Project Name: MultiSure360 – Unified Insurance Management System

---

□ Project Scope:

A Constellation-enabled app that allows:

- Policy creation (Health, Motor, Life, Travel)
- Customer onboarding
- Claims registration & processing
- Policy rating & renewal
- Integration with external systems (for KYC, claim validation, notifications)

□ Core Case Types

Case Type ID	Description
`NewCustomer`	Onboard a new insurance customer
`CreatePolicy`	Create a new insurance policy
`ManageClaim`	Register and process insurance claims
`RateInsurer`	Rate insurer based on service and payouts
`RenewPolicy`	Policy renewal process

Each case will follow **Constellation** design standards using **View Templates**, **Field Groups**, and **Embedded Pages**.

□ Phase 1 – Application Setup (Constellation UI)

➤ Step-by-Step:

1. Create Application in App Studio

Field	Value
Application Name	**MultiSure360**
Built-on App	**Theme-Cosmos-React** (Constellation)
App Template	Custom
Primary Class	`MultiSure360-Work`
Data Class Layer	`MultiSure360-Data-*`
Organization	`MultiSureCorp`

Field	Value
Ruleset	`MultiSure360`

Ensure: □ **Use Constellation UI** is selected (NOT Cosmos React Classic).

□ Concepts Covered in This Project

Category	Concepts Covered
□ App Setup	Application structure, Access Group, Channels, Portals
□ UI in Constellation	View Templates, Panels, Regions, Field Groups, View Reuse
□ Case Mgmt	Multi-stage flows, Conditions, Alternate Stages, Temporary Work Objects
□ Data	Data Types, Reference Data, Embedded Pages, Customer-Policy linking
□ Integration	KYC via REST, Claim Validation APIs, OTP Services
□ Security	Role-based controls, Access Groups, Authorization
□ Testing	PegaUnit, Scenario Tests, Field Value Assertions
□ Deployment	Product Rule, Deployment Pipeline (Dev → QA → Prod) via Deployment Manager
□ Reporting	Live UI Dashboards, Export Reports, PDF Summary Generation
□ Dev Studio Edge	Custom Data Transforms, Activities (safe), Validation Rules
□ AI/Decisioning	Eligibility Decision Tables, Claim Risk Model (placeholder)
□ Channels	Web portal, Mobile channel, Email Templates, Chat IVA for Claims
□ DX API	Expose policy and claim data for external consumption
□ Performance	DLP Usage, Constellation Rendering, View Optimization, Clipboard minimalism
□ Accessibility	WCAG-Compliant templates, Localization, Readout support

---

□ Let's Begin Phase 2: Define the First Case Type – `NewCustomer`

□ Full Inclusion Checklist for Constellation Master Project – *MultiSure360*

□ Category	□ Items Covered
□ **Application Setup**	App creation, class structure, ruleset, Constellation UI shell
□ **Case Types**	`NewCustomer`, `CreatePolicy`, `ManageClaim`, `RenewPolicy`,

□ Category	□ Items Covered
	`RateInsurer`
□ **UI (Constellation)**	View Templates, Field Groups, Regions, Embedded Views, Panels, Reusable Views, Reference views
□ **Case Life Cycle**	Stages, Steps, Alternate Stages, Temporary objects, Parallel flows, SLAs, Optional Actions
□ **Data Layer**	Data Types (`Customer`, `Policy`, `Claim`, `Insurer`), Page Structures, Embedded Pages, Referential Integrity, Relationship modeling
□ **Integrations**	REST APIs for KYC, Claims Check, OTP, and External Rating Engines
□ **Security & Access**	Authentication, Authorization (Access Groups/Roles/Privileges), Field-Level Security, Access Control Policies (ABAC if needed)
□ **Testing**	PegaUnit, Scenario Tests, Data Validation, View Tests
□ **DevOps**	Branching Strategy, Merge Conflicts, Deployment Pipeline using Deployment Manager, Product Rule Packaging
□ **Reporting & Insights**	Report Definitions, Live Dashboards, Exports (Excel, PDF), Summary Views
□ **Decisioning (Basic)**	Claim eligibility via Decision Table, Risk scoring using Data Transforms / Declare Expression
□ **Channels & IVA**	Web Portal, Mobile-ready UI, Email OTP, Pega Intelligent Virtual Assistant (Chatbot) for claims and policy inquiries
□ **DX API**	RESTful APIs to expose policy & claims data for Power BI / external portals
□ **Mobile Optimization**	Responsive views, Mobile channel configurations, Image uploads for claims
□ **Localization**	Field values, labels, date formats, support for i18n
□ **Accessibility (WCAG)**	Constellation templates compliance with WCAG 2.1, ARIA support, color contrast validation
□ **Performance & PAL**	Optimized Data Pages (DLP), Smart loading of views, minimal clipboard, using new client-rendering strategies
□ **Custom Logic**	Declarative Rules, Custom Data Transforms, Activities (limited), Validation Rules
□ **Attachments & Uploads**	File attachments, drag-and-drop (for KYC/claims), preview in view
□ **Guardrails & Score**	Pega Guardrail Score compliance, Rule warnings fixing, Rule reuse
□ **Documentation & Help**	Embedded contextual help, tooltips, dynamic instructions
□ **End-to-End Scenario**	Full scenario: Onboard → Create Policy → File Claim → Renew Policy → Rate Insurer → Export insights → Push to external consumers via API

□ Optional Advanced Extras (If You Want to Go *Ultra-Master Level*)

□ Feature	Description
CDH-lite Strategy Integration	Mini next-best-action strategy for policy recommendations
Kafka Integration (for Claims)	Push claim notifications to a Kafka topic
Push to Power BI	Use Service REST + JSON for Power BI to consume report data
BIX Extraction	For nightly policy export to warehouse
Pulse & Collaboration	Add Pulse widget in policy view for underwriters
AI Predictions	Use Prediction Studio model for fraud detection (mocked)
Job Schedulers/Queue Processors	Schedule nightly rating updates or retry failed policy activations
Constellation SDK/Custom DXC	Embed custom React components inside a region (like rating sliders or insurer graphs)

□ Final Project Scope: MultiSure360 – Elite Edition

Category	Full Features Included
□ Core Case Types	NewCustomer, CreatePolicy, ManageClaim, RenewPolicy, RateInsurer
□ UI (Constellation)	View templates, reusable embedded views, panels, responsive & mobile-ready
□ Data Model	Customer, Policy, Claim, Insurer, DocumentAttachment, Ratings
□ Integrations	REST APIs for KYC, claim validation, OTP; DX API for Power BI; Kafka events for Claim status
□ Security	RBAC (Roles/Privileges), authentication, field-level access control, Access Groups
□ Testing	PegaUnit, Scenario Testing, Mocked responses
□ DevOps	Deployment Manager pipeline, Product Rules, branch merging, guardrail compliance
□ Reporting	Report Definitions, Dashboards, Power BI push, PDF generation
□ Decisioning	CDH-lite strategy for “Next Best Policy” recommendation + Fraud Prediction model
□ Performance	DLP, view optimization, clipboard usage, PAL inspection
□ Accessibility	WCAG compliance, i18n localization, screen reader support
□ Channels	Web, Mobile, Email OTP, Chatbot (IVA) for claims and renewals
□ Extensibility	Custom DX Components (e.g., policy comparison graph, dynamic rating slider), Pulse integration
□ Background Jobs	Job Scheduler for premium reminders, Queue Processor for failed claim retries
□ Data Export	BIX configuration for nightly extract of Policy & Claim records
□ Documentation	Contextual help, labels, tooltips, instructions, audit logs

□ Implementation Flow – End-to-End

□ Phase 1: Application Setup

☑ Create app with Constellation UI
☑ Define base class structure and rulesets

□ Phase 2: Case Types

1. `NewCustomer` – KYC flow + OTP
2. `CreatePolicy` – With branch by insurance type (health, motor, life, travel)
3. `ManageClaim` – File, validate (API), approve/reject with parallel SLAs
4. `RenewPolicy` – Auto-renew with Job Scheduler + optional customer action
5. `RateInsurer` – Use decisioning & Pulse to collaborate with underwriters

□ Phase 3: Integrations

☑ REST (KYC, Claim Check, OTP)
☑ Kafka (Claim processed event)
☑ Power BI (via custom REST endpoint + JSON contract)
☑ DX API exposure
☑ IVA chatbot for "Policy Info", "Renew", "Raise Claim"

□ Phase 4: Data Model & View Structure

☑ Build Data Types, link to embedded views
☑ Create Field Groups and reference views for each policy type

□ Phase 5: AI & CDH-lite

☑ Strategy to recommend policy bundles (mocked)
☑ Prediction rule for high-risk claims

□ Phase 6: Reporting & BIX

☑ Create embedded dashboards
☑ Configure BIX extract rule
☑ Generate REST API for Power BI consumption

□ Phase 7: DevOps

☑ Setup Product rule
☑ Create branches for case types
☑ Build Deployment Manager pipeline

□ Phase 8: DX Components & Pulse

☑ Embed React slider for policy rating
☑ Graph widget for policy premium history
☑ Enable Pulse in `RateInsurer` case type

□ Phase 9: Testing

☑ PegaUnit for all flows
☑ Scenario test for UI
☑ Field validations, REST mock tests

□ Final Outcome:

An **enterprise-grade, modern, intelligent, cloud-ready, multi-insurance portal** using **Constellation + Decisioning + DX + Automation + DevOps + Insights**—a complete demonstration of **everything a senior Pega architect should know**.

□ PHASE 1: Application Setup with Constellation (Ultra-Beginner Friendly)

□ GOAL:

We will create a brand-new Pega application that:

- Uses **Constellation UI**
- Supports **MultiSure360** case types
- Is ready for future development in App Studio and Dev Studio

---

□ STEP-BY-STEP GUIDE

□ STEP 1: Login to Pega Dev Studio

1. Open your browser and go to your Pega 8.7+ or Pega 24 environment.
2. Login with a user who has `Administrator` privileges (e.g., `admin@pega.com`).
3. After logging in, go to the top-right corner → click on your operator name → **Switch to Dev Studio**.

□ STEP 2: Create a New Application Using the Wizard

1. Go to the **Application** menu → **New Application**.
2. Select **Custom** (DO NOT choose Industry templates).
3. In the **UI architecture**, select:

```
CopyEdit
□ Constellation
```

□ *DO NOT choose Theme-Cosmos (Classic).*

□ STEP 3: Fill in App Configuration

Field	Value
Application Name	`MultiSure360`
Organization Name	`MultiSureCorp`
Division	`Insurance`
Unit	`UnifiedMgmt`
Base Class	`MultiSure360-Work`
Ruleset	`MultiSure360`
Primary Persona	`Case Manager`
Development Team	`MultiSure360Team`

□ Note: These values help auto-generate class structure and ruleset names.

4. Click **Create Application**.

□ STEP 4: Confirm the Application Was Created with Constellation

Once created:

1. You'll be redirected to **App Studio**.
2. Click **Preview Portal**.
3. You should see a **minimalist UI** with:
   - Navigation bar on the left (Home, My Work)
   - Blank landing page
   - "+ Create" case button (empty for now)

□ This confirms your app is built on **Constellation Shell UI**.

□ STEP 5: Set Up Developer Access

To avoid permission issues later:

1. Go to **Records → Security → Access Group**.
2. Open the access group for your application: `MultiSure360:Administrators`.
3. Confirm:
    - Application points to `MultiSure360`
    - Roles include: `PegaRULES:SysAdm4`, `PegaRULES:SecurityAdministrator`
    - Available portals include: `User`, `Dev Studio`, `App Studio`
4. Add your operator ID to this access group:
    - Go to **Records → Security → Operator ID**.
    - Open your operator (e.g., `admin@pega.com`).
    - Update **Access Group** to: `MultiSure360:Administrators`.
    - Save and log out → log back in.

□ STEP 6: Validate That App Studio is Constellation-Ready

1. Switch to **App Studio**.
2. Go to **Channels** → Select **Web**.
3. Click on **Web Channel Interface** named `MultiSure360`.
4. You should see a layout that says "Constellation is enabled."

□ Now you are fully ready to start building your first case types using **Constellation rules and patterns**.

□ NEXT UP:

Phase 2: Build `NewCustomer` case type with Constellation views.

It will include:

- Basic Customer data entry form
- OTP verification (mocked)
- KYC document upload
- Review summary
- Confirmation screen

□ PHASE 2: Building `NewCustomer` Case Type (Customer Onboarding)

□ GOAL:

To allow a new user to register into the MultiSure360 system with:

- Personal details
- Email/mobile OTP verification (simulated)
- Uploading KYC documents
- Review screen
- Confirmation + Case ID

□ STEP-BY-STEP GUIDE

□ STEP 1: Create the Case Type in App Studio

1. Switch to **App Studio**.
2. From the left nav bar, click **Case Types**.
3. Click **+ Add case type**.

Fill in:

Field	Value
Case Name	`NewCustomer`
Description	`Onboard a new customer with verification and KYC`
Starting process	Keep default

4. Click **Submit**.

➡□ You'll now be inside the **Case Designer**.

□ STEP 2: Define the Case Stages & Steps

We'll use **4 stages**:

Stage Name	Steps
1. Enter Details	- Collect Customer Info (View)
2. Verification	- Simulate OTP Check
3. Upload KYC	- Upload Document
4. Confirmation	- Review & Confirm

Modify stage names:

1. Click each stage name → Rename.
2. Add new stages using **+ Stage**.

□ STEP 3: Step 1 – Enter Customer Details

1. In **Stage 1: Enter Details**, click on the step (default Collect Info).
2. Rename it to **Enter Personal Info**.
3. On the right panel → click **Configure View**.
4. Use **Collect Information** view template.
5. Add the following fields:

Field Label	Type	Data Model Path (auto)
Full Name	Text	.FullName
Date of Birth	Date	.DateOfBirth
Email	Email	.Email
Mobile Number	Phone	.PhoneNumber
Preferred Contact	Dropdown	.PreferredContact (Choices: Email, SMS)

□ *Constellation auto-creates the fields in* `NewCustomer` *case class.*

6. Click **Submit**.

□ STEP 4: Step 2 – OTP Verification (Simulated)

1. Add a new step in Stage 2 → Click **+ Step → More → Automations → Send Email**.
2. Rename step: **Send OTP to Email**.
3. In configuration:
   - Subject: `Your One-Time Password (OTP)`
   - Body: `Your OTP is 123456. Please enter this to continue.`
4. Add a second step → **Collect Info** → Label: `Enter OTP`.
5. Add field: `.EnteredOTP` (Text Input).
6. Add a **When rule** for validation:
   - Click on View → Add Validation → If `.EnteredOTP ≠ "123456"`, show message: "Invalid OTP."

□ STEP 5: Step 3 – Upload KYC Document

1. Add new step in Stage 3 → **Collect Information** → Label: `Upload KYC`.
2. Add field:
   - Field Label: `Upload ID Proof`
   - Type: File
   - Property: `.KYCUpload`

□ This enables users to upload a PDF or image (Aadhaar, PAN, etc.)

□ STEP 6: Step 4 – Review & Confirm

1. In Stage 4, add step: **Review**.
2. Select **View summary of case info**.
3. This shows all captured fields in read-only mode.
4. Add final step: **Approval → Confirm Submission**.

□ STEP 7: Save and Run

1. Click **Save**.
2. Go to the main dashboard.
3. Click **+ Create → NewCustomer**.

You'll now see the full **Constellation View-based onboarding flow** in action:

- Responsive UI
- Validations
- File Upload
- Case ID generated on submit

IVKPEGA@GMAIL.COM

□ WHAT YOU'VE LEARNED

Concept	Applied In
View Templates	Used "Collect Information", "Review" views
Field Groups	Auto-created field groups under `.NewCustomer`
File Upload	Added `KYCUpload` field with drag-drop
View Validation	Used conditional validation for OTP
Constellation UX	View rendered on the client, zero clipboard bloat
App Studio Modeling	100% built in App Studio – ideal for Business-IT collaboration

---

□ **NewCustomer case type complete.**

□ PHASE 2: `CreatePolicy` Case Type – Policy Creation with Branching

□ GOAL:

To allow users to create a new insurance policy based on the type selected:

- **Motor**, **Health**, **Life**, or **Travel**
- Dynamically show different fields per policy type
- Show premium summary
- Confirm before submission

□ STEP-BY-STEP GUIDE

□ STEP 1: Add the `CreatePolicy` Case Type

1. In **App Studio**, go to **Case Types**.
2. Click **+ Add case type**.
3. Case name: `CreatePolicy`
4. Description: `Initiate a new insurance policy application`
5. Click **Submit**.

□ STEP 2: Define Stages & Flow Steps

Stage	Steps
1. Policy Type	Choose Insurance Type
2. Policy Details	Branch: Motor / Health / Life / Travel
3. Summary	Show calculated premium & policy info
4. Confirmation	Submit and generate Policy ID

1. Rename stages accordingly.
2. Add steps under each stage.

□ STEP 3: Stage 1 – Select Policy Type

1. In **Stage 1**, rename step: `Choose Type`.
2. Click **Configure View**.
3. Use **Collect information**.
4. Add:
    - Field: `Policy Type` (Dropdown)
    - Property: `.PolicyType`
    - Options:
        - Motor
        - Health
        - Life

    - Travel
5. Submit view.

□ STEP 4: Stage 2 – Branch by Policy Type (Smart Shape!)

1. In Stage 2, click **+ Step → More → Automation → Decision → Smart Shape: Decision**.
2. Name it: `Branch by PolicyType`.
3. Drag **Connectors** from this decision shape to four different views:
    - **Motor Details**
    - **Health Details**
    - **Life Details**
    - **Travel Details**

Constellation does not use full flow shapes like Dev Studio, but branching is supported via conditions in step-level visibility.

Instead of classic flows, do this:

Step 1: Add 4 steps → one for each type.

Step Name	Type	Fields to Add	Visibility Condition
Motor Details	Collect Info	Vehicle Number, Type, Engine CC	`.PolicyType == "Motor"`
Health Details	Collect Info	Family Members, Pre-existing Diseases	`.PolicyType == "Health"`
Life Details	Collect Info	Annual Income, Smoker?	`.PolicyType == "Life"`
Travel Details	Collect Info	Destination, Travel Dates	`.PolicyType == "Travel"`

□ **Set visibility condition on each step:**

- Go to step → Right pane → Configure View → Settings tab → *When* → Enter visibility condition like `.PolicyType == "Motor"`

□ *Constellation renders only that step based on the choice made.*

□ STEP 5: Stage 3 – Policy Summary

1. Add step → Label: `Show Policy Summary`.
2. Use **View Summary of Information**.
3. This automatically renders a read-only view of all collected fields.
4. Optional: Add a calculated field `.EstimatedPremium`.

- Calculate premium based on dummy logic:
  - Motor: ₹5000
  - Health: ₹10,000
  - Life: ₹15,000
  - Travel: ₹3000

5. Display premium in this step.

□ STEP 6: Stage 4 – Confirmation

1. Add step → Label: `Confirm Policy`.
2. Use **Approval** or **Submit** step.
3. Enable "Show case ID on confirmation screen."

□ STEP 7: Data Model Auto-Generated

From your steps, Constellation has created these:

Field Name	Type
`.PolicyType`	Text
`.VehicleNumber`	Text
`.FamilyMembers`	Integer
`.Income`	Currency
`.EstimatedPremium`	Decimal
`.TravelDestination`	Text
`.TravelDates`	Date Range

If you want centralized models later, we can create **Field Groups** like `.MotorDetails`, `.HealthDetails`, etc.

□ STEP 8: Test the Case

1. Click **Save**.
2. Go to Home → Click **+ Create → CreatePolicy**.
3. Try all four flows:
   - Choose "Motor" → Only Motor fields visible
   - Choose "Life" → Only Life fields shown
4. On Summary screen, check that `.EstimatedPremium` is calculated and displayed.

□ OPTIONAL: Auto-calculate EstimatedPremium

To simulate backend logic:

1. Add **Data Transform**: `CalculatePremium`
2. Add logic:

```
plaintext
CopyEdit
When .PolicyType == "Motor"
→ Set .EstimatedPremium = 5000
When .PolicyType == "Health"
→ Set .EstimatedPremium = 10000
...
```

3. In App Studio → Select Summary step → Open **Step Settings** → Add pre-processing Data Transform.

□ WHAT YOU'VE MASTERED

Concept	How You Applied It
Conditional View Steps	Only show step based on `.PolicyType`
Dynamic UI	Adjust views without branching flows
Data Transform	Auto-populate calculated field
Summary View	Auto-read-only summary without extra setup
Reuse and Modularity	One case type handles 4 policy types

□ `CreatePolicy` Case Type DONE.

□ **Next up:** Shall we proceed to build **`ManageClaim` case type** with file upload, REST validation mock, and SLA parallel review steps?

It will include:

- Upload Claim Form
- Auto-check policy validity via REST (mock)
- Assign parallel approval and fraud review (parallel flow)
- Final approval/rejection

□ PHASE 2 :`ManageClaim` Case Type – File Upload, REST Validation, SLA & Parallel Processing

□ GOAL:

Allow policyholders to file a claim, validate eligibility via REST (mocked), upload evidence, and undergo approval + fraud checks in parallel before settlement.

□ STEP-BY-STEP GUIDE

□ STEP 1: Add the `ManageClaim` Case Type

1. In **App Studio**, go to **Case Types**.
2. Click **+ Add case type**.
3. Case name: `ManageClaim`
4. Description: `File and process an insurance claim with necessary validations`
5. Click **Submit**.

□ STEP 2: Define Stages & Flow Steps

Stage	Steps
1. Initiate Claim	Collect Claim Info + Upload Files
2. Validate Policy	Call REST service to check policy coverage (mocked)
3. Review Claim	Parallel: Approver Review + Fraud Check
4. Settlement	Consolidate results and approve/reject + Confirm

Let's implement this step-by-step.

□ STEP 3: Stage 1 – Claim Initiation

□ Step 1.1: Collect Claim Details

1. Add step → Label: `Enter Claim Details`
2. Use **Collect information**.
3. Add fields:

Label	Type	Property Name
Policy Number	Text	`.PolicyNumber`
Claim Type	Dropdown	`.ClaimType`
Incident Date	Date	`.IncidentDate`
Description	Paragraph	`.ClaimDescription`

Use dropdown options like: Motor, Health, Life, Travel.

□ Step 1.2: Upload Evidence

1. Add another step in same stage → Label: `Upload Documents`
2. Use **Collect information**
3. Add field:
    - Label: `Upload Supporting Files`
    - Type: File
    - Property: `.EvidenceUpload`

You can allow `.pdf`, `.jpg`, `.png`.

□ STEP 4: Stage 2 – Validate Policy (Mock REST Call)

Step: Call REST to Validate Policy

We'll simulate a REST call using **Data Page** + **Connector Simulation**.

□ STEP A: Create a REST Connector Rule

1. In **Dev Studio → Records → Integration-Connectors → Connect REST**
2. Create:
    - Name: `ValidatePolicyConnector`
    - URL: `https://mock.policy/check/{PolicyNumber}`
    - Method: `GET`

Mock response:

```
json
CopyEdit
{
  "status": "Valid",
  "policyHolder": "John Doe",
  "validUntil": "2026-03-31"
}
```

3. Add **Simulated Response** in the **Simulation Data** tab.

□ STEP B: Create Data Page

- Name: `D_ValidatePolicy`
- Scope: Requestor
- Data Source: Connector → `ValidatePolicyConnector`
- Parameter: `PolicyNumber`

- Response Data Class: `MultiSure360-Data-PolicyValidation`

Add fields: `status`, `policyHolder`, `validUntil`

□ STEP C: Call in Flow

Back in **App Studio**:

1. Add step in Stage 2 → Label: `Check Policy Status`
2. Type: **Automation → Data Transform**
3. Create Data Transform: `CallValidatePolicy`
    - Set `.ValidationStatus = D_ValidatePolicy[.PolicyNumber].status`
4. Show result on a **read-only view**:
    - Add new **Collect Info** view: Label `Show Policy Check`
    - Show `.ValidationStatus`

□ STEP 5: Stage 3 – Parallel Review (Approver + Fraud)

This uses Constellation's **parallel process** smart shape.

1. In Stage 3, add **Parallel Process**.
2. Add **2 assignments inside it**:
    - **Step 1: Approver Review**
        - Assigned to: `Approver@MultiSureCorp`
        - Collect: Comments (`.ApproverComments`)
    - **Step 2: Fraud Analyst Review**
        - Assigned to: `Fraud@MultiSureCorp`
        - Collect: Risk Rating (`.FraudScore`) and Notes

□ These will run **in parallel** and both must be completed to move ahead.

You can enforce SLA here.

Add SLA to each review step:

- Go to step → Settings → SLA → Create new SLA rule (e.g., 1 day)
- Route to `ClaimsManager` if deadline missed

□ STEP 6: Stage 4 – Final Settlement

Step: Consolidate & Approve

1. Add step: `Decision - Final Approval`

2. Use **Approval shape**.
3. Add field:
    - `Final Decision` (Dropdown): Approve / Reject
    - `Remarks`

Step: Confirm Result

1. Add final step: `Confirmation`
2. Use **Show confirmation screen with Case ID and Summary**

□ STEP 7: Save & Run

1. Save the case type.
2. Go to Home → `+ Create → ManageClaim`
3. Try uploading evidence, trigger mock REST, watch parallel approval happen.

□ WHAT YOU MASTERED

Concept	Where Used
File Upload	Step: Upload Supporting Documents
REST Integration (Mock)	Policy check via Data Page → Connector Simulation
Data Page Usage	`D_ValidatePolicy` for dynamic values
Parallel Processing	Approver + Fraud Analyst in parallel
SLA with Routing	Deadline enforcement in review steps
Conditional Flow	Approval after dual parallel reviews

□ `ManageClaim` Case Type Complete.

□ PHASE 3: MyPolicies Dashboard (Constellation View + Data Page)

□ PHASE 4: Predictive Risk Scoring (AI + Prediction Studio + Case-level Insights)

You'll get:

- A live portal view that shows each user's active policies using **Data Page + Embedded Table**
- A risk score generated using **Prediction Studio**, injected into the case to guide decisions

---

□ PHASE 3: MyPolicies Dashboard – Show Logged-in User's Policies

□ GOAL:

Display a list of policies belonging to the logged-in user (contextually), using:

- A **Data Page** that filters policies by current user
- An **Embedded Table View** inside a **Constellation Portal**
- Optional links to `CreateClaim` or `Renew`

□ STEP-BY-STEP

□ STEP 1: Create Report Definition

1. In **Dev Studio**, go to `Records → Reports → Report Definition`.
2. Create:
   - Name: `MyPoliciesReport`
   - Applies To: `MultiSure360-Work-Policy`
3. Add filters:
   - `.CustomerID = Param.CustomerID`
4. Display columns:
   - Policy Number
   - Policy Type
   - Status
   - Valid Until
   - Estimated Premium

□ *We'll use this in a Data Page to feed the view.*

□ STEP 2: Create a Data Page

1. Name: `D_MyPolicies`
2. Object Type: `MultiSure360-Work-Policy`
3. Structure: **List**
4. Scope: **Requestor**
5. Data Source: Report Definition → `MyPoliciesReport`
6. Parameter: `CustomerID`
7. Key settings:
   - Refresh Strategy: On user login or manual trigger

- Filter using: `Param.CustomerID` → set from `.pyUserIdentifier` or operator ID mapping

□ STEP 3: Create Embedded Table View in App Studio

1. Switch to **App Studio → Views**
2. Click **+ Add View**
3. Name: `MyPoliciesTable`
4. View Type: **Table**
5. Source: `D_MyPolicies[CustomerID:.pyUserIdentifier]`
6. Columns:
   - `Policy Number`
   - `Policy Type`
   - `Status`
   - `Estimated Premium`
   - Add button column: `Create Claim` → Launch `ManageClaim` with policy number pre-filled
7. Optional: Use **row click action** → Open existing case in read-only mode

---

□ STEP 4: Add to Portal Home Page

1. Go to **Channels → Web** → Select your portal (`CustomerPortal`)
2. Add widget → Embed `MyPoliciesTable` view
3. Title: **"My Policies"**

□ WHAT YOU MASTERED:

Feature	Applied
Param-driven D_Page	Filter policies per user
Embedded Table View	Show list with action buttons
Portal Composition	Drag-drop Constellation widget
Contextual Access	Dynamic filtering using `.pyUserIdentifier`

□ PHASE 4: Predictive Risk Scoring in ManageClaim

□ GOAL:

Use **Prediction Studio** to score risk of a claim and guide approvals.

□ STEP-BY-STEP

□ STEP 1: Launch Prediction Studio

1. In Dev Studio → App Explorer → Go to **Prediction Studio**.
2. Click **+ Prediction**

Prediction Name: **`ClaimRiskPrediction`**

- Subject Type: `Case`
- Case Type: `ManageClaim`
- Field to Predict: **`.ClaimRiskScore`**
- Prediction Type: **Scorecard** (static logic) or **Model** (ML later)

□ STEP 2: Configure Scorecard (for now)

We'll create a simple rule-based scorecard:

Condition	Score
ClaimType = "Motor"	20
IncidentDate < Today - 90 days	30
Description contains "accident"	50
File Uploaded	10

1. In Prediction Studio, choose **Scorecard**.
2. Configure each condition.
3. Output Field: `.ClaimRiskScore`

→ The final score is between 0-100.

□ STEP 3: Add Model to Case Type

1. Go to `ManageClaim` Case Type → Open `Check Policy` step.
2. Add pre-processing action:
   - Call Prediction: `ClaimRiskPrediction`
   - Input Mapping: `.ClaimType`, `.IncidentDate`, `.ClaimDescription`, `.EvidenceUpload`
   - Output Mapping: `.ClaimRiskScore`

□ STEP 4: Display Risk Score

1. Add **read-only field** in Summary screen:
    - Label: **“Predicted Risk Score”**
    - Value: `.ClaimRiskScore`
2. (Optional) Add logic:
    - If `.ClaimRiskScore > 60` → show warning banner
    - Route high-risk claims to `SeniorFraud@MultiSureCorp`

□ WHAT YOU MASTERED:

Concept	Applied Where
Prediction Studio	Defined `ClaimRiskPrediction` for scoring
Scorecard Model	Rule-based scoring logic
Risk Injection in Flow	Used as pre-processing in case stage
Display + Routing	Dynamic behavior based on AI output

□ DONE □

You now have:

- □ `MyPolicies` dashboard: A real-time, per-user portal widget
- □ `ClaimRiskPrediction`: AI logic applied mid-case to steer outcomes

□ Next Optional Add-ons (Choose Any):

1. **Power BI Integration** – Push data via REST or expose Pega RD to Power Query
2. **Kafka Messaging** – Publish claim events to Kafka topic (for external analytics)
3. **BIX Extraction** – Nightly export of Policy + Claim data for audit
4. **Pulse & Notifications** – Add alerts when claim reaches critical stages
5. **Custom DX Component** – Embed Angular/React widget inside Constellation view
6. **Job Scheduler + Queue Processor** – For claim aging, auto-reminders

□ PHASE 5: Power BI Integration + BIX Data Export

You’ll learn:

- □ How to **expose Pega Report Definition via REST API** so Power BI can query it using `OData` or `Web API`
- □ How to **configure BIX** to export claim + policy data as XML/CSV
- □ How to **secure both** integrations

□ PART A: Power BI ↔ Pega Integration via REST API

□ GOAL:

Expose `Report Definition` data as a REST API that Power BI can fetch via **Power Query Web Connector**.

□ STEP 1: Create a Report Definition

Already created: □ `MyPoliciesReport`
We'll now expose it as an API.

□ STEP 2: Create REST Service (Service REST)

1. **Dev Studio → Records → Integration-Services → Service REST**
2. Create:
   - Name: `GetPoliciesService`
   - Class: `MultiSure360-Work-Policy`
   - Service Package: `BIDataService`
   - Endpoint: `/policies`
3. Resource Method: `GET`
   - Map to Activity: `FetchPoliciesForBI`

□ STEP 3: Create Activity: **`FetchPoliciesForBI`**

□ Activity Steps

1. **Step 1**: Use `Obj-Open-By-Handle` or call **`MyPoliciesReport`** using `pxRetrieveReportData`
2. **Step 2**: Loop over results, build JSON manually using `Property-Set` + `Append to Page List`
3. **Step 3**: Use `Property-Set` on `pyResponseData` to return output

□ Sample JSON Output

```
json
CopyEdit
```

```
[
  {
    "PolicyNumber": "POL1234",
    "CustomerName": "John Doe",
    "Premium": 4200,
    "ValidUntil": "2026-12-31"
  },
  {
    "PolicyNumber": "POL5678",
    "CustomerName": "Priya Sharma",
    "Premium": 3050,
    "ValidUntil": "2025-11-20"
  }
]
```

□ STEP 4: Configure Service Package Security

1. Go to **Records → Integration-Services → Service Package →** **`BIDataService`**
2. Set:
   - Authentication Type: **Basic**
   - Access Group: `MultiSure360:BIUser`
   - Check `Requires Authentication`

□ STEP 5: Create BI Access Group

1. Create new **Access Group**: `MultiSure360:BIUser`
2. Add:
   - Application: `MultiSure360`
   - Roles: `PegaRULES:SysAdm4`, or custom role with Report Run, View, API access
3. Create Operator ID: `bi.integration@multisure`
4. Set password and uncheck "Force password change"
5. Add to Access Group: `MultiSure360:BIUser`

□ STEP 6: Consume in Power BI

In Power BI:

1. Go to **Get Data → Web**
2. Enter URL:

   ```
   bash
   CopyEdit
   https://<your-pega-
   host>:<port>/prweb/PRRestService/BIDataService/policies
   ```

3. Choose **Basic Auth**, use the `bi.integration@multisure` user.

4. Load data → Power BI Table → Done!

 POWER BI LIVE QUERY DONE.

□ PART B: BIX EXPORT – Nightly XML/CSV Extract for Claim/Policy

---

□ GOAL:

Extract `ManageClaim` and `Policy` data every night using BIX in CSV or XML format.

□ STEP 1: Enable BIX

1. Open `prconfig.xml`, add or ensure:

```
xml
CopyEdit
<env name="indexing/BIX/enabled" value="true"/>
```

2. Restart the JVM.

□ STEP 2: Create BIX Extract Rule

1. Dev Studio → Records → SysAdmin → **Extract**
2. Create:
   - Rule Name: `ClaimDataExtract`
   - Applies To: `MultiSure360-Work-ManageClaim`
   - File Type: `CSV`
   - Include all relevant properties (Policy No, Type, Amount, Risk Score, Status)
   - File Name: `Claims_%yymmdd%_extract.csv`
   - Location: `/bix/export/claims/`
3. Repeat for `PolicyDataExtract` on `MultiSure360-Work-Policy`

□ STEP 3: Schedule via Job Scheduler

1. Dev Studio → Records → SysAdmin → **Job Scheduler**
2. Create Job: `RunClaimBIXJob`

- Class: `MultiSure360-Work`
- Rule to Execute: Activity → `pxExtractDataWithArgs` (OOTB)
- Parameters:

```
diff
CopyEdit
-i ClaimDataExtract
-o /bix/export/claims/
-f CSV
```

3. Set recurrence: **Daily at 1:00 AM**

□ Security Best Practices

- Limit BIX access to secure folder path
- Encrypt output files if required (Pega 8.5+ supports encrypted CSV/XML)
- Rotate exported files periodically

□ VERIFICATION:

- Open `/bix/export/claims/` → You should see:

```
CopyEdit
Claims_250609_extract.csv
Policies_250609_extract.csv
```

- Import into Excel or upload to Power BI/BigQuery/Snowflake

□ WHAT YOU MASTERED:

Integration Type	Description
Power BI via REST	Realtime consumption using REST + Power Query
Service REST	Built secure endpoint backed by Activity
Role-based API Access	Limited Access Group & Operator for BI Integration
BIX CSV Export	Batch extraction with field selection + job scheduling
Data Security Practices	REST auth, folder protection, file naming + rotation

---

□ DONE

Both **Power BI live dashboards** and **BIX nightly extracts** are now integrated into your MultiSure360 project.

□ NEXT OPTIONS (Optional/Advanced):

1. **Kafka Integration** – Real-time claim events pushed to Kafka
2. **Pulse & Notification Rules** – Notify adjusters/underwriters on SLA breach
3. **Job Scheduler + QP** – Automatically escalate aging claims
4. **Custom DX Component** – React/Angular component embedded in Constellation view
5. **Localization/Accessibility/Mobile offline support**

□ PHASE 6: Kafka Messaging + Pulse Notifications

□ PART A: Kafka Integration — Publish Claim Events

□ GOAL:

Send real-time claim events (like claim created or updated) from Pega to an Apache Kafka topic so external systems can consume them.

□ STEP 1: Setup Kafka Connector in Pega

1. **Install Kafka Connector Add-on**
   - In Dev Studio → **Records → Integration-Connectors → Connector**
   - Create Connector: `KafkaConnector`
2. Configure the connector with Kafka broker details:
   - Broker List (host:port)
   - Topic Name: `MultiSure360Claims`
   - Serialization: JSON

□ STEP 2: Create Kafka Producer Activity

1. Create an Activity `PublishClaimToKafka`
2. Steps:
   - Use `Call Connector` step → use `KafkaConnector`

IVKPEGA@GMAIL.COM

- Build JSON payload with claim details: `.ClaimID`, `.PolicyNumber`, `.ClaimStatus`, `.ClaimRiskScore`
- Send message to Kafka topic

□ STEP 3: Invoke Activity on Claim Events

1. Go to **ManageClaim** Case Type
2. Add a **Post-Processing step** after `CreateClaim` or `UpdateClaim` steps
3. Call `PublishClaimToKafka` activity to send event

□ STEP 4: Test Kafka Event Publishing

- Use Kafka CLI or Kafka UI tool to monitor the `MultiSure360Claims` topic
- Create/update a claim → Confirm JSON event received

□ What you mastered:

- Kafka connector setup
- JSON event serialization
- Event-driven integration for real-time data flow

□ PART B: Pulse Notifications — Alerts for SLA Breach or High-Risk Claims

---

□ GOAL:

Use Pega Pulse to send alerts/notifications to users (adjusters, underwriters) on claim status changes or SLA breach.

□ STEP 1: Enable Pulse in Your Application

1. Dev Studio → Records → Integration → **Pulse Configuration**
2. Enable and configure Pulse channels (email, SMS, Slack, Microsoft Teams, etc.)

□ STEP 2: Define Pulse Notifications

1. Create Pulse Notification rule: `ClaimHighRiskAlert`
2. Message:

```
css
CopyEdit
ALERT: Claim {{ .ClaimID }} for policy {{ .PolicyNumber }} has high
risk score {{ .ClaimRiskScore }}.
Immediate review required.
```

□ STEP 3: Create SLA Rule on Claim Case

1. Define SLA: `ClaimReviewSLA`
2. Set target resolution time (e.g., 48 hours)
3. Attach **On Delay** or **On Deadline** action: Send Pulse Notification `ClaimHighRiskAlert`

□ STEP 4: Attach SLA to ManageClaim Flow

- Attach the SLA to the claim review stage
- When SLA deadline hits, Pulse notification triggers

□ STEP 5: Test Notifications

- Create claims with high risk → verify Pulse alert is sent to assigned users
- Wait for SLA deadline → confirm notification triggers

□ What you mastered:

- Pulse notification configuration
- SLA creation and binding to case
- Automated alerting for timely action

□ DONE!

Your MultiSure360 system now supports:

Feature	Description
Kafka Messaging	Real-time claim event streaming

Feature	Description
Pulse Notifications	Automated alerts for risk & SLA breach

□ Next Options (Advanced):

1. Job Scheduler + Queue Processor for claim aging
2. Custom React DX Component inside Constellation view
3. Localization and Accessibility best practices
4. Mobile offline claim entry and sync

□ PHASE 7: Job Scheduler + Queue Processor for Claim Aging and Escalation

---

□ GOAL:

Automate claim aging management by scheduling periodic checks to escalate claims pending beyond SLA or specific durations.

□ STEP 1: Create a Job Scheduler Rule

1. Dev Studio → Records → SysAdmin → **Job Scheduler**
2. Create new Job Scheduler:
   - Name: `ClaimAgingScheduler`
   - Class: `MultiSure360-Work-ManageClaim`
   - Recurrence: Daily at 2:00 AM
   - Run Mode: Background

□ STEP 2: Create a Queue Processor Rule

1. Dev Studio → Records → SysAdmin → **Queue Processor**
2. Create:
   - Name: `ClaimEscalationProcessor`
   - Class: `MultiSure360-Work-ManageClaim`
   - Queue: `ClaimEscalationQueue`
   - Max Threads: 5 (adjust based on load)

□ STEP 3: Create an Activity to Enqueue Claims

1. Create Activity: `EnqueueAgingClaims`
2. Steps:
    - Use a Report Definition to find claims older than SLA threshold or not updated for X days
    - Loop over results and add each Claim Work Object to Queue Processor's Queue (`ClaimEscalationQueue`) using `Queue-Add`

□ STEP 4: Create an Activity to Process Queue Items

1. Create Activity: `ProcessClaimEscalation`
2. Logic:
    - Retrieve the Claim case (work object)
    - Check if escalation criteria met
    - Change status or assign to escalation workbasket
    - Optionally send Pulse notification for escalation

□ STEP 5: Configure Job Scheduler to Call Enqueue Activity

- Job Scheduler Rule `ClaimAgingScheduler` → calls `EnqueueAgingClaims`

□ STEP 6: Test Full Flow

- Create old claims > SLA
- Run Scheduler manually or wait for trigger
- Queue Processor picks claims from queue and escalates

□ What you mastered:

Concept	Explanation
Job Scheduler	Periodic execution to trigger batch processes
Queue Processor	Asynchronous multi-threaded claim escalation
Claim Aging Logic	Criteria-based claim selection & escalation
Pulse Notifications	Alerting during escalation

□ PHASE 8: Custom DX Component + Localization + Mobile Offline

---

□ PART A: Custom DX Component in Constellation (React/Angular)

□ GOAL:

Embed a custom React or Angular component inside a Constellation dashboard or case view for richer, interactive UI.

□ STEP 1: Setup your Dev Environment

1. Install **Node.js** & **npm**
2. Use React or Angular CLI:
   - React: `npx create-react-app multisure360-dx`
   - Angular: `ng new multisure360-dx`

□ STEP 2: Build Your Component

- Build a component for policy rating, claim visualization, or risk dashboard
- Make sure it can accept **props** or **inputs** (e.g., policy ID, claim ID) from Pega

---

□ STEP 3: Package & Bundle Your Component

- For React, use `npm run build` → produces static JS/CSS files
- For Angular, use `ng build --prod` → produces `dist/` files

□ STEP 4: Upload Static Files to Pega

- Use **File Resource** rule to upload JS/CSS bundles
- Or host externally (S3, CDN) with CORS enabled

□ STEP 5: Create a Custom DX Component Rule

1. Dev Studio → Records → User Interface → **Custom DX Component**
2. Name: `ClaimRiskChart` or `PolicyRatingWidget`
3. Define the `ComponentID` and mapping for input parameters
4. Provide the URL or path to your JS bundle

□ STEP 6: Use the DX Component in a Section

- Open your case type section (e.g., ManageClaim overview)
- Drag **Custom DX Component** shape
- Select your custom component
- Map inputs (property values) from clipboard

□ STEP 7: Test & Debug

- Launch case in Constellation
- Check browser dev tools for errors
- Verify the component renders and interacts with Pega data

□ What you mastered:

Concept	Explanation
DX Component	Embedding React/Angular in Pega UI
Data Binding	Passing clipboard props to component
Bundling & Hosting	Static resource management

□ PART B: Localization + Accessibility Best Practices

□ GOAL:

Make the app accessible and multilingual for global users.

□ STEP 1: Setup Localization

1. Dev Studio → Records → Application → **Locale**
2. Add languages you want to support (e.g., English, Hindi, Spanish)

3. Use **Translation** rules to provide translations for labels/messages
4. Use **Resource Bundle** to manage static text

□ STEP 2: Design for Accessibility (a11y)

- Use Pega's **Section Accessibility Settings** to set ARIA roles, tab order, and alt texts
- Ensure color contrast ratios meet WCAG guidelines
- Use keyboard navigation for all interactive elements
- Test with screen readers (NVDA, JAWS)

□ STEP 3: Enable User Language Preference

- Configure user settings to select preferred language
- Use `pxSetLocale` utility to switch locale dynamically

□ STEP 4: Testing

- Verify UI switches languages correctly
- Test accessibility using browser tools and assistive devices

---

□ What you mastered:

Concept	Explanation
Multilingual UI	Locale and translation rules
Accessibility	ARIA roles, keyboard nav, contrast
User Preferences	Dynamic language switching

□ PART C: Mobile Offline Claim Entry & Sync

□ GOAL:

Enable users to create and update claims offline on mobile and sync data when connected.

□ STEP 1: Enable Mobile Offline Support

1. Dev Studio → Records → Application → **Mobile Offline Profile**
2. Configure offline settings for **ManageClaim** case class
3. Define properties and data pages to be cached offline

□ STEP 2: Design Mobile Sections for Offline Use

- Keep UI minimal for offline (avoid heavy layouts)
- Use **Offline Forms** with required validations

□ STEP 3: Configure Sync Profiles

1. Define Sync Profile with endpoints to sync data
2. Setup Conflict Resolution rules for merges

□ STEP 4: Deploy Mobile App

- Build mobile app via Pega Mobile Client or custom app with SDK
- Test offline mode by disabling network, create/update claims
- Reconnect and trigger sync → verify data consistency

□ STEP 5: Handle Sync Conflicts

- Configure rules to detect and resolve conflicting changes (Last write wins, manual merge)

□ What you mastered:

Concept	Explanation
Offline Profile	Defining offline cache & data sync
Mobile Forms	Designing mobile-friendly offline UIs
Sync & Conflict	Data sync strategies and conflict resolution

□ SUMMARY

Topic	Description
DX Component	React/Angular integration into Constellation UI

Topic	Description
Localization	Multilingual and accessibility support
Mobile Offline	Offline mobile claim entry and sync

1️⃣ Custom React DX Component for Constellation

Step 1: Setup React Project

```
bash
CopyEdit
npx create-react-app multisure360-dx
cd multisure360-dx
```

Step 2: Create a Component

Create `src/ClaimRiskChart.js`:

```
jsx
CopyEdit
import React from 'react';

const ClaimRiskChart = ({ claimId, riskScore }) => {
  return (
    <div style={{ border: '1px solid #ccc', padding: 10 }}>
      <h3>Claim Risk Chart</h3>
      <p><b>Claim ID:</b> {claimId}</p>
      <p><b>Risk Score:</b> {riskScore}</p>
      {/* Placeholder for chart - can integrate chart libraries */}
    </div>
  );
};

export default ClaimRiskChart;
```

Step 3: Modify `src/App.js` to accept props

```
jsx
CopyEdit
import React from 'react';
import ClaimRiskChart from './ClaimRiskChart';

function App(props) {
  // Pega will pass input props here as JSON string in window.pegaProps
  const pegaProps = window.pegaProps ? JSON.parse(window.pegaProps) : {};
  return (
```

JVKPEGA@GMAIL.COM

```
    <div className="App">
      <ClaimRiskChart
        claimId={pegaProps.claimId}
        riskScore={pegaProps.riskScore}
      />
    </div>
  );
}

export default App;
```

Step 4: Build for Production

```
bash
CopyEdit
npm run build
```

Step 5: Upload Assets to Pega

- Upload the content of `build/static/js/main.*.js` and `build/static/css/main.*.css` as **File Resource** rules or host externally with CORS enabled.

Step 6: Create Custom DX Component Rule

- In Dev Studio → **Records → User Interface → Custom DX Component**
- Name: `ClaimRiskChart`
- Component ID: `ClaimRiskChart`
- Enter URLs for JS and CSS bundles
- Define input parameters: `claimId`, `riskScore` mapped to clipboard properties

---

Step 7: Use DX Component in Section

- Open your case view section (e.g., ManageClaim Overview)
- Add **Custom DX Component** shape
- Select `ClaimRiskChart` component
- Map inputs (e.g., `.ClaimID`, `.RiskScore`)

---

2 Localization + Accessibility

Localization Steps

- Add locales: Dev Studio → Records → Application → Locale
  - English (`en`), Hindi (`hi`), Spanish (`es`), etc.
- Create **Translation** rules for each label/message
- Use **Resource Bundle** rules for common text

Dynamic Locale Switching

In Activity or Data Transform, call `pxSetLocale` with user's language code:

```
plaintext
CopyEdit
Obj-Open-By-Handle <UserObject>
Property-Set pxLocale = "hi-IN"
Commit
```

Accessibility Best Practices

- Use Pega's Accessibility tab in Sections to set ARIA roles
- Add **Alt text** on images and buttons
- Ensure tab order and keyboard navigation is logical
- Check color contrast (use browser plugins like Axe)
- Test with NVDA screen reader on Windows or VoiceOver on Mac

---

3 Mobile Offline Support with Sync

Step 1: Create Mobile Offline Profile

- Dev Studio → Records → Application → Mobile Offline Profile
- Name: `MultiSure360Offline`
- Add **ManageClaim** class
- Select properties and data pages needed offline

Step 2: Create Sync Profile

- Dev Studio → Records → Application → Sync Profile
- Name: `MultiSure360Sync`
- Configure endpoints and sync intervals

Step 3: Offline Form Design

- Keep UI lightweight and minimal for mobile
- Use **Offline Forms** in your Mobile UI sections
- Validate input fields client-side

Step 4: Build & Deploy Mobile App

- Use **Pega Mobile Client** or create a custom app with Mobile SDK
- Test by disabling network → create/update claim → reconnect and sync

---

Step 5: Conflict Resolution Rules

- Use **Conflict Resolution** rules to decide on merges:
  - Last write wins
  - Manual review for conflicts
  - Custom merge logic as needed

IVKPEGA@GMAIL.COM

□ Summary & Samples

Topic	Sample / Rule Type	Key Points
React DX Component	Custom DX Component + React code	Props mapping, JS/CSS bundles
Localization	Locale + Translation rules	Multi-language support, pxSetLocale
Accessibility	Accessibility tab settings	ARIA roles, tab order, alt text
Mobile Offline Support	Mobile Offline Profile + Sync Profile	Offline caching & syncing, conflict resolution

---

1□□ Custom React DX Component — Full Setup

React Code: `ClaimRiskChart.js`

```
jsx
```

```
CopyEdit
import React from 'react';

const ClaimRiskChart = ({ claimId, riskScore }) => {
  return (
    <div style={{ border: '1px solid #ccc', padding: 10 }}>
      <h3>Claim Risk Chart</h3>
      <p><b>Claim ID:</b> {claimId}</p>
      <p><b>Risk Score:</b> {riskScore}</p>
    </div>
  );
};

export default ClaimRiskChart;
```

React App Entry: `App.js`

```
jsx
CopyEdit
import React from 'react';
import ClaimRiskChart from './ClaimRiskChart';

function App() {
  const pegaProps = window.pegaProps ? JSON.parse(window.pegaProps) : {};
  return (
    <div>
      <ClaimRiskChart
        claimId={pegaProps.claimId}
        riskScore={pegaProps.riskScore}
      />
    </div>
  );
}

export default App;
```

Build and Upload Bundles

- Run `npm run build`
- Upload `build/static/js/main.[hash].js` and `build/static/css/main.[hash].css` as **File Resource** rules
- Or host externally with proper CORS

Pega Custom DX Component Rule XML Snippet

```
xml
CopyEdit
<CustomDXComponent name="ClaimRiskChart" pyID="ClaimRiskChart">
  <ComponentID>ClaimRiskChart</ComponentID>
  <JavaScriptURL>~File:MultiSure360/main.[hash].js</JavaScriptURL>
  <CSSURL>~File:MultiSure360/main.[hash].css</CSSURL>
  <Parameters>
    <Parameter name="claimId" clipboardProperty="ClaimID" />
```

```
    <Parameter name="riskScore" clipboardProperty="RiskScore" />
  </Parameters>
</CustomDXComponent>
```

2️⃣ Localization + Resource Bundle Example

Sample Translation Rule (English)

Key	Text
ClaimLabel	Claim
RiskScoreLabel	Risk Score
SubmitButton	Submit

Sample Translation Rule (Hindi)

Key	Text
ClaimLabel	दावा
RiskScoreLabel	जोखिम स्कोर
SubmitButton	जमा करें

Resource Bundle Snippet (common.properties)

```
ini
CopyEdit
welcome.message=Welcome to MultiSure360!
alert.highrisk=Alert: High risk claim detected.
button.submit=Submit
```

Dynamic Locale Switch Sample (Activity Step)

```
plaintext
CopyEdit
Property-Set
  .pxLocale = "hi-IN"
Commit
```

---

3 Mobile Offline Profile + Sync + Conflict Resolution

Mobile Offline Profile Config

Setting	Value
Name	MultiSure360Offline
Classes Included	MultiSure360-Work-ManageClaim
Properties Cached	ClaimID, PolicyNumber, ClaimStatus, RiskScore
Data Pages Cached	D_PolicyDetails, D_UserSettings

Sync Profile Config

Setting	Value
Name	MultiSure360Sync
Sync Endpoint	`/prweb/api/v1/mobile/sync`
Sync Interval	Every 30 minutes
Conflict Resolution	Last write wins (default)

Conflict Resolution Rule Example

- Rule type: **Conflict Resolution**
- Strategy:
  - For `RiskScore`, choose max value
  - For `ClaimStatus`, prefer “Escalated” over others

Offline Form Tips

- Keep forms simple, use dropdowns for status
- Validate required fields client-side
- Show sync status UI on mobile app

Summary

Area	Key Points

Area	Key Points
DX Component	React code + upload bundles + XML
Localization	Translation rules + resource bundle + locale switching
Mobile Offline	Offline profile + sync + conflict resolution

1️⃣ Full React DX Component Project + Pega Rule XML Export

React Project (zipped) structure

```
pgsql
CopyEdit
multisure360-dx/
├── public/
│   └── index.html
├── src/
│   ├── App.js
│   └── ClaimRiskChart.js
├── package.json
└── README.md
```

Key files:

- src/ClaimRiskChart.js

```
jsx
CopyEdit
import React from 'react';

const ClaimRiskChart = ({ claimId, riskScore }) => {
  return (
    <div style={{ border: '1px solid #ccc', padding: 10 }}>
      <h3>Claim Risk Chart</h3>
      <p><b>Claim ID:</b> {claimId}</p>
      <p><b>Risk Score:</b> {riskScore}</p>
    </div>
  );
};

export default ClaimRiskChart;
```

- src/App.js

```
jsx
CopyEdit
import React from 'react';
import ClaimRiskChart from './ClaimRiskChart';
```

IVKPEGA@GMAIL.COM

```
function App() {
  const pegaProps = window.pegaProps ? JSON.parse(window.pegaProps) : {};
  return (
    <div>
      <ClaimRiskChart
        claimId={pegaProps.claimId}
        riskScore={pegaProps.riskScore}
      />
    </div>
  );
}

export default App;
```

Build Instructions:

```
bash
CopyEdit
npm install
npm run build
```

Upload the generated `build/static/js/main.[hash].js` and `build/static/css/main.[hash].css` as **File Resources** in Pega.

Pega Custom DX Component XML (export snippet)

```
xml
CopyEdit
<CustomDXComponent name="ClaimRiskChart" pyID="ClaimRiskChart">
  <ComponentID>ClaimRiskChart</ComponentID>
  <JavaScriptURL>~File:MultiSure360/main.[hash].js</JavaScriptURL>
  <CSSURL>~File:MultiSure360/main.[hash].css</CSSURL>
  <Parameters>
    <Parameter name="claimId" clipboardProperty="ClaimID" />
    <Parameter name="riskScore" clipboardProperty="RiskScore" />
  </Parameters>
</CustomDXComponent>
```

---

2 Localization Package

Translation Rule Example

Key	English Text	Hindi Text
ClaimLabel	Claim	दावा
RiskScoreLabel	Risk Score	जोखिम स्कोर
SubmitButton	Submit	जमा करें

Resource Bundle `common.properties`

```
ini
CopyEdit
welcome.message=Welcome to MultiSure360!
alert.highrisk=Alert: High risk claim detected.
button.submit=Submit
```

Activity Step for Locale Switch

```
plaintext
CopyEdit
Property-Set
  .pxLocale = "hi-IN"
Commit
```

---

3 Mobile Offline & Sync Package

Mobile Offline Profile

Name	MultiSure360Offline
Classes Included	MultiSure360-Work-ManageClaim
Properties Cached	ClaimID, PolicyNumber, ClaimStatus, RiskScore
Data Pages Cached	D_PolicyDetails, D_UserSettings

Sync Profile

Name	MultiSure360Sync
Sync Endpoint	`/prweb/api/v1/mobile/sync`
Sync Interval	30 minutes
Conflict Resolution	Last Write Wins (default)

Conflict Resolution Example

- For `RiskScore`, choose max value
- For `ClaimStatus`, prefer “Escalated” state over others

Offline Form Best Practices

- Use dropdowns for status fields
- Validate inputs client-side
- Show sync status on the mobile UI

IVKPEGA@GMAIL.COM