Project Report
Social Media Post Scheduler

Madhumathi
22b0924

Project Summary

The Social Media Post Scheduler is a compact, demo-ready application that enables users to schedule social posts for one or more platforms and automatically publishes them at the specified time. Designed with a focus on clarity, extensibility and interview demonstration value, the project emphasizes persistence, reliable background execution, and pluggable adapters for platform integration.

Business Objective

- Reduce manual posting effort by enabling scheduled publishing.
- Provide a simple UI for marketers to queue and review posts.
- Offer a safe demo environment that can later be extended to real social APIs.

Key Features

- **Schedule posts:** create posts with content, target platforms, and scheduled time.
- **Persistent storage:** scheduled items are stored reliably to survive restarts.
- **Background publishing:** a scheduler picks up due posts and publishes them automatically.
- **Pluggable adapters:** platform-specific posting logic is isolated so real APIs can be added without changing core logic.
- **Admin UI:** quick web interface to create, view and refresh scheduled posts for demo purposes.
- **Status tracking:** each entry records ‘pending‘, ‘posted‘, or ‘failed‘ with an execution result string for debugging and audit.

Technical Architecture

- **API Layer:** lightweight REST endpoints for scheduling and listing posts.
- **Persistence:** file-based relational store (SQLite) for ease of setup and portability.
- **Scheduler Worker:** an in-process background job runner polls for due posts and invokes adapters.
- **Adapters:** adapter interface handles posting to each platform (mockable for demo, replaceable with real API clients).
- **Frontend:** minimal web UI that interacts with API endpoints to schedule and display posts.

Technology Stack

- Backend: modern Python web framework (FastAPI) for rapid API development.
- Scheduling: a reliable scheduler library for background jobs (demo uses a lightweight in-process scheduler).
- Storage: SQLite for simple, portable persistence in demo and local environments.
- Frontend: minimal HTML/JavaScript for immediate demoability; Charting or richer UI can be added later.

Design Decisions & Tradeoffs

- **In-process scheduler** (APScheduler or equivalent) chosen for quick demos and simple deployment. For production, an external worker system (e.g., Celery + Redis) is recommended for reliability, retries, and horizontal scalability.
- **SQLite** selected for fast setup and portability; migrating to Postgres is straightforward when moving to a multi-node deployment.
- **Pluggable adapter pattern** ensures testability (mock adapters) and separation of concerns between scheduling logic and external API specifics.
- **Polling interval** is configurable — shorter intervals are appropriate for live demos; longer intervals and batching are better for production to respect API rate limits.

Security & Operational Considerations

- **Credentials management:** store API keys and secrets in environment variables or a secret manager, never in source code.
- **Rate limits & backoff:** implement exponential backoff and logging for adapter-level API calls to handle throttling.
- **Idempotency and retries:** ensure adapters can handle retries safely or mark failures explicitly to avoid duplicate postings.
- **Audit logging:** store timestamps and external IDs returned by platforms for traceability and dispute resolution.