# Business Requirements for Delegate App

**1. Objective**

The Delegate App will allow bank employees to set and manage Out of Office (OOF) statuses across different platforms like Outlook 365, ServiceNow, and PPM. The app will provide a single interface for handling OOF settings, supporting both employee self-service and administrative override.

**2. Actors**

**Bank Employee**

* A user who logs into the Delegate App to manage their own OOF settings.
* Can edit personal profile and set out of office preferences.

**Administrator**

* A user with elevated privileges.
* Can view and edit other employees’ out of office settings.
* Can manage application-wide settings.

**3. Functional Requirements**

**3.1 User Login and Authentication**

* The app will support authentication using Scotia ID and password through WAM (Web Access Manager).
* If not logged in, the user will be redirected to the WAM login page.
* After successful login:
  + If the user is a bank employee → Redirect to the delegate homepage.
  + If the user is an administrator → Redirect to the admin panel.

**3.2 Delegate Homepage**

**Profile Management**

* User can access a "My Profile" section through the top-right corner.
* The following information can be viewed and updated:
  + Preferred language
  + Name and email (validated)
  + Transit number
  + Profile picture

**Out of Office Management**

1. **View Existing Out of Office Record**  
   If an existing OOF record exists, show:
   * Start date
   * End date
   * OOF message (if provided)
   * Delegate email address
2. **Update Existing Out of Office Record**  
   User can update the following fields:
   * Start date
   * End date
   * OOF message (optional)
   * Delegate email address
3. **Create New Out of Office Record**  
   User can create a new OOF record by providing:
   * Start date
   * End date
   * OOF message (optional)
   * Delegate email address
4. **View Past Out of Office History**  
   User can click a link to view historical OOF records.  
   Show the following fields:
   * Start date
   * End date
   * OOF message
   * Delegate email address

**Outlook 365 Integration**

* When a new OOF record is created or updated:
  + The app will make a service call to Outlook 365 to configure auto-reply and email forwarding.
* At the end of an OOF period:
  + A scheduled batch job will clean up the OOF forwarding.

**3.3 Administrator Panel**

1. **Manage Out of Office for Other Employees**
   * Administrator can search for any employee.
   * Administrator can edit or create OOF settings on behalf of an employee.

**4. Non-Functional Requirements**

**Scalability**

* The system should support thousands of concurrent users without performance degradation.

**Performance**

* The system should respond to user actions within 500ms for most requests.

**Security**

* Authentication and authorization will be handled via OAuth2 using WAM.
* Personal data (e.g., email, transit number) will be encrypted at rest.

**Logging and Monitoring**

* Application logs will be captured using **slf4j** and exported to **Splunk** for analysis.
* System should monitor failed service calls and retry automatically if necessary.

**Reliability**

* The system should handle temporary service outages gracefully.
* Batch jobs should retry failed OOF cleanup attempts automatically.

**5. Design Patterns**

**Facade Pattern**

* The Delegate App will use the Facade Pattern to provide a single interface for handling OOF operations across different platforms (Outlook, ServiceNow, PPM).
* The facade will:
  + Route the request to the correct handler based on the selected platform.
  + Abstract the complexity of dealing with different API structures.

**Factory Pattern**

* A Factory Pattern will be used to create handlers dynamically for different OOF platforms.
* Example:
  + OOFHandlerFactory.createHandler("Outlook") → Creates an OutlookHandler
  + OOFHandlerFactory.createHandler("ServiceNow") → Creates a ServiceNowHandler
  + OOFHandlerFactory.createHandler("PPM") → Creates a PPMHandler

**Adapter Pattern**

* An Adapter Pattern will map the OOF request to the platform-specific API:
  + OutlookAdapter.setOutOfOffice() → Calls Outlook Graph API
  + ServiceNowAdapter.setOutOfOffice() → Calls ServiceNow REST API
  + PPMAdapter.setOutOfOffice() → Calls PPM REST API

**6. Data Model**

**Employee Table**

| **Field** | **Type** | **Description** |
| --- | --- | --- |
| id | INT | Primary key |
| name | VARCHAR | Employee name |
| email | VARCHAR | Employee email |
| transit | VARCHAR | Transit number |
| language\_preference | VARCHAR | Preferred language |

**Out of Office Table**

| **Field** | **Type** | **Description** |
| --- | --- | --- |
| id | INT | Primary key |
| employee\_id | INT | Foreign key to Employee table |
| start\_date | DATETIME | OOF start date |
| end\_date | DATETIME | OOF end date |
| message | TEXT | OOF message |
| delegate\_email | VARCHAR | Email of delegate |

**7. API Interface**

**Set Out of Office**

**POST** /api/oof/set  
**Request Body**:

{

"employee\_id": 123,

"start\_date": "2025-04-01",

"end\_date": "2025-04-10",

"message": "On vacation",

"delegate\_email": "delegate@example.com"

}

**Response**:

{

"status": "success",

"message": "OOF set successfully"

}

**Finish Up Out of Office (Batch Job)**

**POST** /api/oof/cleanup  
**Description:**

* A batch job will execute at the end of the out-of-office period to clean up any temporary settings (e.g., email forwarding).
* This ensures that forwarding and auto-reply settings are reset after the OOF period ends.

**Request Body**:

{

"employee\_id": 123,

“oofrecordId”: 111

}

**Response**:

{

"status": "success",

"message": "Out of Office settings cleaned up successfully"

}

**8. Error Handling**

| **Error Code** | **Description** | **Action** |
| --- | --- | --- |
| 401 | Unauthorized | Redirect to login |
| 500 | Internal Server Error | Log and retry |
| 404 | Not Found | Show message to user |
| 409 | Conflict | Inform user of overlap |

**9. Future Enhancements**

* Add support for more platforms (e.g., Slack, Teams).
* Add email notifications when an OOF setting is changed.
* Allow scheduling of OOF in advance.

**Summary**

The Delegate App will provide a unified experience for setting and managing out-of-office statuses across multiple platforms. It will use a scalable and flexible design, allowing future extensions without major code changes. The use of the **Facade**, **Factory**, and **Adapter** patterns will ensure clean separation of concerns and flexibility in adding support for new platforms.

# System Detailed Design for Delegate App

**1. Objective**

The Delegate App will allow bank employees to set and manage Out of Office (OOF) statuses across multiple platforms, including **Outlook 365**, **ServiceNow**, and **PPM**. It will provide a unified interface to handle OOF settings and support both employee self-service and administrative overrides.

**2. System Overview**

**2.1 System Components**

The Delegate App will consist of the following major components:

* **UI Layer** – React-based frontend to handle user interaction.
* **Service Layer** – Spring Boot-based backend to handle business logic.
* **Integration Layer** – Adapter-based integration with external platforms (Outlook, ServiceNow, PPM).
* **Persistence Layer** – PostgreSQL to store employee and OOF records.
* **Authentication Layer** – WAM-based OAuth2 authentication.

**2.2 Technology Stack**

| **Layer** | **Technology** |
| --- | --- |
| Frontend | React, Redux |
| Backend | Spring Boot, Java 17 |
| Database | PostgreSQL |
| Authentication | OAuth2 via WAM |
| Integration | RESTful APIs (Outlook Graph, ServiceNow, PPM) |
| Logging | slf4j, Splunk |
| Deployment | Kubernetes, Docker |
| CI/CD | Jenkins |

**3. Functional Design**

**3.1 Login and Authentication**

**Flow:**

1. User accesses the Delegate App.
2. If not logged in:
   * Redirect to WAM (Web Access Manager) for OAuth2 login.
3. After successful authentication:
   * Check user role.
   * If user is a bank employee → Redirect to homepage.
   * If user is an administrator → Redirect to admin panel.

**3.2 Delegate Homepage**

**3.2.1 Profile Management**

* User can view and update:
  + Name
  + Email
  + Preferred language
  + Transit number
  + Profile picture

**3.2.2 Out of Office Management**

1. **View Existing Out of Office Record**
   * Fetch current OOF record from the database.
   * Display:
     + Start date
     + End date
     + Message
     + Delegate email address
2. **Update Existing Out of Office Record**
   * Allow user to modify existing record.
   * Fields:
     + Start date
     + End date
     + Message
     + Delegate email address
   * Call the **OOF Handler** to update record in the integrated platform.
3. **Create New Out of Office Record**
   * Allow user to create a new OOF record.
   * Fields:
     + Start date
     + End date
     + Message
     + Delegate email address
   * Call the **OOF Handler** to create the OOF record in the integrated platform.
4. **View Past Out of Office History**
   * Display historical OOF records from the database:
     + Start date
     + End date
     + Message
     + Delegate email address

**3.2.3 Outlook 365 Integration**

* When a new OOF record is created or updated:
  + Call the **Outlook Adapter**.
  + Configure auto-reply and forwarding settings in Outlook.
* When OOF period ends:
  + A batch job will remove auto-reply and forwarding settings.

**3.3 Administrator Panel**

1. **Manage Out of Office for Other Employees**
   * Search employee by name or ID.
   * Edit existing OOF settings on behalf of employees.
2. **Manage Application Settings**
   * Update system-wide configurations:
     + Supported platforms
     + Default language
     + Notification settings

**4. Non-Functional Design**

**4.1 Scalability**

* Use Kubernetes to auto-scale based on traffic.
* Load balancing across microservices.

**4.2 Performance**

* Ensure REST API response time < 500ms under normal load.
* Use caching for frequently accessed data.

**4.3 Security**

* Encrypt sensitive data at rest using AES-256.
* OAuth2-based token authentication using WAM.

**4.4 Logging and Monitoring**

* Use **slf4j** for application-level logging.
* Send logs to **Splunk** for analysis.

**4.5 Reliability**

* Graceful failure handling and retry for failed external calls.
* Monitor health and restart pods on failure.

**5. Design Patterns**

**5.1 Facade Pattern**

* Provide a unified interface for handling OOF operations.
* Facade calls the appropriate handler based on platform:
  + OutlookHandler
  + ServiceNowHandler
  + PPMHandler

**5.2 Factory Pattern**

* Factory dynamically creates handlers based on platform type.
* Example:

public class OOFHandlerFactory {

public static OOFHandler createHandler(String platform) {

switch (platform.toLowerCase()) {

case "outlook":

return new OutlookHandler(new OutlookAdapter());

case "servicenow":

return new ServiceNowHandler(new ServiceNowAdapter());

case "ppm":

return new PPMHandler(new PPMAdapter());

default:

throw new IllegalArgumentException("Unsupported platform");

}

}

}

**5.3 Adapter Pattern**

* Map platform-specific APIs to a common interface.
* Example:

public class OutlookAdapter {

public void setOutOfOffice(String fromDate, String toDate, String message) {

// Call Outlook Graph API

}

}

**6. Data Model**

**6.1 Employee Table**

| **Field** | **Type** | **Description** |
| --- | --- | --- |
| id | INT | Primary key |
| name | VARCHAR | Employee name |
| email | VARCHAR | Employee email |
| transit | VARCHAR | Transit number |
| language\_preference | VARCHAR | Preferred language |

**6.2 Out of Office Table**

| **Field** | **Type** | **Description** |
| --- | --- | --- |
| id | INT | Primary key |
| employee\_id | INT | Foreign key to Employee table |
| start\_date | DATETIME | OOF start date |
| end\_date | DATETIME | OOF end date |
| message | TEXT | OOF message |
| delegate\_email | VARCHAR | Email of delegate |

**7. API Design**

**7.1 Set Out of Office**

**POST** /api/oof/set  
**Request Body:**

{

"employee\_id": 123,

"start\_date": "2025-04-01",

"end\_date": "2025-04-10",

"message": "On vacation",

"delegate\_email": "delegate@example.com"

}

**Response:**

{

"status": "success",

"message": "OOF set successfully"

}

**7.2 Finish Up Out of Office (Batch Job)**

**POST** /api/oof/cleanup  
**Request Body:**

{

"employee\_id": 123,

"oof\_record\_id": 111

}

**Response:**

{

"status": "success",

"message": "Out of Office settings cleaned up successfully"

}

**8. Error Handling**

| **Error Code** | **Description** | **Action** |
| --- | --- | --- |
| 401 | Unauthorized | Redirect to login |
| 500 | Internal Server Error | Log and retry |
| 404 | Not Found | Show message to user |
| 409 | Conflict | Inform user of overlap |

**9. Future Enhancements**

* Add support for Slack and Teams.
* Add notifications when OOF settings change.
* Allow future scheduling of OOF in advance.

**10. Deployment Strategy**

* **Blue-Green Deployment** – Minimize downtime during releases.
* **Kubernetes-based Deployment** – Auto-scaling and high availability.
* **CI/CD Pipeline** – Jenkins pipeline for build and deployment.

**Summary**

The Delegate App will provide a unified experience for setting and managing Out of Office across platforms. The combination of **Facade**, **Factory**, and **Adapter** patterns will ensure clean separation of concerns and future scalability. The system will handle high traffic, support security standards, and allow easy extension for new platforms.

# System Design Summary:

**System Design Summary for Delegate App**

The **Delegate App** allows bank employees to manage Out of Office (OOF) statuses across multiple platforms, including **Outlook 365**, **ServiceNow**, and **PPM**, through a single interface. The app supports both employee self-service and administrative override, providing a consistent and efficient user experience.

**System Overview**

The Delegate App consists of five major components:

1. **UI Layer** – A React-based frontend for user interaction.
2. **Service Layer** – A Spring Boot backend to handle business logic.
3. **Integration Layer** – Adapter-based layer to handle platform-specific APIs.
4. **Persistence Layer** – PostgreSQL database to store employee and OOF data.
5. **Authentication Layer** – OAuth2-based authentication using WAM (Web Access Manager).

**Technology Stack**

* **Frontend**: React, Redux
* **Backend**: Spring Boot, Java 17
* **Database**: PostgreSQL
* **Integration**: RESTful APIs (Outlook, ServiceNow, PPM)
* **Logging**: slf4j, Splunk
* **Deployment**: Kubernetes, Docker
* **CI/CD**: Jenkins

**Functional Design**

1. **Login and Authentication**:
   * Redirect to WAM for login if the user is not authenticated.
   * Role-based redirection to employee homepage or admin panel after login.
2. **Delegate Homepage**:
   * **Profile Management** – Users can update their name, email, preferred language, transit number, and profile picture.
   * **Out of Office Management** –
     + View, update, and create OOF records.
     + OOF settings include start date, end date, message, and delegate email.
     + Historical OOF records can be viewed.
   * **Outlook 365 Integration** –
     + When a new OOF record is created, the app configures auto-reply and email forwarding through the Outlook Adapter.
     + A batch job will clean up forwarding settings after the OOF period ends.
3. **Administrator Panel**:
   * Administrators can search and update employee OOF settings.
   * System-wide settings such as supported platforms and default language can be configured.

**Non-Functional Design**

* **Scalability** – Kubernetes for auto-scaling and load balancing.
* **Performance** – REST API response time under 500ms.
* **Security** – Data encrypted at rest using AES-256; OAuth2-based authentication.
* **Logging** – slf4j and Splunk for monitoring.
* **Reliability** – Graceful failure handling and auto-retry for failed service calls.

**Design Patterns**

1. **Facade Pattern** – A unified interface for managing OOF settings across different platforms.
2. **Factory Pattern** – Dynamically creates handlers for Outlook, ServiceNow, and PPM based on platform type.
3. **Adapter Pattern** – Maps platform-specific APIs to a common interface for consistency.

**Data Model**

* **Employee Table** – Stores employee details (name, email, transit, preferred language).
* **Out of Office Table** – Stores OOF records (start date, end date, message, delegate email).

**API Design**

* **Set Out of Office** – Endpoint to create or update OOF settings.
* **Finish Up Out of Office** – Batch job to clean up OOF settings after the period ends.

**Error Handling**

* 401 – Unauthorized
* 500 – Internal Server Error
* 404 – Not Found
* 409 – Conflict

**Future Enhancements**

* Add support for Slack and Teams.
* Schedule OOF settings in advance.
* Add notifications when OOF settings are updated.

**Deployment Strategy**

* **Blue-Green Deployment** – Minimize downtime during releases.
* **Kubernetes-based Deployment** – Ensure high availability and auto-scaling.
* **Jenkins CI/CD** – Automate build and deployment.

**Summary**

The Delegate App provides a unified experience for managing Out of Office settings across multiple platforms using a scalable and flexible architecture. The combination of **Facade**, **Factory**, and **Adapter** patterns ensures clean separation of concerns and simplifies future extensions.

# Implementation Plan

**Implementation Plan for Delegate App**

This implementation plan outlines the key phases, deliverables, and timelines required to implement the Delegate App based on the system design. The plan includes a breakdown of tasks, resource allocation, and expected milestones.

**1. Project Phases and Timeline**

| **Phase** | **Duration** | **Key Deliverables** |
| --- | --- | --- |
| Phase 1: Planning and Setup | 2 weeks | Project scope, architecture setup, repository setup, environment setup |
| Phase 2: Backend Development | 4 weeks | Spring Boot microservices, authentication, and database setup |
| Phase 3: Frontend Development | 4 weeks | React-based UI, Redux state management |
| Phase 4: Integration | 3 weeks | Outlook 365, ServiceNow, and PPM integration using adapters |
| Phase 5: Testing | 2 weeks | Unit tests, integration tests, and performance tests |
| Phase 6: Deployment | 1 week | Kubernetes-based deployment, Jenkins CI/CD |
| Phase 7: Monitoring and Finalization | 1 week | Logging, monitoring, and final fixes |

**2. Phase Details**

**Phase 1: Planning and Setup**

Duration: 2 weeks  
Team Involved: Architect, DevOps, Backend and Frontend Leads

Tasks:

* Define project scope and milestones
* Set up Git repository and branching strategy
* Set up development and test environments (Docker, Kubernetes)
* Finalize database schema and API contracts

Deliverables:

* Architecture document
* Git repository and CI/CD pipeline configuration

**Phase 2: Backend Development**

Duration: 4 weeks  
Team Involved: Backend Developers, DevOps

Tasks:

* Create Spring Boot microservices
* Implement Factory Pattern for OOF handler creation
* Develop Facade Pattern to provide a unified OOF interface
* Implement OAuth2 authentication using WAM
* Build REST APIs for OOF creation, update, and cleanup
* Implement PostgreSQL persistence for OOF records

Deliverables:

* Backend microservices with authentication
* API endpoints for OOF and user management

**Phase 3: Frontend Development**

Duration: 4 weeks  
Team Involved: Frontend Developers

Tasks:

* Create React-based UI components using Redux for state management
* Develop profile management UI
* Implement OOF creation, update, and history display
* Build administrator panel for managing user settings
* Handle API integration for backend communication

Deliverables:

* Functional and responsive UI
* Role-based access control (employee/admin)

**Phase 4: Integration**

Duration: 3 weeks  
Team Involved: Backend Developers

Tasks:

* Implement Adapter Pattern for platform-specific OOF calls
  + OutlookAdapter → Outlook Graph API
  + ServiceNowAdapter → ServiceNow REST API
  + PPMAdapter → PPM REST API
* Implement logging and error handling
* Secure platform-specific credentials using a secure vault

Deliverables:

* Successfully integrated Outlook, ServiceNow, and PPM APIs
* Secure handling of authentication and secrets

**Phase 5: Testing**

Duration: 2 weeks  
Team Involved: QA Engineers, Developers

Tasks:

* Unit Testing (90% coverage)
* Integration Testing with platform-specific APIs
* Performance Testing for scalability (1000+ users)
* Security Testing (OAuth2, data encryption)
* Regression Testing

Deliverables:

* Test results with >95% pass rate
* Bug reports and fixes

**Phase 6: Deployment**

Duration: 1 week  
Team Involved: DevOps, Infrastructure Engineers

Tasks:

* Deploy microservices using Kubernetes
* Enable Blue-Green Deployment strategy
* Configure auto-scaling and load balancing
* Set up Splunk for logging and monitoring

Deliverables:

* Fully deployed Delegate App
* Working CI/CD pipeline using Jenkins

**Phase 7: Monitoring and Finalization**

Duration: 1 week  
Team Involved: DevOps, Developers

Tasks:

* Monitor system performance using Grafana and Prometheus
* Ensure successful OOF setup and cleanup in Outlook
* Handle final bug fixes and optimizations

Deliverables:

* Monitored and optimized system
* Performance benchmarks

**3. Resource Allocation**

| **Role** | **Resource** | **Involvement** | **Notes** |
| --- | --- | --- | --- |
| Project Manager | 1 | 100% | Oversee project timeline and coordination |
| Architect | 1 | 50% | Provide design guidance |
| Backend Developer | 3 | 100% | Develop microservices and integration layer |
| Frontend Developer | 2 | 100% | Build UI components |
| QA Engineer | 2 | 100% | Test and validate the system |
| DevOps Engineer | 2 | 100% | Setup CI/CD and infrastructure |

**4. Risk and Mitigation**

| **Risk** | **Impact** | **Likelihood** | **Mitigation Strategy** |
| --- | --- | --- | --- |
| API changes in Outlook, ServiceNow, PPM | High | Medium | Monitor API documentation, implement versioning |
| Scalability issues under heavy load | High | Low | Use Kubernetes for auto-scaling |
| Authentication failure | High | Low | Implement retries and error logging |
| Data inconsistency | Medium | Low | Ensure transactional integrity with ACID compliance |

**5. Success Criteria**

* Successful authentication and login for both employee and admin
* Ability to create, update, and delete OOF records across all platforms
* OOF cleanup batch job functioning as expected
* System capable of handling >1000 concurrent users without performance degradation
* 95% test pass rate for all functional and non-functional tests

**6. Milestones**

| **Milestone** | **Date** | **Deliverable** |
| --- | --- | --- |
| Phase 1 Complete | Week 2 | Architecture and repository setup |
| Phase 2 Complete | Week 6 | Backend microservices ready |
| Phase 3 Complete | Week 10 | Frontend interface ready |
| Phase 4 Complete | Week 13 | API integration done |
| Phase 5 Complete | Week 15 | Testing completed |
| Phase 6 Complete | Week 16 | Deployment successful |
| Phase 7 Complete | Week 17 | System running and monitored |

**7. Deliverables**

| **Deliverable** | **Description** | **Owner** | **Completion Date** |
| --- | --- | --- | --- |
| Architecture Document | High-level and detailed design | Architect | Week 2 |
| API Contract | REST API contract | Backend Team | Week 4 |
| Backend Microservices | Completed backend logic | Backend Team | Week 6 |
| Frontend Interface | Functional UI | Frontend Team | Week 10 |
| Integration with Platforms | Working integration with Outlook, ServiceNow, PPM | Backend Team | Week 13 |
| Testing Report | Test coverage, performance benchmarks | QA Team | Week 15 |
| CI/CD Setup | Jenkins pipeline and Kubernetes config | DevOps Team | Week 16 |

**8. Deployment Strategy**

* Blue-Green Deployment – Minimize downtime and rollback issues.
* Kubernetes-based Deployment – Auto-scaling and load balancing.
* Jenkins-based CI/CD – Automate build, test, and deployment processes.

**9. Completion Criteria**

* End-to-end user flow working correctly.
* All functional and non-functional requirements met.
* Successful production deployment with no critical issues.

**10. Summary**

This plan outlines a phased implementation strategy for the Delegate App, combining microservices, OAuth2 authentication, and external platform integration. The use of Facade, Factory, and Adapter patterns ensures clean separation of concerns and flexibility for future expansion. The plan ensures scalability, high availability, and secure handling of user data.