**Smart Service Request System**

**Phase 1: Problem Understanding & Industry**

**Analysis**

Goal: Understand what we’re building and why

# Requirement Gathering Steps

* **Identify Stakeholders** – Determine who will use or be impacted by the system (Customers, Staff, Managers).
* **Collect Customer Needs** – Ensure customers can easily raise and track service requests.
* **Collect Staff Requirements** – Provide a clear process for assignment, updates, and managing requests.
* **Collect Manager Expectations** – Capture the need for visibility into performance metrics and request statuses.
* **Document Requirements** – Record all gathered requirements to ensure SSRS aligns with practical business needs.

# Stakeholder Analysis Steps

* **Identify Stakeholder Roles**:
* **Admins** – Configure and maintain the system.
* **Support Staff** – Resolve service requests.
* **Managers** – Monitor and approve requests.
* **Customers** – Initiate service requests.
* **Analyze Stakeholder Expectations**:
* Admins need easy configuration and system management.
* Support Staff need a streamlined process to handle requests efficiently.
* Managers need dashboards and reports to track progress.
* Customers want a simple interface to submit and monitor requests.
* **Align System Features** – Design SSRS to meet the needs of all stakeholders effectively.

# Business Process Mapping Steps

* **Analyze Current Workflow** – Identify all stages:
* Logging the request
* Assigning staff
* Updating progress
* Closing once resolved
* **Identify Pain Points** – Highlight delays, missed updates, and lack of visibility.
* **Design Optimized Workflow** – Use automation and reporting to address inefficiencies.

# Industry-Specific Use Case Analysis Steps

* **Analyze IT Helpdesk Requirements** – Focus on SLAdriven ticketing and quick resolution.
* **Analyze HR Requirements** – Use service requests for employee support tasks.
* **Analyze Facilities Requirements** – Use service requests for task-based issues.
* **Adopt Flexible Features** – Implement workflows that work across multiple departments while keeping the system simple.

# AppExchange Exploration Steps

* **Review Relevant Apps** – Analyze IT ticketing apps and Service Cloud add-ons.
* **Identify Useful Features** – Look for automated routing, SLA tracking, and self-service portals.
* **Incorporate Best Practices** – Adapt proven features to SSRS to enhance functionality and efficiency.

**Phase 2: Org Setup & Configuration**

**Goal: Set up Salesforce environment for building SSRS.**

# Choose Salesforce Edition

* Use a Developer Edition Org (free) for development and testing.

# Configure Company Profile

* Enter organization details in *Company Settings*.
* Set local time zone and default currency (INR/USD).

# Define Service Hours & Holidays

* Create business hours (e.g., 9 AM–6 PM) for service request handling.
* Mark public holidays so requests are not escalated on non-working days.

# Set Fiscal Year

* Keep Standard Fiscal Year (Jan–Dec) to align reporting on requests with calendar year.

# Create Users

* Add users for Admin, Support Staff, and Manager roles.
* Assign appropriate Salesforce licenses.

# Assign Profiles

* **Admin Profile** – Full customization rights.
* **Support Staff Profile** – Limited to creating and updating service requests.
* **Manager Profile** – Extended access including approvals and dashboards.

# Define Role Hierarchy

* Place Manager role above Support Staff, ensuring visibility rolls up.
* Keep Customer Community/Portal users at the lowest level, limited to their own requests.

# Apply Permission Sets

* Use Permission Sets to grant extra access (e.g., Reports or Dashboards) without changing core profiles.

# Set Organization-Wide Defaults (OWD)

* **Service Request Object** → Private (only owner and manager can see).
* **Accounts/Contacts** → Public Read Only (for better linking and visibility).

# Establish Sharing Rules

* Add rules if requests should be shared across multiple teams or departments.

# Restrict Login Policies

* Limit login hours for Support Staff (e.g., 9

AM–6 PM).

* Apply IP restrictions for Admins for extra security.

# Prepare Development Org

* Use this Org as a sandbox for building and testing SSRS features.

# Sandbox Strategy

* In real-world implementation, develop in Sandbox and then promote changes to Production.

# Deployment Plan

* Use Change Sets to move configurations and customizations to Production after testing.

**Phase 3: Data Modeling & Relationships**

**Goal:** Build the data structure for SSRS.

# Step 1: Standard & Custom Objects

We use Salesforce’s standard objects such as Account and Contact to manage customer details, and create a custom object called Service Request to track issues raised by customers. This ensures we extend Salesforce without duplicating built-in features.

# Step 2: Fields for Service Request

The Service Request object is enriched with key fields:

* Request ID (Auto Number) for unique identification.
* Request Type (Picklist) to classify requests as IT, HR, or Facility.
* Priority (Picklist) to set urgency (High, Medium, Low).
* Status (Picklist) to track progress (New, Assigned, In Progress, Closed).
* Assigned Staff (Lookup to User**)** for responsibility tracking.
* Created Date (System) for timestamps.
* Resolution Notes (Long Text Area) for closure details.

These fields ensure all critical details are captured for reporting and resolution.

# Step 3: Record Types

Two record types are created:

* **Internal Requests** – raised by employees.
* **External Requests** – raised by customers.

This classification helps apply different page layouts and business rules for each scenario.

# Step 4: Page Layouts

Page layouts are configured to display relevant information:

* Service RequestLayout shows Request ID, Type, Status, Priority, Assigned Staff, and Resolution Notes.
* Contact Layout displays all related service requests raised by a contact.
* Account Layout shows service requests linked to that account.

Layouts improve usability by organizing fields logically.

# Step 5: Compact Layouts

For mobile users, compact layouts highlight the most important fields: Request ID, Priority, Status, and Assigned Staff. This allows quick scanning of request details on the go.

# Step 6: Schema Builder

Using Schema Builder, relationships are visualized:

* Account → Contact (1:M) oContact → Service Request (1:M) oService Request → User (M:1)

This graphical model helps stakeholders understand data relationships clearly.

**Step 7: Relationship Types**

* Contact ↔ Service Request → Lookup, since requests should remain even if a contact changes.
* User ↔ Service Request → Lookup, for flexible staff assignment.
* Account ↔ Contact → Standard One-to-Many, built into Salesforce.

Lookups are chosen over master-detail for flexibility and independent record ownership.

# Step 8: Junction Objects (if needed)

A junction object named Request Assignment can be introduced if a single request needs multiple staff members assigned. This enables many-to-many relationships.

# Step 9: External Objects (Optional)

If SSRS needs to integrate with external systems (e.g., a TCS knowledge base), External Objects can be used to access external data in real-time without duplication.

**Phase 4: Process Automation (Admin)**

**Goal:** Automate SSRS workflows to ensure faster resolution, accuracy, and accountability.

# Step 1: Validation Rules

Validation rules enforce data quality by preventing incomplete or invalid entries. oExample: Prevent closing a Service Request unless **Resolution Notes** are filled.

oExample: Ensure **Priority** is selected before saving a request.

These rules maintain data consistency and reliability across the system.

# Step 2: Workflow Rules

Workflow rules are configured to trigger simple automated actions.

* Example: Send an **email alert** to assigned staff when a new request is created.
* Example: Notify customers when the **status changes to Closed**.

Workflows reduce manual communication and speed up response times.

# Step 3: Process Builder

Process Builder is used for more advanced, conditional automation.

* Example: Automatically **assign requests** based on type (IT requests go to IT team, HR requests go to HR team).
* Example: Update **Status = In Progress** when a staff member is assigned.

This ensures requests are routed and updated automatically.

# Step 4: Approval Process

Approval processes add structured oversight for critical cases.

* Example: **High-priority requests** must be approved by a Manager before closure.
* Example: Escalation approvals for overdue requests.

This improves accountability and ensures sensitive cases get managerial attention.

# Step 5: Notifications & Alerts

Real-time alerts improve visibility.

oEmail and in-app notifications for **new assignments**. oPush notifications for **overdue requests**.

This ensures staff and managers are always aware of request progress.

# Step 6: Escalation Rules

Escalation rules automatically reassign or escalate unresolved cases.

* Example: If a request is not updated within **48 hours**, escalate to Manager.

This helps maintain SLAs and prevents customer dissatisfaction.

# Step 7: Testing Automation

Each automation is tested with sample data.

* Create test requests to confirm validation rules work.
* Simulate request assignment and closure to verify workflows, approvals, and escalations.

Testing ensures automation behaves correctly before production deployment.

With these automations, Smart Service Request System becomes a **smart system** that reduces manual effort, prevents errors, and ensures timely resolution of service requests.

**Phase 5: Apex Programming (Developer)**

**Goal:** Add custom logic and automation to enhance Smart Service Request System beyond point-and-click configuration.

# Apex Triggers

* **Auto-generate Request ID:** A trigger ensures each Service Request gets a unique ID if not already assigned.
* **Validation on Closure:** Prevents a request from being closed unless **Resolution Notes** are entered.
* **Status Updates:** Automatically update the related **Account or Contact** with the latest service request status.

# Apex Classes

* **Escalation Handler:** A class that checks overdue requests and sends escalation notifications to managers.
* **Assignment Logic:** Assigns incoming service requests to staff based on workload, category, or priority.
* **Utility Methods:** Reusable methods for logging actions, formatting data, or sending custom notifications.

# Batch Apex

* **Weekly Status Update:** A batch job that reviews all open service requests at the end of the week and sends summary reports to managers.
* **Data Cleanup:** Batch classes can archive closed requests older than a specific time period.

# Test Classes

* **Unit Tests:** Cover triggers, classes, and batch processes with test data.
* Ensure at least **75% code coverage**, as required by Salesforce for deployment.
* Include **positive, negative, and bulk test cases** to ensure system reliability.

# Error Handling & Logging

* Use **try-catch blocks** in Apex to handle unexpected errors gracefully.
* Implement a **custom error log object** to capture failures (e.g., assignment errors, escalation failures).
* Send admin alerts when critical errors occur.

# Future Enhancements

* Add **Queueable Apex** for more complex asynchronous processing.
* Use **Platform Events** for real-time notifications when a high-priority request is escalated.

With Apex programming, Smart Service Request System gains flexibility for complex logic such as automatic assignment, escalations, and reporting, which cannot be achieved with clicks alone.

**Phase 6: User Interface Development**

**Goal:** Build a user-friendly interface in Salesforce to simplify service request creation, tracking, and management.

**41 .Lightning Record Pages**

* + Create a **custom record page** for the **Service Request** object.
  + Use the **Lightning App Builder** to organize fields into sections (Request Details, Assignment, Resolution).
  + Add related lists like **Contacts** and **Accounts** for quick reference.

**42.Quick Actions**

* + Add global and object-specific quick actions:
    - **Log Service Request** → allows staff to quickly create a new request.
    - **Reassign Staff** → lets managers reassign a request in fewer clicks.
    - **Close Request** → updates status and prompts for resolution notes.

**43.Dynamic Forms**

* + Configure **Dynamic Forms** on Service Request pages.
  + Show or hide fields based on conditions:
    - If Status = “Closed”, then display **Resolution Notes**.
    - If Request Type = “IT”, show **Technical Details** section.

**44.App Navigation**

* + Create a dedicated app called **Smart Service Request**.
  + Add navigation tabs for **Accounts, Contacts, Service Requests, Reports, and Dashboards**.
  + Keep navigation simple so staff can move between objects without confusion.

**45.Related Lists & Components**

* + On the **Contact page**, display a related list of Service Requests raised by that contact.
  + On the **Account page**, display all Service Requests linked to that account.
  + Use components like **Highlights Panel** to show key fields (Request ID, Priority, Status).

**46.UI Testing**

* + Test layouts in both **desktop and Salesforce Mobile App**.
  + Ensure that **compact layouts** display the most important information (Request ID, Priority, Status, Assigned Staff).
  + Collect feedback from staff and managers to refine the interface.

With this setup, Smart Service Request System provides an **intuitive and responsive user interface**, ensuring staff, managers, and customers can access and manage requests efficiently.

**Phase 7: Integration & External Access**

**Goal:** Enable customers and external systems to interact with SSRS through secure integrations and access points.

# 47.Email-to-Case

* Configure **Email-to-Case** so that when a customer sends an email, it automatically creates a Service Request record in Salesforce.
* Route the request to the appropriate staff queue based on subject or priority.

# 48.Web-to-Case

* Create a **public-facing web form** that allows customers to submit requests directly.
* The submitted form populates fields like Request Type, Priority, and Contact.
* Add **reCAPTCHA** for spam protection.

# 49.REST API Integration

* Expose SSRS data using **Salesforce REST API** so external apps (e.g., customer portals, mobile apps) can push or pull request data.
* Use **OAuth 2.0** for secure authentication.

# 50.Experience Cloud Portal

* Set up an **Experience Cloud site** for customers.
* Customers can log in to view their service request history, track status, and communicate with staff.
* Managers can monitor feedback and request activity through the portal.

# 51.External System Integration

* Integrate with external systems like a **knowledge base** or third-party support platform.
* For example, requests can link to relevant knowledge articles to help staff resolve issues faster.

# 52.Security for Integrations

* Implement **OAuth 2.0 and Named Credentials** for API security.
* Use **Profiles, Permission Sets, and Sharing Rules** to control what external users can access.
* Enable **audit trails** to monitor external access activity.

# 53.Testing & Validation

* Test Email-to-Case and Web-to-Case with sample submissions.
* Verify API requests and responses using tools like **Postman**.
* Ensure external portals show correct data without exposing sensitive information.

With Phase 7, Smart Service Request System is no longer limited to internal users — customers and external systems can securely create, track, and resolve requests, making the system more powerful and flexible.

**Phase 8: Data Management & Deployment**

**Goal:** Ensure clean data, safe migration, and smooth deployment of the Smart Service Request System to production.

# 54.Sample Data Import

* Use **Data Import Wizard** to load sample Accounts, Contacts, and Service Requests.
* Validate that relationships (Account → Contact → Service Request) are correctly maintained after import.

# 55.Data Quality Rules

* Create **Validation Rules** to prevent incomplete records (e.g., Resolution Notes required before closure).
* Set up **Duplicate Rules** to stop duplicate Contacts or Service Requests.
* Apply **Picklist Value Restrictions** to maintain consistency.

# 56.Backup Strategy

* Schedule regular **Data Export** in Salesforce to back up Accounts, Contacts, and Service Requests.
* Store backup files securely for recovery in case of accidental data loss.

# 57.Deployment Tools

* Use **Change Sets** to move customizations (objects, fields, layouts, automation, Apex) from **Sandbox to Production**.
* Ensure dependencies (like fields, validation rules) are included in the deployment.
* For larger projects, consider using **Salesforce CLI or ANT Migration Tool**.

# 58.Sandbox Usage

* Perform all development and testing in a **Sandbox environment**.
* Only push final changes to Production after successful testing.
* Use **Partial Copy Sandbox** if real test data is required.

# 59.Post-Deployment Testing

* After deploying to Production, test full scenarios:
  + Create → Assign → Resolve a Service Request. oValidate reports and dashboards.
  + Check integrations (Email-to-Case, API, Experience Cloud).
* Fix any issues before go-live.

# 60.End-User Training & Documentation

* Provide staff and managers with training sessions or user guides.
* Share documentation on how to create, assign, and close requests.
* Educate admins on backup, monitoring, and troubleshooting.

With Phase 8, the Smart Service Request System ensures **data integrity, safe deployment, and smooth transition to production**, reducing risks of errors or downtime.

**Phase 9: Reporting, Dashboards & Security**

**Review**

**Goal:** Provide insights into system performance while ensuring data security and compliance.

# 61.Custom Reports

* Create reports to track service requests by **status, priority, request type, and assigned staff**.
* Build reports showing **average resolution time**, **open vs closed requests**, and **overdue requests**.
* Use **summary and matrix reports** for workload distribution analysis.

# 62.Dashboards

* Design dashboards for different stakeholders:
  + **Manager Dashboard** → shows workload distribution, SLA compliance, and overdue requests.
  + **Support Staff Dashboard** → displays assigned requests, pending tasks, and resolution trends.
  + **Executive Dashboard** → provides overall service performance and customer satisfaction metrics.
* Use charts such as bar graphs, pie charts, and gauges for better visualization.

# 63.Security Review

* Perform a **profile and permission set review** to ensure correct access levels.
* Confirm that sensitive fields (like Resolution Notes) are hidden from unauthorized users.
* Validate **record-level access** so that support staff only see their assigned requests, while managers have broader visibility.

# 64.Sharing Rules

* Configure **sharing rules** to extend access when collaboration is required (e.g., another team member needs visibility into a request).
* Ensure **private access** for sensitive service requests by default.

# 65.Audit Trail

* Enable the **Setup Audit Trail** to track configuration changes.
* Use **Field History Tracking** on critical fields like Status

**Phase 10: Final Presentation & Demo Day**

**Goal:** Demonstrate the full working Smart Service Request System to stakeholders and submit documentation.

**66. Prepare End-to-End Demo**

* Start from **Account creation → Contact → Service Request logging → Assignment → Resolution**.
* Show how automation works (e.g., request auto-assigned, approval process, validation on closure).
* Demonstrate **mobile view** (Compact Layouts) and **Experience Cloud portal** (customer login).

**67. Dashboard Showcase**

* Open the **Manager Dashboard** to show workload distribution, SLA performance, and overdue requests.
* Show **Staff Dashboard** to display assigned tasks and pending work.
* Present **Executive Dashboard** to highlight customer satisfaction metrics and service trends.

**68. Collect Stakeholder Feedback**

* Ask managers if reports/dashboards give them the right insights.
* Ask staff if request logging and tracking is simple enough.
* Ask customers (via portal demo) if request tracking is clear.

**69. Submit Project Documentation**

* Compile **Requirement Document, ERD, Screenshots, Phase-by-Phase Explanation, and Apex Code Samples**.
* Export documentation as **PDF / Word**.
* Upload to GitHub or share with stakeholders as final deliverable.

**70. Q&A Session**

* Answer queries about:
  + Data model (Accounts, Contacts, Service Requests).
  + Automation (validation, workflows, approvals).
  + Security (profiles, sharing rules).
  + Integrations (Email-to-Case, APIs, Portal).
* Clarify how the Smart Service Request System can be extended in the future.

With Phase 10, the **Smart Service Request System** project is officially completed, demonstrated, and documented, showing its ability to **streamline service request management from start to finish**.