□ Fix Add Beneficiary to Not Send SMS When Insured Person Is Still Alive

- Assigned to: Buks
- Description: Modify the beneficiary addition process to prevent SMS notifications to beneficiaries when the insured person is alive.
- Plan:
    - Update the beneficiary addition logic (Existing Task 4: beneficiary backend) to check the insured person's status (e.g., alive/deceased flag in the database, linked to New Task 1: insured person integration).
    - Modify the notification workflow (Existing Task 8: insured person notifications, New Task 1: insured person addition) to skip SMS for beneficiaries if the insured person is alive.
    - Add a condition in the backend API to verify the insured person's status before triggering notifications.
    - Test scenarios: adding a beneficiary when the insured person is alive (no SMS) vs. deceased (SMS sent).
- Priority: Medium
- Dependencies: Existing Task 4 (beneficiary backend), New Task 1 (insured person integration), Existing Task 8 (notification logic)

□ Implement Backend for When an Insured Person Is Declared Dead for Beneficiaries

- Assigned to: Apfeh
- Description: Create backend logic to handle processes when an insured person is declared dead, enabling beneficiary-related actions (e.g., claim initiation).
- Plan:
    - Add a status field (e.g., alive/deceased) to the insured person schema (New Task 1: insured person integration).
    - Create an API endpoint to update the insured person's status to deceased, with validation (e.g., authorized admin or Home Affairs data from Existing Task 7).
    - Trigger beneficiary notifications (e.g., SMS/email) upon status change, linking to claim processes (New Task 2: policy/claim review).
    - Log status changes (integrate with Additional Task 1: audit logging, if assigned).

    - Test with scenarios: status update, notification triggers, and claim initiation for multiple beneficiaries.
- Priority: High
- Dependencies: Existing Task 4 (beneficiary backend), New Task 1 (insured person integration), Existing Task 7 (Home Affairs integration)

□ Implement Frontend for Login (ID and Email + OTP), Password Update, and Beneficiary Actions When Insured Person Is Dead

- Assigned to: Letho
- Description: Develop frontend for secure login using ID and email with OTP, password update functionality, and beneficiary actions when the insured person is deceased.
- Plan:
    - Login: Create a login form accepting ID and email, followed by OTP input (integrate with backend OTP generation, e.g., via email/SMS).
    - Password Update: Build a frontend page for password updates with validation (e.g., strong password rules, confirm new password).
    - Beneficiary Actions: Develop a beneficiary dashboard to view/initiate claims when the insured person is deceased (link to Task 2 above).
    - Ensure consistent styling with Existing Task 5 (page styling) and responsive design.
    - Test login flow, password updates, and beneficiary actions across browsers/devices.
- Priority: High
- Dependencies: Existing Task 1 (policyholder login), Existing Task 4 (beneficiary backend), Task 2 (deceased insured backend)

□ Implement Error Pages

- Assigned to: Alfredo
- Description: Create user-friendly error pages (e.g., 404, 500, unauthorized access) to handle system errors gracefully.
- Plan:

    - Design error pages (e.g., 404: Page Not Found, 500: Server Error, 403: Unauthorized) with clear messages and navigation options (e.g., “Back to Home” or “Try Again”).
    - Integrate error pages into the frontend routing (e.g., React Router, Angular, or server-side redirects).
    - Update backend to return appropriate HTTP status codes and trigger error pages for exceptions (e.g., invalid API requests, database errors).
    - Ensure styling aligns with Existing Task 5 (page styling) and is responsive.
    - Test error scenarios: invalid URLs, unauthorized access, server downtime.
- Priority: Medium
- Dependencies: Existing Task 5 (page styling), Existing Task 1 (policyholder system for login errors), Additional Task 3 (API security, if assigned, for handling unauthorized access)

▯ Implement Policyholder Profile Update Functionality

- Assigned to: Boitshepo
- Description: Allow policyholders to update their profile details (e.g., contact information, address) via the policyholder dashboard.
- Plan:
    - Extend the policyholder dashboard (already implemented by Boitshepo) with a profile update form (fields: email, phone, address, etc.).
    - Build backend API endpoints to update policyholder data in the database (Existing Task 1: policyholder system), with validation (e.g., valid email format, phone number).
    - Log updates for auditing (integrate with Additional Task 1: audit logging, if active).
    - Ensure secure access (only logged-in policyholders can update their own data, using Existing Task 1: login system).
    - Test updates with sample data and verify UI consistency with Existing Task 5 (page styling).
- Priority: Medium
- Dependencies: Existing Task 1 (policyholder login/registration), Existing Task 5 (styling), Boitshepo’s completed Task 1 (policyholder dashboard)

- Rationale: Policyholders need to update their details to keep records accurate, enhancing the dashboard’s functionality.

□ Add Claim Status Tracking Notifications for Policyholders and Beneficiaries

- Assigned to: Maite
- Description: Send automated notifications (email/SMS) to policyholders and beneficiaries when claim statuses change (e.g., submitted, under review, approved).
- Plan:
    - Extend the claim submission backend (Maite’s completed Task 3) to trigger notifications on status updates (integrate with Existing Task 8: notifications, New Task 1: insured person notifications).
    - Create backend logic to detect claim status changes (e.g., via database triggers or API updates in New Task 2: claim review).
    - Use an email/SMS service (e.g., SendGrid/Twilio) to send status updates with details (e.g., “Your claim #123 is under review”).
    - Allow users to view notification history in the policyholder/beneficiary dashboard (Boitshepo’s Task 1, New Task 3: beneficiary frontend).
    - Test notifications for various claim status transitions and delivery accuracy.
- Priority: High
- Dependencies: Maite’s completed Task 3 (claim submission), New Task 2 (claim review), Existing Task 8 (notifications), New Task 1 (insured person notifications), New Task 3 (beneficiary frontend)
- Rationale: Real-time claim status updates improve user experience and transparency, building on the claim submission system.

□ Implement Admin Approval Workflow for Insured Person Status Updates

- Assigned to: Nyiko
- Description: Create an admin workflow to review and approve insured person status updates (e.g., deceased status) before they are finalized.
- Plan:
    - Build a backend API to queue insured person status updates (New Task 2: deceased insured backend) for admin review.

  - Add a frontend section in the admin dashboard (Existing Task 2, Existing Task 9) to display pending status updates with approve/reject options.
  - Integrate with Home Affairs API (Existing Task 7, Nyiko's completed Task 4: document verification) for validation (e.g., death certificate).
  - Notify policyholders/beneficiaries of approval/rejection (link to Maite's Task 2 above).
  - Test the workflow with sample status updates and admin actions.
- Priority: High
- Dependencies: New Task 2 (deceased insured backend), Existing Task 7 (Home Affairs integration), Nyiko's completed Task 4 (document verification), Existing Task 2 (admin dashboard), Existing Task 9 (admin functionality)
- Rationale: Admin oversight for sensitive status changes (e.g., deceased) ensures accuracy and security, leveraging Nyiko's prior work on document verification.

 Add Export Functionality for Admin Analytics Data

- Assigned to: Mashabela
- Description: Enable admins to export analytics data (e.g., policies, claims, risky policyholders) as CSV or PDF from the analytics dashboard.
- Plan:
  - Extend the analytics dashboard (Mashabela's completed Task 5) with export buttons for key metrics (e.g., active policies, pending claims).
  - Build backend API endpoints to generate exportable files (e.g., CSV using csv-writer, PDF using pdfkit in Node.js or equivalent).
  - Ensure data aligns with analytics metrics (completed Task 5) and includes filters (Existing Task 3: search filters).
  - Secure exports with admin-only access (Existing Task 9: admin functionality).
  - Test export functionality for accuracy and file format compatibility.
- Priority: Medium
- Dependencies: Mashabela's completed Task 5 (analytics dashboard), Existing Task 3 (search filters), Existing Task 9 (admin functionality), New Task 3 (risky policyholders)

- Rationale: Exporting analytics data supports reporting and compliance, enhancing the analytics dashboard's utility.

Add Multi-Language Support for User-Facing Pages

- Assigned to: Mashaba
- Description: Implement multi-language support (e.g., English, Afrikaans, Zulu) for user-facing pages (login, policyholder dashboard, claim submission).
- Plan:
    - Identify user-facing pages (e.g., login from New Task 3, policyholder dashboard from Task 1, claim submission from Task 3).
    - Integrate a localization library (e.g., i18next for React, or equivalent for other frameworks) to manage translations.
    - Create translation files for at least three languages (English, Afrikaans, Zulu—confirm with stakeholders).
    - Add a language selector UI component and store user preferences (e.g., in local storage or user profile).
    - Test all pages for language switching and ensure consistent styling (Existing Task 5).
- Priority: Medium
- Dependencies: Existing Task 5 (page styling), New Task 3 (login frontend), Task 1 (policyholder dashboard), Task 3 (claim submission)
- Rationale: Multi-language support improves accessibility for diverse users, especially in a multilingual region like South Africa.

Add Automated Email Reminders for Policy Renewals

- Assigned to: Dipuo
- Description: Implement automated email reminders for policyholders when their policies are nearing renewal or expiration.
- Plan:
    - Add a renewal/expiration date field to the policy schema (Existing Task 1: policyholder system).
    - Create a backend scheduled job (e.g., using Node.js with node-cron or Python with APScheduler) to check policies daily and send reminders (e.g., 30 days, 7 days before expiration).

    - Integrate with an email service (e.g., SendGrid, used in Existing Task 8 and New Task 1) for sending reminders.
    - Design a customizable email template with policy details and renewal instructions.
    - Test reminder triggers and email delivery for various scenarios (e.g., expired, soon-to-expire policies).
- Priority: Medium
- Dependencies: Existing Task 1 (policyholder system), Existing Task 8 (notification logic), New Task 1 (insured person notifications)
- Rationale: Automated reminders improve policyholder engagement and prevent policy lapses, which is a common feature in insurance systems.