Table 1: Mavito Project: Architecture Mapping from Requirements to Implementation

| Requirement | Architectural Strategies Used | Architectural Pattern Used | Our Specific Implementation (Mavito Project) |
|---|---|---|---|
| **Scalability** | • Horizontal scale-out | Microservices | • **Backend:** Services are packaged in `Docker` containers and deployed to **Google Cloud Run** for request-based auto-scaling.<br>• **Frontend:** Hosted as a static site (e.g., on GitHub Pages or Firebase Hosting), which scales globally for content delivery. |
| **Performance** | • Async APIs<br>• Database Indexing | Microservices with Asynchronous APIs | • **Backend:** Built with **FastAPI** using an `async/await` model for non-blocking I/O, ensuring low-latency responses.<br>• **Database:** The backend queries an indexed **PostgreSQL** database, allowing for highly efficient filtering and sorting of millions of records. |
| **Availability** | • Replication | Leader-Follower Replication | • **Database: Google Cloud SQL** for PostgreSQL can be configured for High Availability (HA) to manage replication and failover automatically.<br>• **Services: Google Cloud Run** is a managed service that ensures backend services are reliable and restarted on failure. |

Table 1: Mavito Project: Architecture Mapping (continued)

| Requirement | Architectural Strategies Used | Architectural Pattern Used | Our Specific Implementation (Mavito Project) |
|---|---|---|---|
| **Usability & Data Integrity** | • Real-time UI<br>• Persistent State | Component-Based UI | • **Frontend:** A **React** and `TypeScript` application follows a component-based pattern to separate UI concerns.<br>• **Backend:** The backend uses **PostgreSQL** as the single source of truth for all terminology and user interaction data, ensuring data integrity. |
| **Security** | • TLS & tokens<br>• Secure Cloud Storage | API Gateway | • **Gateway Service: Google API Gateway** serves as the single entry point, handling TLS termination (enforcing HTTPS).<br>• **Authentication:** The backend's authentication service manages **JWT tokens** to secure protected API endpoints.<br>• **File Uploads:** User-submitted documents are securely uploaded directly to a private **Google Cloud Storage** bucket using temporary signed URLs. |

Table 1: Mavito Project: Architecture Mapping (continued)

| **Requirement** | **Architectural Strategies Used** | **Architectural Pattern Used** | **Our Specific Implementation (Mavito Project)** |
|---|---|---|---|
| **Offline Accessibility** & **Portability** | • Service workers and caching<br>• Background Sync | Progressive Web App (PWA) | • **Frontend:** Built as a PWA with a Service Worker to support offline access. API POST requests are queued using **Workbox Background Sync** and sent automatically upon reconnection.<br>• **Backend:** The entire backend is containerized with **Docker**, ensuring it is portable and can run consistently in any environment. |
| **Maintainability** & **Deployment** | • Modular design<br>• CI/CD Automation<br>• Database Migrations | Automated Testing & Deployment Pipeline | • **Code Quality: Husky pre-commit hooks** enforce `Ruff`, `Black`, and `Mypy` checks locally inside Docker containers.<br>• **CI/CD:** A **GitHub Actions** workflow automates testing and quality checks for both frontend and backend.<br>• **Database Schema:** The backend uses a dedicated container to run **Alembic** database migrations, ensuring a consistent schema across all environments. |