own) | ✔️ (team) | ✔️ (all) |
| Manage users/roles | ❌ | ❌ | ✔️ |

## 10. UI & UX Design

### Key Screens

1. **Dashboard**: My tasks, team summary, AI suggestions.
2. **Backlog**: Filterable list of stories.
3. **Kanban Board**: Drag-and-drop workflow.
4. **Story Modal**: Details, comments, AI panel.
5. **Reports**: User/team KRAs, cycle time charts.
6. **Admin Console**: User management, integrations.

### UI Principles

* Minimalist, distraction-free.
* Drag-and-drop for boards.
* Inline edits.
* Accessibility (keyboard shortcuts, ARIA labels).

## 11. AI Agent Design

### Capabilities

* **Story drafting**: Convert notes to structured stories.
* **Effort estimation**: Predict hours/points.
* **Assignee recommendation**: Match based on skills & workload.
* **Priority detection**: Identify urgent tasks.
* **Anomaly detection**: Highlight unusual delays.

### Architecture

* **Data**: Historical stories, time logs.
* **Models**: Text embeddings (transformers) + regression for estimates.
* **Service**: Python FastAPI microservice.
* **Integration**: Backend calls AI service for suggestions.

## 12. Deployment & Hosting

### Cloud Infra (AWS Example)

* **Frontend**: React app on S3 + CloudFront.
* **Backend**: FastAPI on ECS (Fargate).
* **DB**: RDS PostgreSQL.
* **Cache**: Redis for sessions, caching AI suggestions.
* **File storage**: S3 for attachments.
* **CI/CD**: GitHub Actions → Docker builds → Deploy.
* **Monitoring**: CloudWatch, Prometheus + Grafana.

### Security

* SSO + MFA for login.
* RBAC enforcement.
* Audit logs for sensitive actions.
* TLS + encryption at rest.

## 13. MVP Roadmap

### Phase 1 (4–8 weeks)

* User auth, roles, projects.
* Story CRUD with Kanban board.
* Time logging & comments.
* Basic reporting.
* Simple AI (prompt-based suggestions).

### Phase 2 (8–12 weeks)

* Advanced reporting (cycle time, workload heatmap).
* AI-trained models (estimation, assignment).
* Slack & GitHub integrations.
* Exportable KRA packets.

### Phase 3 (Ongoing)

* Advanced analytics (SLA, anomalies).
* Enterprise integrations (AD/Okta SCIM).
* Cross-project dashboards.
* Scalability enhancements.

## 14. Conclusion

This design outlines a **scalable, AI-powered, JIRA-like application** tailored for workforce and task management. With modular architecture, role-based permissions, and AI integration, the platform will enhance productivity, transparency, and reporting across organizations.

The recommended approach is a **microservices-based architecture (Option B)** for scalability and AI extensibility, starting with an MVP focused on core workflows and gradually expanding into advanced features.