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	Micro-services	• **Backend:** Services are packaged in `Docker` containers and deployed to **Google Cloud Run** for request-based auto-scaling. • **Frontend:** Hosted on **GitHub Pages**, which scales globally for static content delivery.
Performance	• Async APIs	Micro-services with Asynchronous APIs	• **Backend:** Built with **FastAPI** using an `async/await` model for non-blocking I/O, ensuring low-latency responses. • **Frontend:** A modern **React/Vite** build process creates optimized, static assets for fast initial load times.
Availability	• Replication	Leader-Follower Replication	• **Database: Google Cloud SQL** for PostgreSQL can be configured for High Availability (HA) to manage replication and failover automatically. • **Services: Google Cloud Run** is a managed service that ensures 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 & **Latency**	• Real-time UI • Responsiveness • RAM-first	Model-View-Controller (MVC) / Component-Based UI	• **Frontend:** A **React** and `TypeScript` application follows a component-based pattern to separate UI concerns. • **Backend In-Memory Data:** The ‘search‘ and ‘analytics‘ services load data from JSON files into memory for fast lookups.
Security	• TLS & tokens	API Gateway	• **Gateway Service: Google API Gateway** serves as the single entry point, handling TLS termination (enforcing HTTPS). • **Authentication: JWT tokens** are managed by the ‘auth-service‘ to secure protected API endpoints. • **Secret Management:** Secure values are managed as **GitHub Encrypted Secrets** and injected into the CI/CD pipeline.
Offline Accessibility & **Portability**	• Service workers and caching	Progressive Web App (PWA)	• **Frontend:** The frontend is built as a PWA to support offline access to previously downloaded resources. • **Backend:** The backend is containerized with **Docker**, ensuring it is portable and can run in any environment.

Table 1: Mavito Project: Architecture Mapping (continued)

Requirement	**Architectural Strategies Used**	**Architectural Pattern Used**	**Our Specific Implementation (Mavito Project)**
Maintainability & **Deployment**	• Modular design • CI/CD Automation	Automated Testing & Deployment Pipeline	• **Code Quality: Husky pre-commit hooks** enforce `Ruff`, `Black`, and `Mypy` checks locally. • **CI/CD:** A **GitHub Actions** workflow automates testing, code quality checks, image building, and deployment for both frontend and backend.