Solution Design Document (SDD)

# 1. Document Details

|  |  |
| --- | --- |
| Project Name | Activity Timeline Dashboard |
| Author | Nidhi |
| Date | 2025-06-17 |
| Version | 1.0 |
| Reviewers | TBD |
| Status | Draft |

# 2. Purpose

To design and implement a web application that:  
- Fetches data from two external APIs at the end of every day  
- Aggregates this data based on a common identifier  
- Stores the aggregated data in a database  
- Displays this data on the website  
- Maintains logs for each operation

# 3. Scope

Included:  
- Backend service to call APIs  
- Data aggregation logic  
- Database storage  
- Logging  
- Web UI to display the data  
- Admin panel for data manipulation

Not Included:  
- User authentication (unless specified later)

# 4. Assumptions

- APIs are reliable and accessible  
- Data returned is well-structured JSON  
- API rate limits and authentication (if any) are manageable  
- Website is publicly or internally hosted

# 5. High-Level Architecture

Scheduler → API Fetcher → External APIs → Data Aggregator → Logger + Database → Web Frontend

# 6. Data Flow

1. Daily Trigger (e.g., at 11:59 PM): Calls fetch\_data()  
2. Fetch activityTimeline and JIRA API JSON responses  
3. Parse and merge on id → create a combined DataFrame  
4. Save DataFrame as a table to the DB  
5. Log: Timestamp, API status, merge status, errors  
6. Frontend reads DB and renders data

# 7. Technology Stack

Frontend: React.js  
Backend: FastAPI (Python)  
Scheduler: APScheduler or Celery  
Database: PostgreSQL (or SQLite)  
ORM: SQLAlchemy  
Hosting: Render / Vercel / VPS  
Logs: logging (Python) + DB  
VCS: Git + GitHub

# 8. Database Schema (Sample)

TABLE aggregated\_data (  
 id TEXT PRIMARY KEY,  
 attr1\_from\_api1 TEXT,  
 attr2\_from\_api1 TEXT,  
 attrX\_from\_api2 TEXT,  
 fetch\_timestamp TIMESTAMP  
)

# 9. Logging Schema

TABLE logs (  
 id SERIAL PRIMARY KEY,  
 timestamp TIMESTAMP,  
 status TEXT,  
 message TEXT,  
 level TEXT  
)

# 10. Security & Error Handling

- API errors (timeouts, bad responses) logged with timestamp  
- Retry logic for failed API calls (up to 3 times)  
- Sanitization before DB insert  
- Display fallback message if DB is empty

# 11. JIRA Implementation Plan

Epic: API Data Aggregation Website

* Set up FastAPI backend and endpoints
* Implement scheduler to trigger API calls
* Write logic to fetch data from activityTimeline API
* Write logic to fetch data from JIRA API
* Merge data on `id` and convert to DataFrame
* Store data into PostgreSQL DB
* Implement logging system
* Build React frontend to display data
* Connect frontend to backend (REST API)
* Write test cases
* Deploy application
* Document everything in Confluence

# 12. Risks

- API outages  
- Inconsistent schema from external APIs  
- DB performance if dataset grows large  
- Timezone handling for "end of day"

# 13. Future Enhancements

- Add data visualization (charts, trends)  
- Add authentication  
- Export data (CSV/Excel)  
- API health monitoring dashboard