Comprehensive Guide: Engineering & Scaling par-delta-streamlit

Project Overview

- par-delta-streamlit is a data dashboard project built with Streamlit and Python, designed to visualize, analyze, and report business metrics.
- The project includes modules for data ingestion, processing, visualization, and user management.

Project Structure

- Key folders: streamlit_app/, dashboard/, data/, database/, scripts/.
- Each folder contains modules for specific responsibilities (UI, data, database, automation, uploads).
- Understanding the flow: Data is ingested, processed, stored, and visualized through Streamlit dashboards.

Key Technologies

- Python: Main language for backend and data processing.
- Streamlit: For interactive dashboards.
- pandas, numpy: Data analysis and manipulation.
- SQL/Supabase: Database management.
- Docker: Containerization for deployment.

Software Engineering Practices

- Project Organization: Modularize code into packages and submodules.
- Version Control: Use Git for tracking changes and collaboration.
- Testing: Add unit and integration tests using pytest.
- Documentation: Maintain README, docstrings, and usage guides.

Microservices Architecture

- Split monolithic code into independent services (e.g., data ingestion, reporting, user management).
- Use REST APIs (Flask/FastAPI) for communication between services.
- Deploy services separately using Docker containers.

Agility & Speed

- Implement CI/CD pipelines for automated testing and deployment.
- Modularize code for faster development and easier updates.
- Use feature branches and code reviews for rapid iteration.

Scalability

- Move heavy data processing to background jobs (Celery, RQ).
- Use cloud databases and storage (Supabase, AWS RDS).
- Containerize with Docker for easy scaling.

Resilience

- Implement error handling and retries in data pipelines.
- Use health checks and monitoring (Prometheus, Grafana).
- Design for graceful degradation and failover.

Debugging & Logging

- Add logging to all major modules using Python’s logging library.
- Log errors, warnings, and key events.
- Use centralized log management (ELK stack, cloud logging).
- Add debugging tools (Streamlit debug mode, Python debuggers).

Actionable Next Steps

- Explore the codebase and map out the data flow.
- Identify bottlenecks and areas for improvement.
- Experiment with refactoring and modularization.
- Add logging and basic tests to one module.
- Research microservices and build a small REST API.
- Set up Docker and containerize the app.