Motor Insurance Platform Solution Architecture Overview

Introduction

Executive Summary

This document outlines the proposed solution architecture for developing a motor insurance platform. The platform will provide a seamless and homogenous user experience for all user groups, powered by AngularJS on the front end, SnapLogic for middleware integration in a Microservices architecture, and an on-prem SQL Server for data persistence. Additionally, the

platform will integrate with external bank payment interfaces via SFTP to ensure secure and efficient financial transactions and incorporate Newgen Omnidocs as its document management system.

Scope

This document covers the system's technical architectural aspects, including its components, services, integrations, security, deployment, and testing considerations for Phase 1.

Objectives

- To establish a clear understanding of the Insurance platform’s architecture among stakeholders.
- To guide the development team in implementing the system according to the defined architecture.
- To ensure that the system meets all of its functional and non-functional requirements.

Audience

This document is for project stakeholders in SWAN & Neotech, including the leadership teams, architects, business SMEs, project managers, developers, and testers.

Architectural Overview

System Context

The Motor Insurance Management Platform is an online platform consisting of various modules serving all the operations and parties involved in the management of Motor Insurance Platform serving their customers, external entities such as surveyors, garages, etc. and internal colleagues managing claim handling and processing. The platform will be scalable to support external integrations such as broker's workflow/integrations, reporting, and more.

Phase 1 Key Features

- E-Claims 2.0(in development at SWAN)
- Surveyor Workflow Portal
- Garage Workflow Portal
- Spare Parts Management Portal
- SWAN Colleagues Insurance Handler Portal (Newgen)

Phase 2 Onwards Planned Features

- Car Rental Workflow Portal
    - Integrations with Car Rental Providers

- Litigation & Auctioneers workflows

Architectural Patterns and Styles

The SWAN Insurance platform adheres to the following design principles:

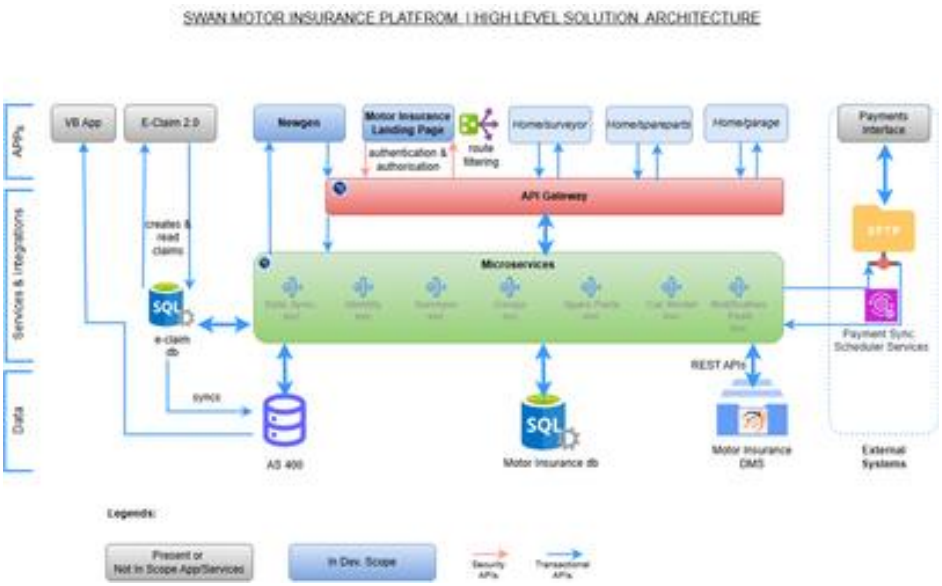
1. Hybrid Integration Platform
    1. Combines on-premises and cloud-based systems, enabling integration across different environments.
2. Modularity and Scalability
    - Leverage cloud-based services like SnapLogic to allow services to be easily added or modified and provide inherent scalability capabilities to handle growing data volumes, accommodating future business needs.
3. Service-Oriented Architecture (SOA)

    1. Similar to microservices but typically with extensive, more tightly coupled services, emphasizing reusability and interoperability.
4. API Gateway Pattern

    1. A single entry point for all clients (React.js client portal and Newgen colleague portal) to interact with the backend services.
5. Data Security and Governance
    - Established data lineage and audit trails to track updates and ensure data integrity.
6. Fault Tolerance and Reliability
    - Retry mechanisms to handle transient errors and ensure reliable data creation and updates.
    - Monitor and log system activities to facilitate troubleshooting and identify areas for improvement.

High-Level Solution Design

The image below shows the high-level architecture diagram for SWAN’s Motor Insurance Platform, its various applications/components/services, and their integration.



The platform features a modern, multi-layered, Microservices driven architecture with backend-for-each of the frontend that delivers seamless experiences for colleagues, customers, and third-party partners

Central to the platform is the Inetegrations/**API stack**, to be developed in SnapLogic. This layer facilitates integration between the front-end portals, external interfaces, data persistence and core systems. It handles data requests and transactions, ensuring secure and efficient communication across the system.

Data persists in SWAN-managed SQL servers, providing reliable and scalable storage for transactional data and user information.
Integration with legacy systems is achieved through orchestrations developed using SnapLogic. This system connects to the AS/400 system for policy administration and creates final insurance records in a DB2 database. This integration ensures continuity and data consistency across all platforms, enabling comprehensive reporting and analytics.

The architecture supports current business operations, including customer-facing and internal operations, and is scalable for future enhancements and integrations.

Technologies and Frameworks

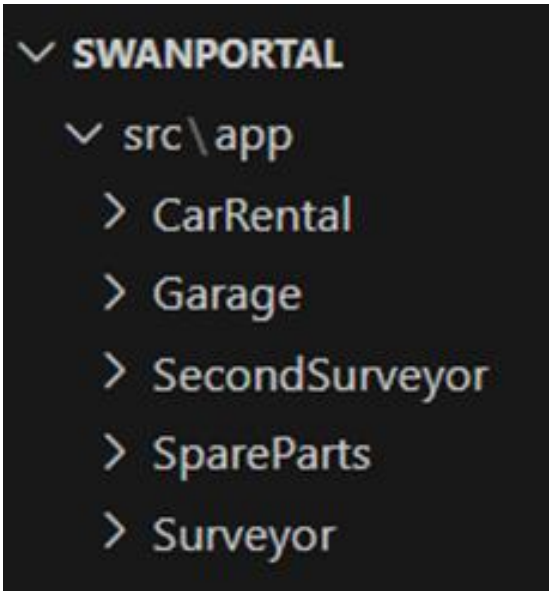
Front-End Applications

- **Client-Facing Portals:**
  - **Technology:** AngularJS, HTML5
    - The goal is to enable multiple group-specific UI flows (group-specific means only members of a given 'group' (general definition at this stage) can access the flows assigned to them. Examples of groups here would be

Surveyor, Garage, Car Rental, Spare Parts, etc. Simultaneous development of these flows would be enabled without impedance on their neighbouring groups whilst also aiming to keep channels open for feature sharing and reusability. In addition, push notifications from the server to the browser should be supported.

- **Recommendation & Important Technical Consideration:** Two frameworks, Angular and React, have been chosen as potential suitors due to their wide adoption in enterprise-grade front-end applications, meaning adequate support & maintenance are available for both, but each has its unique characteristics. For example, React calls itself a library for building user interfaces, but it ends its definition there. React was designed to be minimal and flexible, leaving many implementation details to the developer. On the other hand, Angular offers a cohesive, out-of-the-box solution that offers many functionalities that in a React application would have to be manually added and further configured.

Specifically concerning the project's requirement mentioned above, Angular is being recommended due to its built-in functionality for 'guarded' routes. These routes (flows) will be rendered conditionally based on a given authentication state. There is no configuration limit regarding the interaction between the authentication state and the guarding of the routes themselves. This is a typical pattern in Angular applications, and as a result, there is broad support for libraries/packages that enable such a pattern. This pattern is also achievable in ReactJS using PrivateRoutes. However, the point for recommending Angular here is that the framework will facilitate the configuration and setup more than React, which does not provide such functionality immediately. In Angular, one or multiple guards can be placed against a specific route, ensuring that only an assigned set of users can access it. In the above case, the guard logic will involve reading from the authentication state and ultimately return a boolean value (true or false). If all the guards return true, the respective route will only be activated; otherwise, a fallback route can be used to redirect the user.
Regarding the project structure, there are two apparent ways to structure such an application. The first (and recommended) option is to have one application encapsulating all required "groups" or "flows", as shown in image below:



This allows each flow to leverage the same login flow as an option. It also enables easy sharing of components/features and utilities since they are nearby. Angular also supports 'standalone' components, which are comparable to a standard React component. This allows the component to exist independently and be used within multiple flows.

The second option is to follow a **micro front-end architecture where each flow is separated into its own repo**. Although this would simplify routing in the base layer of each application, the benefits mentioned above regarding dependency/component sharing would be lost.

Angular also supports **lazy route loading**, which means the final build will be split into chunks, which are then loaded as required. Assuming that the routing logic is configured adequately against the auth state, this would guarantee that no user of a specific group will load code that is not meant for them. Although this is more of an optimization than anything else, it will prove essential when scaling up an application of this kind.

State management is also considered for UI state or API data. Angular provides two features for this purpose. One is Observables, which offers data transformation and combination operators via RxJS. Another is a more recent addition to the framework called Signals, a more straightforward way of handling states that multiple state users can interact with. Considering the base scale of the application, both should be sufficient. If a more significant scale state management solution is required (something like ngRx, which resembles Redux), it can be added later.

Angular has a Push API that leverages service workers to register for notifications. Once the user has permission, this will run as a separate background process. It then uses the **Notification API** to display native system notifications to the user. Note that the complete implementation of this would require a number of additional security measures and browser-specific considerations.

- **Features:** A singular multi-faced platform for various user groups:
    - Providing a responsive, intuitive, and user-friendly interface.
    - Implements client-side rendering and lazy loading for a fast and dynamic user experience.
    - Utilizes built-in Angular features like two-way data binding for efficient development.
    - Secures user interactions with authentication mechanisms with APIs using token-based authentication
    - **Target Users**: External users as individuals or from business entities, including garages, spare part suppliers and car rental companies.
- **Colleague Portal:**
    - **Technology:** Newgen & Core Java(services)
    - **Features:**
        - Comprehensive workflow management for internal staff to manage and handle claims processing and request workflow orchestrations.
        - Data synchronisation with other applications using nano-services to share workflow states and any document generated with the other systems in the platform
    - **Target Users**: Internal SWAN Colleagues
    - **Know Limitations:**
        - Push Notifications are not supported in the current version.
        - Bespoke service development is required for data synchronisation
        - Limited User Interface customisations are possible

Backend/APIs/Integrations

- **Technology:** SnapLogic APIM, SnapLogic APIs and .NET Framework*(TBD)*
- **API Gateway - SnapLogic APIM:** Acts as an integration hub to facilitate communication between the frontend apps, data sources, and external systems(Newgen, Banking SFTP server, etc.). The API gateway will manage, secure, govern and enables API discovery in one self-service solution.
- **Backend - Microservices:** The platform will be divided into smaller, independently deployable services, ensuring clear separation of concerns and fault isolation using SnapLogic pipelines. The following core business services are planned for implementation:
    - **Data Sync. Service:** This service synchronises the claims created from the E-Claim 2.0 portal, transforms them as per the agreed schema, and pushes them into Newgen for claim handling and workflow orchestrations. It also provides the Webhook for Newgen service/s to push data into the motor insurance platform.
    - **Identity Service:** This service caters to all the platform's security functions, such as token-based authentication, API call validation using tokens, session management/refresh, and role-based access management.
    - The implementation of this service will only be finalised once the authentication approach has been agreed upon with SWAN(token-based OAuth 2.0).

    - **Surveyor Service:** This service handles all the functions associated with surveyor processes, such as survey assessment, interactions with garages, and approval and payment processing workflows.
    - **Garage Service:** Handles all the functions associated with garage processes, such as creating vehicle EOR, interactions with spare parts procurement processes, and payments processing workflows.
    - **Spare Parts Service:** Handles all the functions associated with spare parts processes such as management & fulfilment of spare parts requests, interactions with garage process and payments processing workflows.
    - **Payment Service:** Facilitates and tracks transactions with external payment systems using SFTP.

Legacy System (AS/400)

Core legacy application for SWAN's insurance processes.

- **Technology**: IBM AS/400
- **Responsibilities**:
    - Maintain core data and legacy processes
    - Provide master data and services to the Motor Insurance Platform
    - Acts as a single-source-of-data repository for claims and members

Communication Integrations

- **Technology:** SMTP
- **Role:** As the notifications hub, facilitating email-based communications within the SWAN domain in the platform for any events per the configurations.

Data Persistence

- **Primary Data Storage:**
    - **Technology:** SQL Server
    - **Role:** Persistent storage of claimant information, claims details and transactional data as it's processed across multiple states in the claims workflow.
    - **Features:** Supports complex queries, indexing for performance optimization, and high availability configurations.
- **Legacy System Integration & Final Record Storage:**
    - **Technology:** AS/400 with IBMDB2 as the data store
    - **Role:** Stores finalized insurance records for compliance and executing associated operations such as payments.
    - **Integration:** Connected to the SnapLogic API layer to exchange data seamlessly with modern systems.
- **Document Management System**: Newgen's Omnidocs platform will be used for for document storage and retrieval, using its REST API capabilities to persist and access documents during the various stages motor insurance claims workflow.

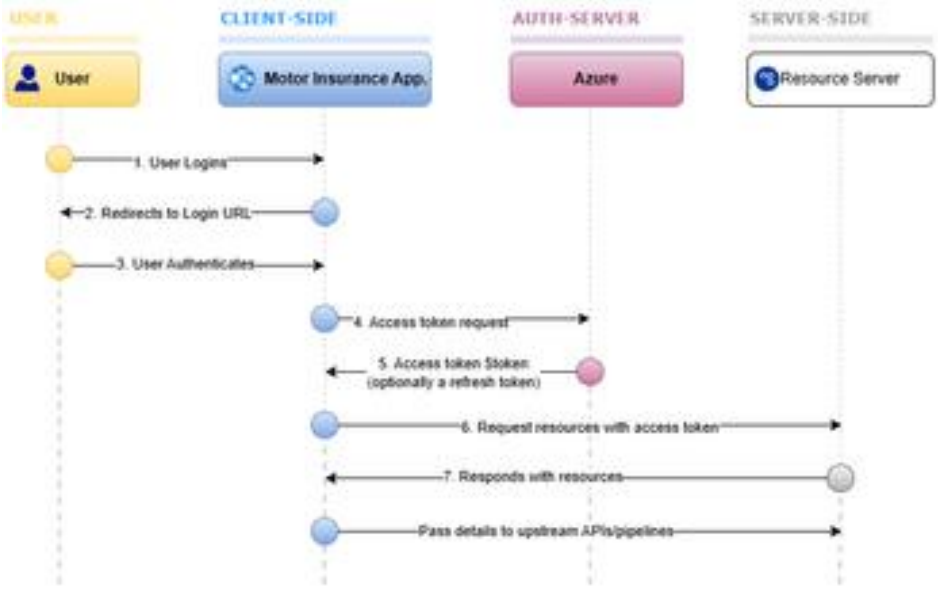
Data Flow and Integration

- **Real-Time Data Sync:** Data is synchronized in real-time between the front-end portals, API layer, and backend systems to ensure consistency and up-to-date information.
- **Error Handling:** error handling and logging mechanisms are implemented to ensure data accuracy and system reliability.

Non Functional Requirements

Security and RBAC

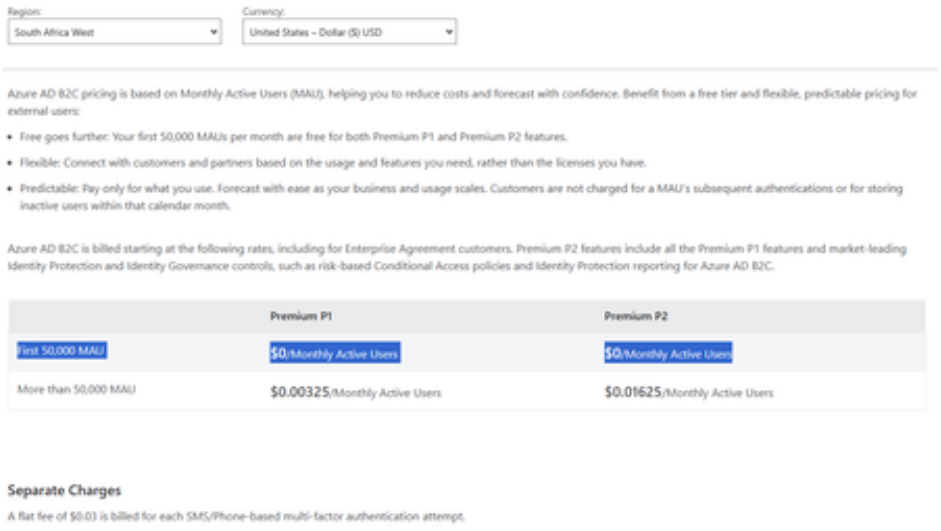
- **Motor Insurance Platform - Authentication and Authorisation:** This can be achieved by either of the following two approaches, described in the order of least time-to-implement
    1. **Bespoke Build**: Custom token-based authentication using a predefined username and password for all users on the Motor Insurance Portal and Workflow Management tool(Newgen). This approach requires custom RBAC defined and shared with the calling client application to decide the application routing. (a similar approach to what's implemented in the HR Portal.
    2. **OAuth Flow:** This approach will authenticate a client by delegating the authentication to an OAuth2 provider such as Azure, Okta, etc. The client is redirected to the OAuth provider to start the authentication flow. Once the flow is complete and the access token is obtained, the policy performs one or more requests to get information about the user, such as the ID and assigned role. Finally, a session cookie is returned to the client, and the client is redirected back to the requested URL. Subsequent requests authenticate based on the session cookie instead of repeating the OAuth flow. This implementation is described in the following authorisation code flow:



Proposed Authorization Code Grant Flow - Motor Insurance App

In terms of costs associated with using AD B2C, please note AD B2C pricing is based on Monthly Active Users (MAU); the first 50,000 MAUs are free, and only a flat fee of $0.03 is billed for each SMS/Phone-based multi-factor authentication attempt.

The image below shows the cost model described above with reference to West Africa as geography, but it is similar across the geos.



Region: South Africa West

Currency: United States – Dollar ($) USD

Azure AD B2C pricing is based on Monthly Active Users (MAU), helping you to reduce costs and forecast with confidence. Benefit from a free tier and flexible, predictable pricing for external users:

- Free goes further: Your first 50,000 MAUs per month are free for both Premium P1 and Premium P2 features.
- Flexible: Connect with customers and partners based on the usage and features you need, rather than the licenses you have.
- Predictable: Pay only for what you use. Forecast with ease as your business and usage scales. Customers are not charged for a MAU's subsequent authentications or for storing inactive users within that calendar month.

Azure AD B2C is billed starting at the following rates, including for Enterprise Agreement customers. Premium P2 features include all the Premium P1 features and market-leading Identity Protection and Identity Governance controls, such as risk-based Conditional Access policies and Identity Protection reporting for Azure AD B2C.

	Premium P1	Premium P2
First 50,000 MAU	$0/Monthly Active Users	$0/Monthly Active Users
More than 50,000 MAU	$0.00325/Monthly Active Users	$0.01625/Monthly Active Users

Separate Charges

A flat fee of $0.03 is billed for each SMS/Phone-based multi-factor authentication attempt.

- **Workflow Management(Newgen)**
- **API Security**: Rate limiting and input parameter validation at the SnapLogic API gateway level prevent malicious activities in the API platform.
- **Data Encryption**: All data will be encrypted at rest through database encryption and in transit using HTTPS(TLS)

Compliances & Regulations

- **Data Residenc**y: The primary data storage, using SQL Server, is located in SWAN's data centres within the Mauritius regions where the business operates. This ensures compliance with local data residency laws and reduces latency.
- **Data Encryption**: Data is encrypted in transit and at rest to protect sensitive information.

Scalability and Performance

- **Horizontal Scaling:** The platform can scale horizontally to handle increasing loads by adding more instances of the front-end portals and API services.

Environments & Deployment

The following environment strategy is recommended for seamless development, testing for testers & business and production deployment, subject to further discussions and approval with SWAN.

- **Development Environment(DEV)**: This is where developers write, test, and debug code, with access restricted to development teams.
- **Testing Environment(TEST)**: For QA(Neo & SWAN) teams to perform functional, integration, and regression testing. Its access is restricted to Dev. and QA teams only.
- **User Acceptance Testing Environment(UAT/Staging/Pre-Prod):** This environment is for SWAN business teams to perform final functional testing/review and conduct demos. It is the final resting place and testing opportunity before a build is deployed into production. The staging environment is set up to mimic the production setup (including SnapLogic pipelines, Newgen Omnidocs, and SFTP integration), including the test data created/replicated from production data. Access is restricted to QA teams and stakeholders performing UAT (User Acceptance Testing).
- **Production Environment: Live environment for end-users.** The setup is deployed on scalable on-prem servers and the SnapLogic cloud platform, which has fully configured load balancers, security measures, and monitoring tools. Access is restricted to authorized users and groups, except for key support team members for testing, maintenance, and monitoring.

All the environments are integrated with BitBucket for version control and follow the Continuos Integration and Deployment startegy on DEV & TEST environment, but for higher environments the deployment will be controlled by approval processes.

Quality Assurance

The testing strategy will include:

- Functional Tests: These are for individual solution components to be performed by developers and signed off after the QA team's review/test. They are usually performed in the development(a.k.aDEV) environment.
- Integration Tests: The QA team will conduct these tests to ensure all integrations work as expected across the environments. They are usually performed in the testing(a.k.a. TEST) environment.
- User Acceptance Testing (UAT): NHG's business to verify the solution meets business requirements, supported by the development team as required. They are usually performed in the testing(a.k.a. UAT) environment.