Habit Tracker with Motivational Quotes

Service-Oriented Application

Mohammadmehdi Rajabpourshirazy

1 Introduction

The goal of this project is to design, implement and demonstrate a small yet complete service-oriented application. The chosen scenario is a *Habit Tracker with Motivational Quotes*. Users can register and authenticate, define daily habits, and mark them as completed. For each day, the application calculates completion statistics and combines them with a motivational quote retrieved from an external service. The final result is a command-line companion that supports users in building consistent study or wellness routines.

Following the course requirements, the system is decomposed into several REST-based services, each implementing a single logical responsibility. The application uses Node.js and Express as the main technology stack, and SQLite as an internal database for persistence. The services expose JSON APIs and are orchestrated by a process-centric gateway service. API documentation is provided using Swagger UI.

The project covers all required layers of the reference architecture: Data Services, Adapter Services, Business Logic Services and Process-Centric Services. In addition, a dedicated authentication service and a command-line interface are implemented. An external quote provider (ZenQuotes.io) is used to showcase integration with third-party APIs.

2 Architecture

Figure 1 illustrates the overall service-oriented architecture. The system is split into five main services, plus a command-line client and an external quote provider.

- **CLI User Interface** — a Node.js script that provides commands such as `register`, `login`, `add-habit`, `list-habits`, `complete-habit` and `daily-summary`. It communicates only with the Auth service and the Gateway service.
- **AuthService** — responsible for user registration, password hashing and token management. On successful login a random token is generated and stored together with the user identifier.
- **HabitDataService** (Data Services layer) — encapsulates all CRUD operations on habits and completion logs, storing them in the SQLite database.
- **HabitStatsService** (Business Logic layer) — queries the HabitDataService via HTTP, then computes daily completion statistics and streaks per habit.
- **QuoteAdapterService** (Adapter layer) — wraps the public ZenQuotes.io API and normalises the response to a simple internal structure containing only `text` and `author`.
- **Gateway Service** (Process-Centric layer) — orchestrates calls to all other services. It verifies access tokens, delegates data access to HabitDataService, requests statistics from HabitStatsService, queries QuoteAdapterService for a motivational quote and aggregates the results for the client.

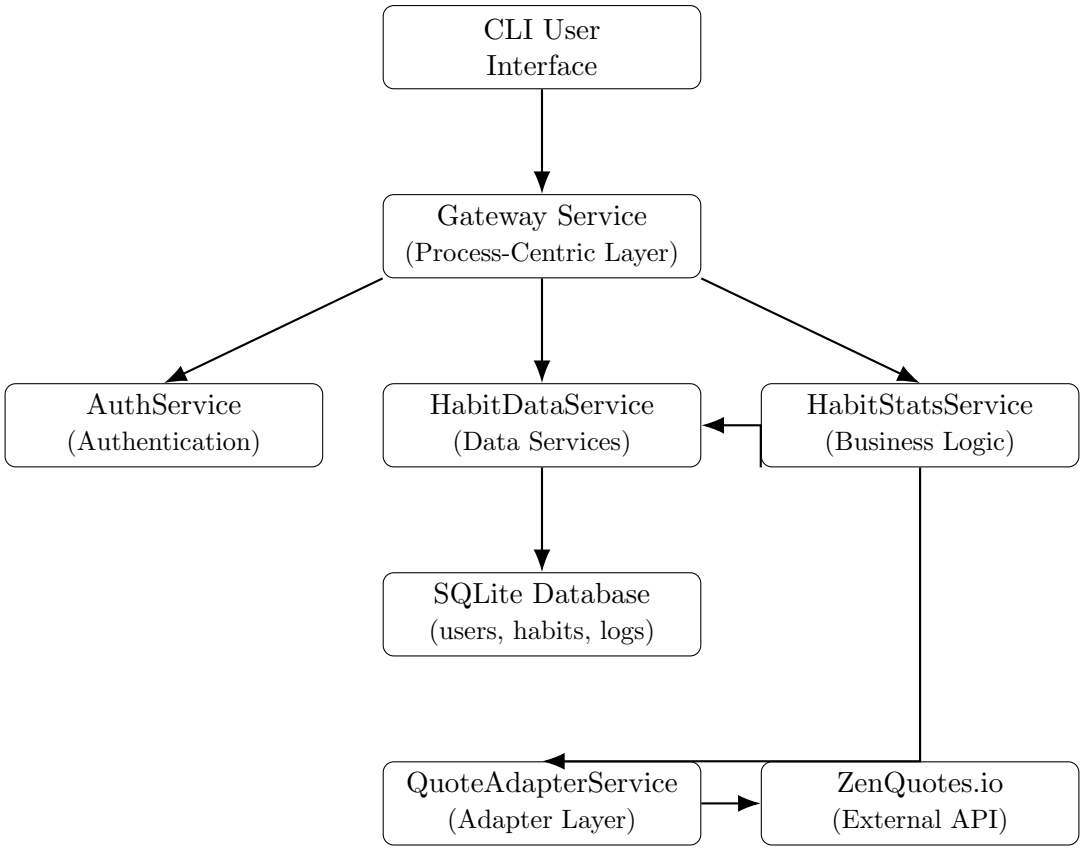

Figure 1: Service-oriented architecture of the Habit Tracker application. Arrows indicate the direction of service calls from caller to callee.

3 Implementation

All services are implemented in Node.js using the Express framework. A monorepo structure groups the five services and the CLI in separate folders while sharing a common `package.json` at the root. SQLite is used as the internal database and accessed through the `sqlite3` driver. Database tables store users, access tokens, habits and habit logs. Each service opens the same database file but accesses only the tables it logically owns.

Authentication is handled by `AuthService`. Passwords are hashed with SHA-256 before storage. On successful login, a random token is generated and stored together with the user identifier. The token is then used by the Gateway service via the HTTP `Authorization` header. Tokens are verified by calling the `/verify-token` endpoint of AuthService before handling any user-specific request.

`HabitDataService` encapsulates all CRUD logic for habits and their completion logs. In contrast, `HabitStatsService` does not access the database directly; instead it calls HabitDataService over HTTP to obtain habits and logs, then performs computations in memory. This separation respects the layering principle and allows the data service to evolve independently from the analytics service.

The `QuoteAdapterService` is a thin wrapper around the public ZenQuotes.io API. It normalises the external JSON format to a simple internal representation with two fields: `text` and `author`. This hides external details from the rest of the system and simplifies potential future changes, such as switching to a different quote provider.

The Gateway service exposes the main operations used by the CLI: adding habits, listing habits, completing a habit and obtaining the daily summary. For each request, it first verifies the token with AuthService and then orchestrates the calls to HabitDataService, HabitStatsService

and QuoteAdapterService. For example, the `/daily-summary` operation calls HabitStatsService to compute completion information and QuoteAdapterService to obtain a quote, merges the results and returns them to the client. This service therefore implements the process-centric layer presented in the lectures.

The command-line client is written in Node.js and communicates only with AuthService and the Gateway. It provides simple commands and stores the access token in a local file. This keeps the user interface minimal while satisfying the project requirement of at least a command-line UI.

4 Testing and Usage

The system has been tested both through the CLI and via Postman. Each service can be invoked independently, demonstrating the loose coupling encouraged by the service-oriented approach. Integration tests were performed by running all services concurrently and executing typical user flows: registration, login, creation of habits, marking habits as completed and retrieving summaries for various dates.

Swagger UI is exposed at the `/docs` endpoint of the Gateway service and loads a shared OpenAPI specification that documents all services. This enables automatic exploration of endpoints and also supports potential third-party consumers of the APIs.

5 Conclusions and Future Work

The Habit Tracker with Motivational Quotes demonstrates how even a small application can benefit from a service-oriented design. The decomposition into dedicated services improves modularity, testability and extensibility. The use of an external quote API illustrates how adapter services can hide third-party details and offer a stable interface to the rest of the system.

Possible future extensions include adding reminder notifications, introducing user-configurable goals and streak rewards, building a simple web or mobile front-end, and refining access control with role-based permissions. Nevertheless, the current prototype already satisfies the course requirements and serves as a compact example of applying service-oriented principles in practice.