# **Detailed Folder Structures for Individual Repositories of Microservices and BFF**

## **Introduction**

In a microservices architecture where each service is stored in its own repository, it’s essential to have a consistent and well-organized folder structure for each repository. This approach promotes service independence, clear ownership, and easier access control management. This document outlines detailed folder structures for each microservice and the Backend For Frontend (BFF) service for the Lusaka Integrated Solid Waste Management Platform.

Each service will have its own repository with a consistent structure to facilitate development, testing, deployment, and maintenance. Shared code and libraries will be managed separately to maintain service independence while ensuring code reuse.

## **General Folder Structure for Each Service Repository**

The following is a general template for the folder structure of each service:

service-name/  
├── README.md  
├── go.mod  
├── go.sum  
├── cmd/  
│ └── service-name/  
│ └── main.go  
├── internal/  
│ ├── config/  
│ │ └── config.go  
│ ├── server/  
│ │ ├── grpc\_server.go  
│ │ └── http\_server.go (if applicable)  
│ ├── handlers/  
│ │ └── handlers.go  
│ ├── services/  
│ │ └── business\_logic.go  
│ ├── repositories/  
│ │ └── database\_access.go  
│ ├── models/  
│ │ └── data\_models.go  
│ ├── middleware/  
│ │ └── middleware.go  
│ └── utils/  
│ └── helper\_functions.go  
├── pkg/  
│ └── (exported packages if any)  
├── api/  
│ └── proto/  
│ └── service-name.proto  
├── migrations/  
│ └── 0001\_initial\_schema.up.sql  
├── tests/  
│ ├── unit/  
│ │ └── unit\_test.go  
│ ├── integration/  
│ │ └── integration\_test.go  
│ └── mocks/  
│ └── mock\_service.go  
├── scripts/  
│ ├── build.sh  
│ ├── test.sh  
│ └── run.sh  
├── Dockerfile  
├── Makefile  
├── .gitignore  
├── .golangci.yml  
├── .editorconfig  
└── LICENSE

**Explanation:**

* **README.md:** Provides an overview, setup instructions, and usage information for the service.
* **go.mod, go.sum:** Go module files for dependency management.
* **cmd/:** Contains the main application entry point.
* **internal/:** Houses the core application code that should not be imported by other services.
  + **config/:** Configuration structures and loading logic.
  + **server/:** Server setup for gRPC and/or HTTP servers.
  + **handlers/:** Request handlers that interact with services.
  + **services/:** Business logic and service implementations.
  + **repositories/:** Data access layer for database interactions.
  + **models/:** Data models and structures.
  + **middleware/:** Middleware functions for request processing.
  + **utils/:** Utility functions specific to the service.
* **pkg/:** Exported packages that can be used by other services or repositories (if any).
* **api/proto/:** Contains Protocol Buffers definitions for gRPC communication.
* **migrations/:** Database migration scripts.
* **tests/:** Contains unit, integration tests, and mocks.
* **scripts/:** Automation scripts for building, testing, and running the service.
* **Dockerfile:** Instructions to build the Docker image for the service.
* **Makefile:** Build commands and shortcuts.
* **.gitignore:** Specifies files to be ignored by Git.
* **.golangci.yml:** Configuration for Go linters.
* **.editorconfig:** Ensures consistent coding styles.
* **LICENSE:** Licensing information.

## **Folder Structures for Specific Services**

### **1. Backend For Frontend (BFF) Service**

**Repository Name:** liswmc-bff-service

liswmc-bff-service/  
├── README.md  
├── go.mod  
├── go.sum  
├── cmd/  
│ └── bff/  
│ └── main.go  
├── internal/  
│ ├── config/  
│ │ └── config.go  
│ ├── server/  
│ │ └── http\_server.go  
│ ├── handlers/  
│ │ ├── user\_handler.go  
│ │ ├── payment\_handler.go  
│ │ ├── waste\_handler.go  
│ │ ├── fleet\_handler.go  
│ │ └── auth\_handler.go  
│ ├── services/  
│ │ ├── user\_service.go  
│ │ ├── payment\_service.go  
│ │ ├── waste\_service.go  
│ │ ├── fleet\_service.go  
│ │ └── auth\_service.go  
│ ├── middleware/  
│ │ ├── auth\_middleware.go  
│ │ ├── logging\_middleware.go  
│ │ └── recovery\_middleware.go  
│ └── utils/  
│ └── response\_util.go  
├── pkg/  
│ └── (exported packages if any)  
├── api/  
│ └── docs/  
│ └── openapi.yaml  
├── tests/  
│ ├── unit/  
│ │ └── bff\_unit\_test.go  
│ ├── integration/  
│ │ └── bff\_integration\_test.go  
│ └── mocks/  
│ └── mock\_services.go  
├── scripts/  
│ ├── build.sh  
│ ├── test.sh  
│ └── run.sh  
├── Dockerfile  
├── Makefile  
├── .gitignore  
├── .golangci.yml  
├── .editorconfig  
└── LICENSE

**Key Points:**

* **handlers/:** Handles HTTP requests and translates them into gRPC calls to backend services.
* **services/:** Contains client code for communicating with backend microservices via gRPC.
* **api/docs/:** OpenAPI/Swagger documentation for the RESTful APIs exposed by the BFF.
* **middleware/:** Middleware functions for authentication, logging, and error recovery.

### **2. Authentication and Authorization Service**

**Repository Name:** liswmc-auth-service

liswmc-auth-service/  
├── README.md  
├── go.mod  
├── go.sum  
├── cmd/  
│ └── auth-service/  
│ └── main.go  
├── internal/  
│ ├── config/  
│ │ └── config.go  
│ ├── server/  
│ │ └── grpc\_server.go  
│ ├── services/  
│ │ └── auth\_service.go  
│ ├── repositories/  
│ │ └── user\_repository.go  
│ ├── models/  
│ │ ├── user.go  
│ │ └── token.go  
│ ├── middleware/  
│ │ └── logging\_middleware.go  
│ └── utils/  
│ ├── crypto\_util.go  
│ └── token\_util.go  
├── api/  
│ └── proto/  
│ └── auth.proto  
├── migrations/  
│ └── 0001\_create\_users\_table.up.sql  
├── tests/  
│ ├── unit/  
│ │ └── auth\_unit\_test.go  
│ ├── integration/  
│ │ └── auth\_integration\_test.go  
│ └── mocks/  
│ └── mock\_user\_repository.go  
├── scripts/  
├── Dockerfile  
├── Makefile  
├── .gitignore  
├── .golangci.yml  
├── .editorconfig  
└── LICENSE

**Key Points:**

* **services/auth\_service.go:** Core authentication logic, including password hashing and token generation.
* **repositories/user\_repository.go:** Interacts with the database for user data.
* **api/proto/auth.proto:** Defines the gRPC service interface and messages.
* **utils/:** Contains utilities for cryptographic functions and token handling.

### **3. User Management Service**

**Repository Name:** liswmc-user-service

liswmc-user-service/  
├── README.md  
├── go.mod  
├── go.sum  
├── cmd/  
│ └── user-service/  
│ └── main.go  
├── internal/  
│ ├── config/  
│ ├── server/  
│ │ └── grpc\_server.go  
│ ├── services/  
│ │ └── user\_service.go  
│ ├── repositories/  
│ │ └── user\_repository.go  
│ ├── models/  
│ │ └── user.go  
│ ├── middleware/  
│ └── utils/  
├── api/  
│ └── proto/  
│ └── user.proto  
├── migrations/  
│ └── 0001\_create\_users\_table.up.sql  
├── tests/  
│ ├── unit/  
│ ├── integration/  
│ └── mocks/  
├── scripts/  
├── Dockerfile  
├── Makefile  
├── .gitignore  
└── LICENSE

**Key Points:**

* Manages user profiles, roles, and permissions.
* Exposes gRPC methods defined in user.proto.

### **4. Payment Service**

**Repository Name:** liswmc-payment-service

liswmc-payment-service/  
├── README.md  
├── go.mod  
├── go.sum  
├── cmd/  
│ └── payment-service/  
│ └── main.go  
├── internal/  
│ ├── config/  
│ ├── server/  
│ │ └── grpc\_server.go  
│ ├── services/  
│ │ ├── payment\_service.go  
│ │ └── reconciliation\_service.go  
│ ├── integrations/  
│ │ ├── mtn/  
│ │ │ └── mtn\_client.go  
│ │ ├── airtel/  
│ │ │ └── airtel\_client.go  
│ │ └── zamtel/  
│ └── zamtel\_client.go  
│ ├── repositories/  
│ │ └── payment\_repository.go  
│ ├── models/  
│ │ └── payment.go  
│ ├── middleware/  
│ └── utils/  
│ └── idempotency\_util.go  
├── api/  
│ └── proto/  
│ └── payment.proto  
├── migrations/  
│ └── 0001\_create\_payments\_table.up.sql  
├── tests/  
│ ├── unit/  
│ ├── integration/  
│ └── mocks/  
├── scripts/  
├── Dockerfile  
├── Makefile  
├── .gitignore  
└── LICENSE

**Key Points:**

* **integrations/:** Contains clients for interacting with mobile money providers.
* **services/payment\_service.go:** Implements idempotent payment operations and handles transaction states.
* **utils/idempotency\_util.go:** Provides functions to manage idempotency keys.

### **5. Waste Collection Service**

**Repository Name:** liswmc-waste-collection-service

liswmc-waste-collection-service/  
├── README.md  
├── go.mod  
├── go.sum  
├── cmd/  
│ └── waste-collection-service/  
│ └── main.go  
├── internal/  
│ ├── config/  
│ ├── server/  
│ ├── services/  
│ │ ├── schedule\_service.go  
│ │ ├── zone\_service.go  
│ │ └── report\_service.go  
│ ├── integrations/  
│ │ ├── google\_maps/  
│ │ │ └── google\_maps\_client.go  
│ │ └── google\_earth\_engine/  
│ └── google\_earth\_engine\_client.go  
│ ├── repositories/  
│ │ ├── zone\_repository.go  
│ │ ├── route\_repository.go  
│ │ └── report\_repository.go  
│ ├── models/  
│ │ ├── zone.go  
│ │ ├── route.go  
│ │ └── report.go  
│ ├── middleware/  
│ └── utils/  
├── api/  
│ └── proto/  
│ └── waste\_collection.proto  
├── migrations/  
│ └── 0001\_create\_zones\_table.up.sql  
├── tests/  
│ ├── unit/  
│ ├── integration/  
│ └── mocks/  
├── scripts/  
├── Dockerfile  
├── Makefile  
├── .gitignore  
└── LICENSE

**Key Points:**

* Manages waste collection schedules, zones, and issue reports.
* Integrates with Google Maps API and Google Earth Engine for route optimization and zone classification.

### **6. Fleet Management Service**

**Repository Name:** liswmc-fleet-management-service

liswmc-fleet-management-service/  
├── README.md  
├── go.mod  
├── go.sum  
├── cmd/  
│ └── fleet-management-service/  
│ └── main.go  
├── internal/  
│ ├── config/  
│ ├── server/  
│ ├── services/  
│ │ ├── vehicle\_service.go  
│ │ ├── driver\_service.go  
│ │ └── tracking\_service.go  
│ ├── repositories/  
│ │ ├── vehicle\_repository.go  
│ │ ├── driver\_repository.go  
│ │ └── tracking\_repository.go  
│ ├── models/  
│ │ ├── vehicle.go  
│ │ ├── driver.go  
│ │ └── tracking.go  
│ ├── middleware/  
│ └── utils/  
├── api/  
│ └── proto/  
│ └── fleet\_management.proto  
├── migrations/  
│ └── 0001\_create\_vehicles\_table.up.sql  
├── tests/  
│ ├── unit/  
│ ├── integration/  
│ └── mocks/  
├── scripts/  
├── Dockerfile  
├── Makefile  
├── .gitignore  
└── LICENSE

**Key Points:**

* Manages vehicles, drivers, and real-time tracking data.
* Provides APIs for assigning routes and dispatching vehicles.

### **7. Notification Service**

**Repository Name:** liswmc-notification-service

liswmc-notification-service/  
├── README.md  
├── go.mod  
├── go.sum  
├── cmd/  
│ └── notification-service/  
│ └── main.go  
├── internal/  
│ ├── config/  
│ ├── server/  
│ ├── services/  
│ │ ├── sms\_service.go  
│ │ ├── push\_service.go  
│ │ └── email\_service.go  
│ ├── integrations/  
│ │ ├── zamtel\_sms/  
│ │ │ └── zamtel\_sms\_client.go  
│ │ └── firebase\_messaging/  
│ └── firebase\_client.go  
│ ├── repositories/  
│ ├── models/  
│ ├── middleware/  
│ └── utils/  
├── api/  
│ └── proto/  
│ └── notification.proto  
├── tests/  
├── scripts/  
├── Dockerfile  
├── Makefile  
├── .gitignore  
└── LICENSE

**Key Points:**

* Manages sending notifications via SMS, push notifications, and email.
* Integrates with external messaging services.

### **8. Reporting and Analytics Service**

**Repository Name:** liswmc-reporting-service

liswmc-reporting-service/  
├── README.md  
├── go.mod  
├── go.sum  
├── cmd/  
│ └── reporting-service/  
│ └── main.go  
├── internal/  
│ ├── config/  
│ ├── server/  
│ ├── services/  
│ │ ├── report\_service.go  
│ │ └── analytics\_service.go  
│ ├── repositories/  
│ │ ├── report\_repository.go  
│ │ └── analytics\_repository.go  
│ ├── models/  
│ │ ├── report.go  
│ │ └── analytics.go  
│ ├── middleware/  
│ └── utils/  
├── api/  
│ └── proto/  
│ └── reporting.proto  
├── tests/  
├── scripts/  
├── Dockerfile  
├── Makefile  
├── .gitignore  
└── LICENSE

**Key Points:**

* Generates reports and provides analytics data to LISWMC staff.
* Aggregates data from other services for comprehensive insights.

## **Managing Shared Code and Libraries**

In a multi-repository setup, shared code and libraries need to be managed carefully to avoid duplication and maintain consistency.

### **Options for Shared Code Management**

1. **Shared Library Repositories:**
   * Create separate repositories for shared libraries or packages.
   * Example repositories:
     + liswmc-shared-utils
     + liswmc-proto-definitions
2. **Go Modules and Versioning:**
   * Publish shared libraries as Go modules.
   * Version the modules using semantic versioning.
   * Services can import the shared modules using go.mod.
3. **Package Repositories:**
   * Use private package repositories (e.g., Go Proxy, Artifactory, or GitHub Packages) to host shared modules.

### **Example Shared Repository: liswmc-proto-definitions**

liswmc-proto-definitions/  
├── README.md  
├── go.mod  
├── api/  
│ ├── auth/  
│ │ └── auth.proto  
│ ├── user/  
│ │ └── user.proto  
│ ├── payment/  
│ │ └── payment.proto  
│ ├── waste\_collection/  
│ │ └── waste\_collection.proto  
│ ├── fleet\_management/  
│ │ └── fleet\_management.proto  
│ ├── notification/  
│ │ └── notification.proto  
│ └── reporting/  
│ └── reporting.proto  
├── scripts/  
│ └── generate\_protos.sh  
├── Makefile  
├── .gitignore  
└── LICENSE

**Key Points:**

* Contains all Protocol Buffers definitions used across services.
* Services import the shared proto definitions from this repository.

### **Using the Shared Modules in Services**

* In each service’s go.mod, include the shared module:
* module github.com/liswmc/liswmc-payment-service  
    
  go 1.17  
    
  require (  
   github.com/liswmc/liswmc-proto-definitions v1.0.0  
   ...  
  )
* Import statements in the code:
* import (  
   "github.com/liswmc/liswmc-proto-definitions/api/payment"  
  )

## **Additional Considerations**

### **Consistent Coding Standards**

* Use linters and formatters (e.g., golangci-lint) configured via .golangci.yml in each repository.
* Enforce coding standards and best practices across all services.

### **CI/CD Pipelines**

* Set up continuous integration and deployment pipelines for each repository.
* Use shared scripts or templates where possible to reduce duplication.

### **Dependency Management**

* Keep dependencies updated.
* Use tools like Dependabot or Renovate to automate dependency updates.

### **Access Control**

* Manage access permissions per repository to control who can contribute to each service.

### **Documentation**

* Maintain up-to-date README.md files in each repository.
* Provide service-specific documentation, including API usage and setup instructions.

### **Issue Tracking and Project Management**

* Use a centralized issue tracker with labels or tags to associate issues with specific services.
* Alternatively, enable issue tracking per repository.

### **Versioning and Releases**

* Use semantic versioning for each service.
* Tag releases in Git and maintain release notes.

## **Conclusion**

By organizing each microservice and the BFF in separate repositories with a consistent folder structure, the Lusaka Integrated Solid Waste Management Platform can achieve service independence, clear ownership, and better access control. Shared code and libraries are managed separately, promoting code reuse while maintaining loose coupling between services.

This approach facilitates independent development, testing, deployment, and scaling of each service, aligning with microservices architecture best practices.

*For any further assistance or clarifications regarding the folder structures or repository management, please feel free to reach out.*