Technical Detailed Design Document

Application: Scotiabank Delegate Application

1. Introduction

The purpose of this document is to provide an in-depth technical design for the development and deployment of the Scotiabank Delegate Application. This application is designed to serve all internal employees of the bank, offering them a centralized platform to configure and manage Out of Office (OOO) settings across a range of enterprise systems, including Outlook, PPM, ServiceNow, Ariba, and others. It supersedes the existing OutOfOffice application that currently caters only to branch officers via the intralink domain. The new Delegate Application addresses limitations of the legacy system by supporting modern architectural standards, cloud-native deployment, cross-system integrations, audit trail logging, and enhanced role-based delegation capabilities.

This document is targeted at solution architects, developers, testers, DevOps engineers, and infrastructure teams involved in the implementation, testing, and maintenance of the Delegate Application. It outlines the functional and non-functional requirements, system architecture, integration details, deployment model, API contracts, data flow, and security architecture.

2. Objective

The objective of the Delegate Application is to deliver a next-generation internal-facing web platform that offers seamless management of Out of Office configurations across multiple bank systems. Key objectives include:

* Replace the legacy OutOfOffice application with a modern, scalable solution.
* Support all bank employees, including branch officers, with consistent capabilities.
* Allow managers to view and configure OOO settings for their subordinates.
* Integrate with existing systems like HR Directory, OPS Service, Identity Service, and WAM SSO.
* Ensure transactional integrity when updating OOO settings in multiple systems and databases.
* Provide backward compatibility by writing to the legacy Intralink DB.
* Maintain audit logs for all operations.
* Enhance system resiliency, observability, and security.

3. System Overview

The Delegate Application is a microservices-based web application comprising of a React frontend and a Spring Boot 3.4.1 backend. It is designed for deployment in Scotiabank’s Atlas Cloud Kubernetes environment. The application integrates with multiple third-party and internal systems through well-defined APIs and adheres to the latest enterprise architecture and security standards.

Key Features:

* Centralized interface for configuring OOO settings across supported applications
* Delegation and forwarding rule management
* Managerial override capabilities
* Automatic cleanup jobs post OOO period
* Real-time Kafka-based processing for specific business events (e.g., transit removal)
* Health check endpoints and monitoring integrations
* Dual-tab user interface: one tab for normal employee OOO management, one tab for designated admin role (branch admin/resource manager) to manage subordinates’ OOO settings.

4. Architecture Overview

[Insert High-Level Architecture Diagram Here]

The system follows a modular architecture with clearly defined components and responsibilities:

* Frontend Layer:
  + Built using React and Scotiabank’s Canvas UI framework
  + Implements a component-based model with reusable widgets for forms, tables, modals, etc.
  + Application state managed using Redux with middleware for async API calls (Redux Thunk)
  + Localization handled via i18n JSON bundles with dynamic language switching
  + Two-tab layout: Employee View and Admin View, conditionally rendered based on user role
  + Communication with backend via secure Axios REST client, using OAuth2 token passed in Authorization header
* Backend Layer:
  + Spring Boot 3.4.1 with modular service beans for separation of concerns
  + RESTful endpoints with Swagger for contract testing and documentation
  + Role-based security enforced using Spring Security with method-level annotations
* Integration Layer:
  + Adapter classes encapsulate logic to interface with external applications
  + AdapterFacade orchestrates adapter resolution, request translation, and error handling
* Persistence Layer:
  + PostgreSQL for Delegate-specific data (settings, users, audit)
  + Legacy Intralink DB for compatibility with OutOfOffice app
  + Connection pooling and schema versioning managed by Flyway
* Infrastructure Layer:
  + Deployed in Atlas Kubernetes clusters
  + All pods expose liveness and readiness probes
  + Secret mounting via Vault sidecar
  + Config maps used to control per-app feature toggles

5. Technology Stack

| Layer | Technology | Purpose |
| --- | --- | --- |
| Frontend | React + Canvas UI Framework | Rich UI interaction |
| Backend | Spring Boot 3.4.1 | Business logic and REST APIs |
| Security | OAuth2, WAM SSO | Secure authentication |
| DB | PostgreSQL, Intralink (legacy) | Persistent storage |
| Cloud | Atlas Kubernetes | Scalable infrastructure |
| Monitoring | Splunk, Dynatrace | Logs and metrics |
| Secrets | Vault | Secure credential storage |
| Messaging | Kafka | Event-based cleanup |

6. Functional Modules

6.1 Login and Initialization

This module handles the initial user authentication and profile enrichment process upon accessing the application.

Key Functions:

* Authenticate user via Scotiabank’s WAM SSO using OAuth2 protocol. The application is registered with WAM and utilizes the Authorization Code Grant flow. Upon successful login, the backend validates the JWT access token, extracting claims including employee ID, name, and roles.
* Tokens are validated via the Spring Security OAuth2 framework and cached in memory for session reuse.
* Fetch enriched user profile details from backend services:
  + HR Directory API (REST): Direct reports, display name, email, photo URL
  + Identity Service: officer ID and organizational mapping
  + OPS Service: transit and branch information
* Initialize a UserSessionContext object in backend memory, which is serialized and shared with the React frontend through the /getUserInfo API.
* Determine the role context (regular user or admin) based on WAM group claim or HR API role response.
* Session information is stored in Redux on the frontend for efficient tab switching and reactivity.
* Full login events are logged to Splunk, including login source IP, user-agent, and timestamp.

6.2 OOO Settings Management (Employee Tab)

This module enables regular bank employees to configure and manage their own out-of-office settings across all supported enterprise systems.

Key Functions:

* React frontend provides a dynamic form driven by the metadata retrieved from /getDelegateApplications, enabling form fields for each supported application.
* Form state is managed using Redux slices. Intermediate changes are autosaved locally before calling backend.
* API /saveOutOfOfficeData triggers a transactional operation in Spring Boot. It uses @Transactional annotation to group the following:
  + Write to delegate\_outofoffice table in PostgreSQL
  + Mirror entry in intralink\_oof legacy table
  + Call AdapterFacade to send config to remote application (e.g., Outlook)
  + If any step fails, a rollback is triggered and an error response returned to UI
* Factory Pattern is used to instantiate the correct adapter implementation (e.g., OutlookAdapter, ServiceNowAdapter) based on application code.
* AdapterFacade abstracts the coordination logic, allowing uniform access to adapter interfaces for save/cancel actions.
* User receives real-time validation feedback and transactional status via Redux async state flags and error modals.
* Custom messages are stored with language preference; UI supports multi-language fallback logic.
* Email forwarding rules are validated for correctness, including circular delegate detection and rule conflicts with existing entries.

6.3 Manager/Admin Role Functions (Admin Tab)

This module is accessible to users with elevated privileges (branch admins, resource managers) and provides capabilities to manage out-of-office settings on behalf of subordinate employees.

Key Functions:

* Admin privileges are determined using WAM role claims and verified via the HR API’s role hierarchy.
* Admin tab on the UI is conditionally rendered based on Redux state isAdminView. It loads additional components such as lookup, filtering panels, and history tables.
* Managers can search employees using /lookupEmployees or /lookupManagedOfficersOptions, which utilize fuzzy search and pagination for scalability.
* The React UI allows selection of multiple employees and batch configuration of OOO using saveOutOfOfficeData with bulkContext enabled.
* Security is enforced using method-level authorization (@PreAuthorize) on sensitive backend methods, limiting scope to managed transit or reporting hierarchy.
* All admin actions are logged with actedBy, targetUser, and actionScope fields in the audit log.
* The UI uses optimistic updates with rollback on API failure, providing immediate feedback and minimizing latency.
* Admins can view historical records using /lookupHistoryDelegateRecords, with timeline filtering and advanced search criteria.

6.4 Sync and Data Consistency

This module ensures data consistency between the Delegate Application and the underlying source systems.

Key Functions:

* Automatically invoked on user login or on-demand sync request
* Call each configured adapter to fetch current OOO settings from the remote system
* Compare with stored settings in Delegate DB and Intralink DB
* If differences are detected:
  + Update both local databases to reflect the true source-of-record (remote application)
  + Flag outdated entries for reconciliation
  + Display sync result to the user if manual sync was triggered
* Preserve data consistency across multiple systems and prevent manual override conflicts
* Detect and reconcile inconsistencies caused by direct updates in external applications

6.5 Cleanup Jobs

Automated background jobs are used to remove expired out-of-office rules and update data based on Kafka events.

Key Functions:

* Periodically scheduled cleanup job:
  + Identifies expired OOO settings based on end date
  + Connects to each remote application using the adapter
  + Removes auto-reply rules, forwarding settings, and delegated access
  + Deletes local records from Delegate DB and Intralink DB
  + Logs actions and outcome per employee per system
* Kafka consumer job:
  + Subscribed to a topic that emits officer removal from transit
  + On event receipt, parse payload and identify affected users
  + Remove their delegate access, cancel active OOO configurations
  + Save change history in the audit log
* Both jobs ensure reliability through retry logic, failure logging, and alerting
* Results visible in system monitoring dashboards and audit search interfaces

6.6 Audit Trail

This module captures and stores a detailed log of all critical user and system actions related to OOO management.

Key Functions:

* Automatically invoked during any create, update, or delete operation on out-of-office settings
* Captures fields including:
  + User ID, acting admin ID (if applicable)
  + Timestamp of action
  + Affected employee(s) and application(s)
  + Before and after values of OOO settings
  + Status (success/failure) and error details if any
* Stored in a dedicated audit table within the Delegate DB
* Accessible via API for audit lookups and filtering
* Supports search by:
  + Date range, employee ID, admin ID, action type, application
* Read-only access enforced through RBAC
* Enables compliance validation and root cause analysis of configuration issues

7. Non-Functional Requirements

* Performance: Response times under 1 second for 95% of requests
* Availability: 99.95% uptime with failover support
* Scalability: Horizontal scaling with Kubernetes
* Security: OAuth2 for access; Vault for secrets; data encryption at rest
* Monitoring: Logs to Splunk; performance tracked by Dynatrace
* Compliance: Audit logs; role-based access control (RBAC)

8. API Design

[Insert API Endpoint Diagram Here]

| API Endpoint | Description |
| --- | --- |
| /getUserInfo | Returns login user profile and subordinate data |
| /getOutofOfficeData | Lists current OOO entries per application |
| /getSupportedLanguages | Lists UI-supported languages |
| /getDelegateApplications | Lists applications available for delegation |
| /lookupEmployees | Search employees by name keyword |
| /lookupManagedOfficersOptions | Lists officers by manager’s transit or HR hierarchy |
| /lookupHistoryDelegateRecords | Search audit logs for historical OOO |
| /saveOutOfOfficeData | Transactional save to systems and DBs |
| /cancelOutOfOfficeData | Deletes OOO from systems and DBs with rollback and audit |
| /getUserorOfficerInfo | Fetches basic user data for a selected officer |

9. Backend Integration

The backend is responsible for orchestrating API calls, data persistence, and external system interactions. All services are developed using Spring Boot and follow RESTful design.

Integration Highlights:

* WAM SSO (OAuth2):
  + The Delegate App is registered as a confidential client with WAM.
  + Uses OAuth2 Authorization Code Grant for user login.
  + Access tokens are validated using public keys provided by WAM’s JWKS endpoint.
  + Claims are extracted and used to initialize session, assign role, and sign requests to downstream services.
* HR Directory API:
  + REST API returning JSON data for employees, reports, department hierarchy
  + Secured via OAuth2 token-based access, with roles scoped for read-only lookup
* OPS Service and Identity Service:
  + Provide officer ID, branch, and transit info for both user and subordinates
  + Used to construct org chart views and access boundaries for admin role
* Kafka:
  + Listener configured using Spring Kafka
  + Subscribed to bank.transit.cleanup topic for downstream officer-removal events
* Adapter Services:
  + Outlook, PPM, Ariba, ServiceNow integrations use application-specific REST/SOAP protocols
  + Each adapter defines headers, authentication strategy, request/response mapping, and retry logic

10. Design Patterns. Design Patterns

The Delegate Application incorporates several well-established design patterns to promote modularity, extensibility, and maintainability.

* Facade Pattern:
  + The AdapterFacade acts as a single point of interaction for all OOO operations. It simplifies access to multiple application-specific adapters (Outlook, Ariba, etc.), orchestrates transactions, and centralizes error handling.
  + Example: AdapterFacade.saveOutOfOffice(EmployeeOOOConfig config) delegates to OutlookAdapter.save(config) or ServiceNowAdapter.save(config) transparently.
* Factory Pattern:
  + The application uses a AdapterFactory to dynamically instantiate the correct adapter class based on the application identifier in the config.
  + Supports new system integrations by simply registering a new implementation class.
  + Uses Spring’s @Component scan and a Map<String, OutOfOfficeAdapter> autowire injection for decoupled resolution.
* Adapter Pattern:
  + Each supported external system has its own adapter implementation that encapsulates the API logic required to push/pull OOO configurations.
  + Adapters implement a common interface with save(), cancel(), and fetchCurrentSettings() methods.
* Strategy Pattern:
  + Used within UI rendering logic to choose behavior dynamically (e.g., calendar vs dropdown delegate picker).
* Transactional Pattern:
  + Critical service methods are annotated with @Transactional(rollbackFor = Exception.class).
  + Remote API calls are wrapped in try/catch blocks and flagged for rollback using Spring’s TransactionAspectSupport if failure occurs.
  + Ensures full ACID compliance when interacting with multiple databases and external systems.

11. Deployment Design

[Insert Deployment Architecture Diagram Here]

* Built using CI/CD pipeline with code review gates
* Artifacts pushed to secure container registry
* Deployed via Helm charts into Atlas Kubernetes clusters
* Pods include:
  + Spring Boot service pod
  + PostgreSQL pod
  + Sidecar for Vault token injection
* Horizontal Pod Autoscaler (HPA) enabled
* DR-ready by replicating deployment in secondary cluster

12. Logging and Monitoring

* Logs output to stdout/stderr, aggregated by Fluentd to Splunk
* Dynatrace agent embedded for APM tracking
* Monitored metrics:
  + API response time
  + Error rates
  + Container health
* Alerts configured for high latency, frequent failures, and pod crashes

13. Testing Strategy

[Insert Testing Strategy Diagram Here]

| Test Type | Tools | Coverage |
| --- | --- | --- |
| Unit Testing | JUnit | 80%+ |
| Integration | Mockito, TestContainers | Endpoint-level validation |
| Functional UI | Cypress, Selenium | Automated flows and regression |
| Load Testing | JMeter | 500 concurrent users |
| Resilience | ChaosMonkey | Restart pod, fail downstream |
| Security Scanning | SonarQube, OWASP ZAP | Regular static/dynamic scans |

14. Environments

| Environment | Purpose |
| --- | --- |
| IST | Internal system testing |
| UAT | User validation and feedback |
| NFT | Load and performance testing |
| PROD | Production deployment with HA setup |

15. Risks & Mitigation

| Risk | Description | Mitigation |
| --- | --- | --- |
| Remote app downtime | OOO save may fail if Outlook/ServiceNow is unreachable | Retry policy and rollback transaction |
| Data inconsistency | Differences between actual OOO state and DB | Sync logic during login refreshes DB |
| Legacy DB dependency | Dual writes needed for backward compatibility | Monitor usage and plan migration strategy |

16. Future Enhancements

* AI-driven delegate recommendation based on history and hierarchy
* Integration with calendar APIs for smart conflict detection
* User-configurable OOO start/end reminders
* Multilingual support across all UI elements
* Service mesh integration for observability and zero-trust access control

17. Appendix

[Insert Data Model Diagrams Here]  
[Insert Sequence Diagrams Here]  
[Insert Adapter Class Design Here]  
[Insert Vault and OAuth2 Interaction Flow Here]