Todo Full-Stack Web Application

Phase II - V : Complete Project Report

Author: Syed Faizan Mustafa

Course: Spec-Kit Plus - Agentic AI

Project: Q4 Hackathon 02

Date: February 18, 2026

Repository: github.com/Syed-Faizan-Mustafa

AI-Powered Todo Application with Microservices, Kafka, Dapr & Kubernetes

Table of Contents

1. Executive Summary

This report documents the Todo Full-Stack Web Application, a production-grade task management system built across multiple development phases (Phase II through Phase V). The application features an AI-powered chatbot interface, microservices architecture, event-driven communication via Apache Kafka, Dapr runtime integration, and Kubernetes deployment on Minikube.

The project demonstrates modern full-stack development practices including: Next.js 14 App Router frontend, FastAPI async backend, PostgreSQL database (Neon serverless), JWT-based authentication with Better Auth, natural language task management via Cohere LLM, and a comprehensive microservices ecosystem with 6 independently deployable services.

Key Achievements

- Full CRUD task management with AI chatbot (natural language interface)
- 6 microservices deployed on Kubernetes with Dapr sidecars
- Apache Kafka event streaming for real-time inter-service communication
- Helm chart-based deployment with automated scripts
- Advanced features: priorities, tags, search, filter, sort, recurring tasks
- Client-side and server-side filtering with debounced search
- 10 MCP tools for AI-driven task operations
- Production deployment on Vercel (frontend) + HF Spaces (backend)
- Local cloud deployment on Minikube with all services running

2. Technology Stack

Frontend

Framework:	Next.js 14 (App Router)
Language:	TypeScript 5.x
Styling:	TailwindCSS + Shadcn/UI
State:	React Query (TanStack Query)
Auth Client:	Better Auth (client SDK)
ORM:	Prisma (for Better Auth sessions)
Hosting:	Vercel (Production) / Minikube (Local)

Backend

Framework:	FastAPI (Python 3.11)
ORM:	SQLModel + SQLAlchemy (async)
Database:	PostgreSQL (Neon Serverless)
Auth:	JWT Bearer Token Verification
Hosting:	HuggingFace Spaces / Minikube

AI / LLM

LLM Provider:	Cohere (command-r-plus)
Protocol:	Model Context Protocol (MCP)
Agent Pipeline:	Intent Analyzer > MCP Executor > Response Composer

Infrastructure

Container Runtime:	Docker (multi-stage builds)
Orchestration:	Kubernetes (Minikube)
Package Manager:	Helm 3
Service Mesh:	Dapr 1.16.x (5 building blocks)
Message Broker:	Apache Kafka (Confluent)
CI/CD:	GitHub Actions (planned)

3. System Architecture

The application follows a microservices architecture pattern with event-driven communication. The system consists of 6 independently deployable services, connected via Apache Kafka for asynchronous event streaming and Dapr for service-to-service invocation.

Architecture Overview

User Browser --> Next.js Frontend (Port 3000)
--> /api/tasks (proxy) --> FastAPI Backend (Port 8000) --> Neon PostgreSQL
--> /api/chat (AI) --> Cohere LLM --> MCP Tools --> Backend API
Backend --> Kafka Events --> Notification Service
Backend --> Kafka Events --> Audit Service
Backend --> Kafka Events --> Recurring Task Service
Backend --> Kafka Events --> WebSocket Service --> Browser (real-time)

Service Communication Patterns

- Synchronous: REST API (Frontend <-> Backend)
- Asynchronous: Kafka PubSub via Dapr (Backend -> Microservices)
- Real-time: WebSocket (WebSocket Service -> Browser)
- State Management: Dapr State Store (PostgreSQL-backed)

4. Frontend (Next.js 14)

The frontend is built with Next.js 14 using the App Router pattern. It provides a modern, responsive task management UI with an integrated AI chatbot for natural language interactions.

Key Pages & Routes

- / - Landing page with app overview
- /signin - Email/password sign-in with Better Auth
- /signup - User registration
- /tasks - Main task management dashboard
- /api/tasks - Proxy API routes to FastAPI backend
- /api/chat - AI chatbot endpoint (Cohere + MCP)
- /api/auth/[...all] - Better Auth API routes

Features

- Task CRUD with inline editing and completion toggle
- AI Chatbot sidebar for natural language task management
- Filter bar: All/Pending/Done status, priority, overdue, search
- Sort: by date, priority, title (ascending/descending)
- Tag management with color-coded badges
- Due date picker with overdue highlighting
- Responsive design with dark/light mode ready
- Client-side filtering with debounced search (300ms)

5. Backend (FastAPI)

The backend is an async FastAPI application providing RESTful API endpoints for task management. It uses SQLModel ORM with async SQLAlchemy for database operations and verifies JWT tokens issued by Better Auth for authentication.

API Endpoints

Method	Endpoint	Description
GET	/api/v1/tasks	List user tasks (with filters)
POST	/api/v1/tasks	Create new task
GET	/api/v1/tasks/{id}	Get task by ID
PUT	/api/v1/tasks/{id}	Full update task
PATCH	/api/v1/tasks/{id}	Partial update task
DELETE	/api/v1/tasks/{id}	Delete task
GET	/health	Health check endpoint

Security

- JWT Bearer token verification on all /api/v1/ routes
- User data isolation (user_id from JWT sub claim)
- CORS middleware with configurable origins
- Rate limiting middleware

6. AI Chatbot Integration

The AI chatbot provides a natural language interface for task management. Users can create, update, complete, delete, and query tasks using conversational English or Urdu/Roman Urdu. The system uses a 3-stage agent pipeline.

Agent Pipeline

1. Intent Analyzer - Analyzes user message to determine intent (add_task, complete_task, update_task, delete_task, list_tasks, set_due_date, set_priority, add_tags, incomplete_task, general_chat) and extracts entities (task title, ID, ordinal reference).

2. MCP Tool Executor - Maps intent to the appropriate MCP tool and executes it against the backend API with the user's JWT token.

3. Response Composer - Takes the tool result and composes a friendly, contextual response with suggested follow-up actions.

Supported Intents

Intent	Description
add_task	Create a new task from natural language
list_tasks	List all tasks or filtered subset
complete_task	Mark a task as completed
incomplete_task	Mark a completed task as pending
update_task	Update task title or description
delete_task	Delete a task permanently
set_due_date	Set or update task due date
set_priority	Set task priority (high/medium/low)
add_tags	Add tags to a task
general_chat	General conversation / help

7. MCP Tools (Model Context Protocol)

The application implements 10 MCP tools that serve as the bridge between the AI chatbot and the backend API. Each tool has a defined schema, input validation, and error handling.

Tool Name	Description
add_task	Create a new task with title, description, priority, tags, due date
list_tasks	List tasks with optional status, priority, tag filters
complete_task	Mark a task as completed (by ID or ordinal)
incomplete_task	Mark a task as incomplete/pending
update_task	Update task fields (title, description, etc.)
delete_task	Permanently delete a task
set_due_date	Set or update the due date of a task
set_priority	Set priority level (high, medium, low)
add_tags	Add tags to a task (merges with existing)
remove_tags	Remove specific tags from a task

8. Phase 5 - Advanced Features

Part A: Advanced Task Features

- Priority levels: high, medium, low (with validation)
- Tags: up to 10 per task, alphanumeric + hyphens, max 30 chars each
- Due dates with overdue detection and highlighting
- Reminder timestamps (remind_at field)
- Recurring tasks: daily, weekly, monthly patterns with configurable intervals
- Parent-child task relationships for recurring task chains
- Full-text search across title and description (client-side)
- Multi-criteria filtering: status, priority, tags, overdue, search
- Sortable columns: created date, priority, title, due date

Part B: Microservices & Kubernetes

- 4 new microservices: Notification, Recurring Task, Audit, WebSocket
- Apache Kafka for event streaming between services
- Dapr runtime with PubSub, State Store, Service Invocation
- Helm chart packaging with configurable values
- Minikube local deployment with all 8 pods running
- Docker multi-stage builds for optimized images

9. Microservices Architecture

Service	Framework	Port	Dapr	Responsibility
Frontend	Next.js 14	3000	No	User interface, API proxy, AI chat
Backend	FastAPI	8000	Yes	Task CRUD, JWT auth, Kafka producer
Notification	FastAPI	8080	Yes	Email/push notifications on events
Audit	FastAPI	8080	Yes	Event logging and audit trail
Recurring Task	FastAPI	8080	Yes	Scheduled task regeneration
WebSocket	FastAPI	8080	Yes	Real-time updates to browser
Kafka	Confluent	9092	No	Event message broker
Zookeeper	Apache	2181	No	Kafka cluster coordination

Kafka Topics

Topic	Description
task.created	New task created event
task.updated	Task updated event
task.completed	Task marked complete
task.deleted	Task deleted event
task.reminder	Reminder due notification

10. Event-Driven Architecture (Kafka)

Apache Kafka serves as the central event bus for inter-service communication. When the backend performs task operations, it publishes events to Kafka topics. Downstream microservices consume these events asynchronously via Dapr PubSub.

Kafka Configuration

Broker:	Confluent Kafka (single broker)
Replication Factor:	1 (single node Minikube)
Auto-Create Topics:	Enabled
Log Retention:	168 hours (7 days)
Heap Memory:	256MB (-Xmx256m -Xms256m)
Storage:	1Gi PersistentVolumeClaim

Event Flow

1. User creates/updates/deletes task via Frontend
2. Frontend proxy calls Backend API
3. Backend persists to PostgreSQL and publishes Kafka event
4. Dapr PubSub delivers event to subscribed microservices
5. Notification Service sends alerts
6. Audit Service logs the event
7. WebSocket Service pushes real-time update to browser

11. Dapr Integration

Dapr (Distributed Application Runtime) v1.16.x provides portable, event-driven runtime building blocks. Each service (except Frontend) runs with a Dapr sidecar container that handles service communication, state management, and pub/sub messaging.

Dapr Building Blocks Used

Building Block	Component Type	Usage
PubSub	pubsub.kafka	Event streaming via Kafka topics
State Store	state.postgresql	Distributed state in PostgreSQL
Service Invocation	Built-in	Service-to-service HTTP calls
Secrets	kubernetes	Secret management via K8s secrets

Dapr Components

- kafka-pubsub: Kafka broker connection for async messaging
- statestore: PostgreSQL-backed state store for Dapr actors

12. Kubernetes & Helm Deployment

Minikube Configuration

Driver:	Docker (WSL2)
CPUs:	2
Memory:	4096 MB
Kubernetes:	v1.32.0
Dapr:	v1.16.9

Namespaces

- todo-app: Frontend, Backend, 4 Microservices
- kafka: Kafka broker, Zookeeper
- dapr-system: Dapr control plane (8 pods)
- kube-system: Core Kubernetes components

Helm Chart Structure

helm/todo-app/
  Chart.yaml - Chart metadata (v0.2.0)
  values.yaml - Default configuration
  values-local.yaml - Minikube overrides
  templates/
    backend-deployment.yaml
    frontend-deployment.yaml
    microservices-deployment.yaml
    kafka-statefulset.yaml
    zookeeper-statefulset.yaml
    dapr-components.yaml
    secrets.yaml
    _helpers.tpl

Docker Images (Local)

Image	Tag	Size	Base
todo-frontend	latest	~311 MB	Node 20 Alpine (3-stage)
todo-backend	latest	~219 MB	Python 3.11 Slim
todo-notification	latest	~180 MB	Python 3.11 Slim
todo-audit	latest	~180 MB	Python 3.11 Slim
todo-recurring	latest	~180 MB	Python 3.11 Slim
todo-websocket	latest	~180 MB	Python 3.11 Slim

13. Database Schema

The application uses Neon PostgreSQL (serverless) as its primary database. The schema includes Better Auth tables for authentication and the tasks table for task management.

Tasks Table (16 columns)

Column	Type	Description
id	UUID	Primary key (auto-generated)
user_id	VARCHAR(255)	Owner's user ID (indexed)
title	VARCHAR(255)	Task title (required)
description	VARCHAR(2000)	Task description (optional)
completed	BOOLEAN	Completion status (default: false)
priority	VARCHAR(10)	high / medium / low (default: medium)
tags	TEXT[]	Array of tags (max 10)
due_date	TIMESTAMP	Due date (optional)
remind_at	TIMESTAMP	Reminder time (optional)
reminder_sent	BOOLEAN	Reminder sent flag (default: false)
is_recurring	BOOLEAN	Recurring flag (default: false)
recurrence_pattern	VARCHAR(10)	daily / weekly / monthly
recurrence_interval	INTEGER	Recur every N periods (default: 1)
parent_task_id	UUID (FK)	Parent task reference
created_at	TIMESTAMP	Creation timestamp
updated_at	TIMESTAMP	Last update timestamp

Better Auth Tables

- user: id, email, emailVerified, name, image, createdAt, updatedAt
- session: id, token, expiresAt, ipAddress, userAgent, userId
- account: id, accountId, providerId, userId, accessToken, refreshToken
- verification: id, identifier, value, expiresAt

14. Authentication & Security

Authentication Flow

1. User signs in via Better Auth (email + password)
2. Better Auth creates a session and sets an HTTP-only cookie
3. Frontend API routes extract the session token from the cookie
4. Frontend generates a JWT and forwards it to the FastAPI backend
5. Backend verifies the JWT signature and extracts user_id
6. All database queries are scoped to the authenticated user_id

Security Measures

- HTTP-only cookies for session tokens (XSS protection)
- JWT Bearer tokens for backend API authentication
- User data isolation at database query level
- CORS middleware with whitelisted origins
- Trusted origins configuration for Better Auth CSRF protection
- Kubernetes secrets for sensitive environment variables
- No hardcoded secrets (all via .env or K8s secrets)

15. Deployment Status

Production (Vercel + HF Spaces)

Frontend:	Deployed on Vercel (Next.js)
Backend:	Deployed on HuggingFace Spaces (FastAPI)
Database:	Neon PostgreSQL (serverless)
Status:	LIVE - All services operational

Local Cloud (Minikube)

Pod	Ready	Status	Details
frontend	1/1	Running	Next.js 14
backend	2/2	Running	FastAPI + Dapr sidecar
audit-service	2/2	Running	Audit + Dapr sidecar
notification-service	2/2	Running	Notification + Dapr sidecar
recurring-task-service	2/2	Running	Recurring + Dapr sidecar
websocket-service	2/2	Running	WebSocket + Dapr sidecar
kafka-0	1/1	Running	Confluent Kafka broker
zookeeper-0	1/1	Running	Apache Zookeeper

Exposed Services

- Frontend: NodePort 30000 (via kubectl port-forward)
- Backend: NodePort 30080
- WebSocket: NodePort 30088

16. Git History

Hash	Message
6a418b5	fix: add Minikube NodePort origins to Better Auth trustedOrigins
4a14515	fix: client-side filtering, incomplete_task intent, Minikube fixes
65e0696	feat: Phase 5 Part A + B - Advanced features, microservices, Kafka, Dapr
17b0332	feat: Phase 4 - Docker + Helm local cloud deployment on Minikube
13b2d44	fix: update/set_due_date improvements and proper task reference handling
fe0cdd5	fix: add pattern-based detection for complete, update, delete intents
17293b5	fix: show API response even when success flag is false
c22da82	debug: add session logging to chat API route
6ae0df0	fix: use consistent backend URL in all MCP tools
a6efec8	fix: add credentials include to chat API fetch for session auth

17. Challenges & Solutions

Kafka CrashLoopBackOff on Minikube

Kubernetes auto-injects KAFKA_PORT env var from the service named 'kafka'. Confluent Docker image interprets this as deprecated config and exits. Fixed with enableServiceLinks: false on the pod spec.

Better Auth 403 FORBIDDEN on Minikube

Minikube serves on random tunnel ports not in Better Auth's trustedOrigins. Fixed by adding localhost:30000 and 127.0.0.1:30000 to the trusted origins list.

Database Schema Mismatch

Phase 5 backend model has 16 columns but Neon DB had only 7 (Phase 2 schema). Fixed by running ALTER TABLE migrations to add 9 missing columns with defaults.

Docker Build TypeScript Strictness

Docker builds use stricter TS checks than dev mode. Set spread on Set objects and ESLint rule references failed. Fixed with Array.from() and generic eslint-disable.

WSL2 + Minikube Port Access

NodePort services not accessible from Windows localhost in WSL2. Resolved with kubectl port-forward --address=0.0.0.0.

Dapr Component Schema Changes

Dapr v1.16.x CRD doesn't support 'scopes' field in Component spec. Removed scopes from component definitions.

18. Future Roadmap

Phase 5 Part C (Planned)

- Cloud deployment on AKS / GKE / OKE
- GitHub Actions CI/CD pipelines
- Prometheus + Grafana monitoring stack
- Loki for log aggregation
- Jaeger for distributed tracing
- Alerting rules and dashboards

Feature Enhancements

- Email notifications for due date reminders
- Real-time task updates via WebSocket
- Task sharing and collaboration
- File attachments on tasks
- Task templates and bulk operations
- Mobile-responsive PWA support

End of Report

Generated on February 18, 2026 at 02:28